

# **Bioinformatics & Deep Learning — Course Notes**

**An open-access companion for the redesigned KB8029 course**

Arne Elofsson

2026-10-24

# Table of contents

|                                                                |           |
|----------------------------------------------------------------|-----------|
| <b>Welcome</b>                                                 | <b>10</b> |
| How to use this alongside the course . . . . .                 | 11        |
| <b>Course Schedule</b>                                         | <b>12</b> |
| Exam preparation . . . . .                                     | 19        |
| <b>Course Information &amp; Logistics</b>                      | <b>20</b> |
| Exam and requirements . . . . .                                | 20        |
| Bonus points on the exam . . . . .                             | 20        |
| The Q&A sessions . . . . .                                     | 21        |
| Labs . . . . .                                                 | 22        |
| Expected workload . . . . .                                    | 22        |
| Plagiarism . . . . .                                           | 24        |
| AI tools . . . . .                                             | 24        |
| Schedule . . . . .                                             | 24        |
| Administration . . . . .                                       | 24        |
| <b>Setup: VS Code and Google Colab</b>                         | <b>25</b> |
| Part 1: Install VS Code, git and the extensions . . . . .      | 25        |
| Part 2: Get the course notebooks into VS Code (once) . . . . . | 26        |
| Part 3: Run a lab notebook on Colab from VS Code . . . . .     | 26        |
| Part 4: Course notebooks in the browser . . . . .              | 27        |
| Troubleshooting . . . . .                                      | 27        |
| Further reading . . . . .                                      | 28        |
| <b>Session 0: Introduction to the Course</b>                   | <b>29</b> |
| What this course is about . . . . .                            | 29        |
| How a session works . . . . .                                  | 30        |
| Where to find everything . . . . .                             | 31        |
| People . . . . .                                               | 31        |
| Help us improve the course . . . . .                           | 32        |
| The rules, in brief . . . . .                                  | 32        |
| At SciLifeLab . . . . .                                        | 32        |
| Computers . . . . .                                            | 33        |
| Before Session 1 . . . . .                                     | 33        |
| Before Session 2 . . . . .                                     | 33        |
| What's next . . . . .                                          | 34        |
| <b>1 Session 1: Math Prerequisites</b>                         | <b>35</b> |
| 1.1 Learning goals . . . . .                                   | 36        |

|          |                                                                      |           |
|----------|----------------------------------------------------------------------|-----------|
| 1.2      | Vectors and matrices                                                 | 36        |
| 1.2.1    | Vectors                                                              | 36        |
| 1.2.2    | Dimensions and shapes                                                | 37        |
| 1.2.3    | The dot product                                                      | 37        |
| 1.2.4    | Matrix multiplication                                                | 38        |
| 1.2.5    | Normalisation                                                        | 40        |
| 1.2.6    | Eigenvectors and PCA, intuitively                                    | 41        |
| 1.2.7    | Further reading                                                      | 42        |
| 1.2.8    | Further watching                                                     | 43        |
| 1.3      | Calculus for machine learning                                        | 43        |
| 1.3.1    | Derivatives                                                          | 43        |
| 1.3.2    | Partial derivatives and the gradient                                 | 44        |
| 1.3.3    | Gradient descent                                                     | 45        |
| 1.3.4    | The chain rule                                                       | 45        |
| 1.3.5    | Further reading                                                      | 47        |
| 1.3.6    | Further watching                                                     | 47        |
| 1.4      | Probability and statistics                                           | 47        |
| 1.4.1    | A short recap: expectation and variance                              | 48        |
| 1.4.2    | Correlation: Pearson and Spearman                                    | 48        |
| 1.4.3    | Conditional probability                                              | 49        |
| 1.4.4    | Log probabilities and log-odds                                       | 50        |
| 1.4.5    | Likelihood                                                           | 51        |
| 1.4.6    | Entropy: average surprise                                            | 51        |
| 1.4.7    | p-values and hypothesis testing                                      | 52        |
| 1.4.8    | The multiple testing problem                                         | 53        |
| 1.4.9    | Worked example: RNA-seq differential expression and the volcano plot | 53        |
| 1.4.10   | Further reading                                                      | 55        |
| 1.4.11   | Further watching                                                     | 56        |
| 1.5      | The course notebook                                                  | 56        |
| 1.6      | What's next                                                          | 56        |
| <b>2</b> | <b>Session 2: Working like a computational biologist in Python</b>   | <b>57</b> |
| 2.1      | Learning goals                                                       | 58        |
| 2.2      | Notebooks: what they are and why we use them                         | 58        |
| 2.3      | Scientific Python objects                                            | 59        |
| 2.4      | Exploring a biological dataset                                       | 60        |
| 2.5      | Seeing the data                                                      | 61        |
| 2.6      | Asking quantitative questions                                        | 63        |
| 2.7      | Biological objects                                                   | 63        |
| 2.8      | Your own objects: classes                                            | 64        |
| 2.9      | Data coming from somewhere                                           | 66        |
| 2.10     | When you don't know how to do something                              | 67        |
| 2.11     | A tool outside Python: git                                           | 69        |
| 2.12     | Mini challenge: a small analysis, start to finish                    | 69        |
| 2.13     | The course notebook                                                  | 70        |
| 2.14     | The lab                                                              | 70        |
| 2.14.1   | Submitting this lab                                                  | 71        |

|          |                                                               |           |
|----------|---------------------------------------------------------------|-----------|
| 2.15     | Further watching                                              | 71        |
| 2.16     | What's next                                                   | 72        |
| <b>3</b> | <b>Session 3: Biological Databases</b>                        | <b>73</b> |
| 3.1      | Learning goals                                                | 73        |
| 3.2      | Why coordinated, worldwide databases                          | 74        |
| 3.2.1    | Further reading                                               | 74        |
| 3.3      | Two ideas you need for every database                         | 74        |
| 3.3.1    | Identifiers: names, accessions, versions and cross-references | 74        |
| 3.3.2    | Where does a record's information come from?                  | 76        |
| 3.3.3    | Further reading                                               | 77        |
| 3.4      | Following one protein through five questions                  | 77        |
| 3.4.1    | Where is the sequence?                                        | 77        |
| 3.4.2    | What is known about the protein?                              | 77        |
| 3.4.3    | Which family does it belong to?                               | 78        |
| 3.4.4    | Is there a structure?                                         | 78        |
| 3.4.5    | What does it interact with?                                   | 79        |
| 3.4.6    | The dossier                                                   | 79        |
| 3.4.7    | Further reading                                               | 80        |
| 3.5      | Searching: know your level, predict, then check               | 80        |
| 3.5.1    | Further watching                                              | 81        |
| 3.5.2    | Further reading                                               | 81        |
| 3.6      | The course notebook                                           | 81        |
| 3.7      | Check your understanding                                      | 82        |
| 3.8      | The lab                                                       | 82        |
| 3.8.1    | Submitting this lab                                           | 82        |
| 3.9      | What's next                                                   | 82        |
| <b>4</b> | <b>Session 4: Sequence Alignment</b>                          | <b>83</b> |
| 4.1      | Learning goals                                                | 83        |
| 4.2      | Why alignment needs an algorithm                              | 84        |
| 4.2.1    | Further reading                                               | 85        |
| 4.3      | Visualising alignments: dot plots                             | 85        |
| 4.3.1    | Further reading                                               | 85        |
| 4.4      | Dynamic programming: Needleman-Wunsch and Smith-Waterman      | 85        |
| 4.4.1    | Further reading                                               | 87        |
| 4.5      | Gap penalties: linear vs. affine                              | 88        |
| 4.5.1    | Further reading                                               | 88        |
| 4.6      | Substitution matrices: PAM and BLOSUM                         | 89        |
| 4.6.1    | Further reading                                               | 89        |
| 4.7      | Is the score significant?                                     | 89        |
| 4.7.1    | Further reading                                               | 90        |
| 4.8      | Multiple sequence alignment                                   | 90        |
| 4.9      | From alignment to tree                                        | 91        |
| 4.9.1    | Further reading                                               | 93        |
| 4.10     | Reading an alignment: entropy and sequence logos              | 93        |
| 4.10.1   | Further reading                                               | 96        |
| 4.11     | The course notebook                                           | 96        |

|          |                                                                           |            |
|----------|---------------------------------------------------------------------------|------------|
| 4.12     | Check your understanding . . . . .                                        | 96         |
| 4.13     | The lab . . . . .                                                         | 97         |
| 4.13.1   | Submitting this lab . . . . .                                             | 97         |
| 4.14     | What's next . . . . .                                                     | 97         |
| <b>5</b> | <b>Session 5: Advanced Database Search &amp; Profile Methods</b>          | <b>98</b>  |
| 5.1      | Learning goals . . . . .                                                  | 98         |
| 5.2      | The scaling problem, and BLAST's heuristic answer . . . . .               | 99         |
| 5.2.1    | Further reading . . . . .                                                 | 99         |
| 5.3      | Stage 1: seeding — turning the query into fast lookups . . . . .          | 99         |
| 5.4      | Stage 2: extension — growing a seed into a real alignment . . . . .       | 100        |
| 5.4.1    | Further watching . . . . .                                                | 101        |
| 5.5      | Stage 3: is this hit significant? E-values and bit scores . . . . .       | 101        |
| 5.5.1    | Further reading . . . . .                                                 | 103        |
| 5.6      | BLAST's ideas at modern database scale: MMseqs2 . . . . .                 | 103        |
| 5.7      | From one sequence to many: PSI-BLAST and the PSSM . . . . .               | 104        |
| 5.7.1    | Building a PSSM, step by step . . . . .                                   | 104        |
| 5.7.2    | Further reading . . . . .                                                 | 106        |
| 5.8      | Profile Hidden Markov Models: what a PSSM still can't represent . . . . . | 106        |
| 5.8.1    | Further watching . . . . .                                                | 107        |
| 5.8.2    | Further reading . . . . .                                                 | 107        |
| 5.9      | Worked example: a real profile-HMM hit on <i>HBB</i> . . . . .            | 108        |
| 5.10     | The course notebook . . . . .                                             | 108        |
| 5.11     | Check your understanding . . . . .                                        | 108        |
| 5.12     | The lab . . . . .                                                         | 108        |
| 5.12.1   | Submitting this lab . . . . .                                             | 109        |
| 5.13     | What's next . . . . .                                                     | 109        |
| <b>6</b> | <b>Session 6: Protein Structure</b>                                       | <b>110</b> |
| 6.1      | Learning goals . . . . .                                                  | 111        |
| 6.2      | Sequence becomes structure . . . . .                                      | 111        |
| 6.2.1    | Four levels, four computational tasks . . . . .                           | 111        |
| 6.2.2    | Haemoglobin: all four levels in one protein . . . . .                     | 112        |
| 6.2.3    | Does the sequence determine the structure? . . . . .                      | 113        |
| 6.2.4    | A structure as a matrix: distance and contact maps . . . . .              | 114        |
| 6.2.5    | Looking at structures: PyMOL . . . . .                                    | 114        |
| 6.2.6    | Further reading . . . . .                                                 | 115        |
| 6.3      | Comparing proteins in 3D . . . . .                                        | 115        |
| 6.3.1    | Sequence alignment versus structure alignment . . . . .                   | 115        |
| 6.3.2    | Measuring structural similarity: RMSD and TM-score . . . . .              | 116        |
| 6.3.3    | DALI: comparing distance maps . . . . .                                   | 116        |
| 6.3.4    | Foldseek: structure search at sequence-search speed . . . . .             | 117        |
| 6.3.5    | Further reading . . . . .                                                 | 118        |
| 6.4      | Domains, complexes and conformational states . . . . .                    | 118        |
| 6.4.1    | Structural domains . . . . .                                              | 118        |
| 6.4.2    | Quaternary structure, interfaces and ligands . . . . .                    | 119        |
| 6.4.3    | One protein, more than one structure: haemoglobin T and R . . . . .       | 119        |
| 6.4.4    | Further reading . . . . .                                                 | 120        |

|          |                                                                                                         |            |
|----------|---------------------------------------------------------------------------------------------------------|------------|
| 6.5      | From structure back to sequence prediction                                                              | 120        |
| 6.5.1    | Secondary structure as a prediction target                                                              | 120        |
| 6.5.2    | Predicting the whole fold                                                                               | 120        |
| 6.5.3    | Further reading                                                                                         | 121        |
| 6.6      | The course notebook                                                                                     | 121        |
| 6.7      | Check your understanding                                                                                | 121        |
| 6.8      | The lab                                                                                                 | 122        |
| 6.8.1    | Submitting this lab                                                                                     | 122        |
| 6.9      | What's next                                                                                             | 122        |
| <b>7</b> | <b>Session 7: Linear Models, Gradient Descent, the XOR Problem &amp; Secondary Structure Prediction</b> | <b>123</b> |
| 7.1      | Learning goals                                                                                          | 124        |
| 7.2      | Two pictures to keep in mind                                                                            | 124        |
| 7.3      | A profile score and a perceptron: the same weighted sum                                                 | 125        |
| 7.3.1    | Further reading                                                                                         | 127        |
| 7.3.2    | Further watching                                                                                        | 127        |
| 7.4      | Learning the weights from examples                                                                      | 127        |
| 7.4.1    | Further reading                                                                                         | 129        |
| 7.4.2    | Further watching                                                                                        | 129        |
| 7.5      | What one weighted sum can represent                                                                     | 129        |
| 7.6      | What one weighted sum can never represent: XOR                                                          | 130        |
| 7.6.1    | Further reading                                                                                         | 132        |
| 7.7      | Why a hidden layer helps                                                                                | 132        |
| 7.7.1    | The activation must be nonlinear                                                                        | 133        |
| 7.8      | Biological analogues                                                                                    | 134        |
| 7.8.1    | Interactions between positions: Bulyk et al.                                                            | 134        |
| 7.8.2    | Secondary structure prediction: PHD                                                                     | 136        |
| 7.9      | Optional: Chou-Fasman, GOR and the accuracy curve                                                       | 137        |
| 7.9.1    | Further reading                                                                                         | 145        |
| 7.10     | Representation matters as much as the model                                                             | 145        |
| 7.11     | Summary: the two pictures again                                                                         | 146        |
| 7.12     | Check your understanding                                                                                | 146        |
| 7.13     | The course notebook                                                                                     | 147        |
| 7.14     | The lab                                                                                                 | 147        |
| 7.14.1   | Submitting this lab                                                                                     | 147        |
| 7.15     | What's next                                                                                             | 147        |
| <b>8</b> | <b>Session 8: Machine Learning Foundations &amp; Practical PyTorch</b>                                  | <b>149</b> |
| 8.1      | Learning goals                                                                                          | 149        |
| 8.2      | Posing the problem                                                                                      | 150        |
| 8.2.1    | Supervised and unsupervised learning                                                                    | 150        |
| 8.2.2    | Regression vs. classification                                                                           | 150        |
| 8.2.3    | A running toy example: 3×3 pixel patterns                                                               | 153        |
| 8.3      | How do we know a model learned anything? Evaluation and generalisation                                  | 154        |
| 8.3.1    | Why three data sets?                                                                                    | 155        |
| 8.3.2    | Evaluating a classifier                                                                                 | 157        |
| 8.3.3    | Evaluating a regression model                                                                           | 158        |

|           |                                                                       |            |
|-----------|-----------------------------------------------------------------------|------------|
| 8.3.4     | How much can you trust one split?                                     | 160        |
| 8.3.5     | The biological trap: homologs on both sides of the split              | 161        |
| 8.4       | Towards training networks                                             | 163        |
| 8.4.1     | Connectivity decides the number of weights                            | 163        |
| 8.4.2     | From step functions to trainable activations                          | 166        |
| 8.4.3     | From binary to multi-class: softmax                                   | 167        |
| 8.4.4     | Practical PyTorch: writing a model as a class                         | 168        |
| 8.5       | Check your understanding                                              | 168        |
| 8.6       | The course notebook                                                   | 169        |
| 8.7       | The lab                                                               | 169        |
| 8.7.1     | Submitting this lab                                                   | 170        |
| 8.8       | What's next                                                           | 170        |
| <b>9</b>  | <b>Session 9: How a Neural Network Learns, and Why Training Fails</b> | <b>171</b> |
| 9.1       | Learning goals                                                        | 171        |
| 9.2       | From an untrained network to a trained network                        | 172        |
| 9.3       | How backpropagation finds every gradient                              | 174        |
| 9.3.1     | Further watching                                                      | 175        |
| 9.4       | Which loss to minimise                                                | 175        |
| 9.5       | Why training fails                                                    | 177        |
| 9.5.1     | Failure 1: every weight starts the same                               | 177        |
| 9.5.2     | Failure 2: deep sigmoid networks, and vanishing gradients             | 177        |
| 9.5.3     | Failure 3: the optimiser and the learning rate                        | 178        |
| 9.5.4     | Momentum and Adam: why an optimiser can beat plain SGD                | 179        |
| 9.6       | When a hidden layer pays off: epistasis in a protein                  | 179        |
| 9.7       | Why not make the network enormous?                                    | 181        |
| 9.8       | What can we do about overfitting?                                     | 183        |
| 9.8.1     | Early stopping: when to stop training                                 | 183        |
| 9.8.2     | Penalising large weights: L2 and L1                                   | 183        |
| 9.8.3     | Dropout                                                               | 184        |
| 9.9       | Is the improvement real?                                              | 185        |
| 9.9.1     | Further reading                                                       | 186        |
| 9.10      | Check your understanding                                              | 187        |
| 9.11      | The course notebook                                                   | 187        |
| 9.12      | The lab                                                               | 187        |
| 9.12.1    | Submitting this lab                                                   | 188        |
| 9.13      | What's next                                                           | 188        |
| <b>10</b> | <b>Session 10: CNNs, and How They Took Over Biology</b>               | <b>189</b> |
| 10.1      | Learning goals                                                        | 189        |
| 10.2      | Why convolution?                                                      | 190        |
| 10.2.1    | The problem with dense layers                                         | 190        |
| 10.2.2    | A small detector, reused everywhere                                   | 190        |
| 10.2.3    | Equivariance is not invariance                                        | 191        |
| 10.2.4    | Stacking convolutions: the receptive field                            | 192        |
| 10.2.5    | What actually happened: from LeNet to ResNet                          | 193        |
| 10.3      | What assumptions does a CNN build in?                                 | 195        |
| 10.3.1    | More weights are not better: dense vs. CNN on blood cells             | 195        |

|           |                                                                           |            |
|-----------|---------------------------------------------------------------------------|------------|
| 10.3.2    | Few labels: transfer learning on cancer histology . . . . .               | 195        |
| 10.3.3    | Which transformations should not matter? . . . . .                        | 196        |
| 10.4      | CNNs on biological sequences . . . . .                                    | 197        |
| 10.4.1    | Filters become motifs . . . . .                                           | 197        |
| 10.4.2    | Proteins: our own CNN finds the signal peptide . . . . .                  | 198        |
| 10.5      | Beyond sequences: a contact map is an image . . . . .                     | 200        |
| 10.6      | Check your understanding . . . . .                                        | 201        |
| 10.7      | The course notebook . . . . .                                             | 201        |
| 10.8      | The lab . . . . .                                                         | 201        |
| 10.8.1    | Submitting this lab . . . . .                                             | 202        |
| 10.9      | What's next . . . . .                                                     | 202        |
| <b>11</b> | <b>Session 11: Sequences, Language Models and SignalP</b>                 | <b>203</b> |
| 11.1      | Learning goals . . . . .                                                  | 203        |
| 11.2      | How can a model remember sequence context? . . . . .                      | 204        |
| 11.2.1    | A hidden state carried along the sequence . . . . .                       | 204        |
| 11.2.2    | LSTM: an additive memory path . . . . .                                   | 205        |
| 11.2.3    | Case study: which residues belong to a signal peptide? . . . . .          | 206        |
| 11.3      | Can we learn from sequences without labels? . . . . .                     | 208        |
| 11.3.1    | What a language model is . . . . .                                        | 209        |
| 11.3.2    | Attention, from scratch . . . . .                                         | 210        |
| 11.3.3    | The Transformer: attention plus four tricks . . . . .                     | 211        |
| 11.3.4    | What actually happened . . . . .                                          | 213        |
| 11.3.5    | Protein language models . . . . .                                         | 213        |
| 11.3.6    | Same residue, different context: contextual embeddings . . . . .          | 213        |
| 11.3.7    | Foundation models: pretrain once, reuse everywhere . . . . .              | 214        |
| 11.4      | Pretraining changes biological prediction . . . . .                       | 215        |
| 11.4.1    | SignalP: thirty years in three steps . . . . .                            | 215        |
| 11.4.2    | Reproducing the idea at classroom scale . . . . .                         | 216        |
| 11.5      | Check your understanding . . . . .                                        | 218        |
| 11.6      | The course notebook . . . . .                                             | 219        |
| 11.7      | The lab . . . . .                                                         | 219        |
| 11.7.1    | Submitting this lab . . . . .                                             | 219        |
| 11.8      | What's next . . . . .                                                     | 220        |
| <b>12</b> | <b>Session 12: AlphaFold and Generative Models of Protein Structure</b>   | <b>221</b> |
| 12.1      | Learning goals . . . . .                                                  | 221        |
| 12.2      | Where does long-range structural information come from? . . . . .         | 222        |
| 12.2.1    | The folding problem . . . . .                                             | 222        |
| 12.2.2    | Three generations of structure prediction . . . . .                       | 222        |
| 12.2.3    | Coevolution: long-range restraints from evolution . . . . .               | 223        |
| 12.3      | How AlphaFold2 combines the course's ideas . . . . .                      | 227        |
| 12.3.1    | The data flow . . . . .                                                   | 228        |
| 12.3.2    | Confidence: pLDDT and PAE . . . . .                                       | 233        |
| 12.4      | From single chains to interactions . . . . .                              | 234        |
| 12.4.1    | Complexes are harder . . . . .                                            | 234        |
| 12.4.2    | The ranking problem . . . . .                                             | 235        |
| 12.4.3    | When can a complex prediction be trusted? A practical heuristic . . . . . | 235        |

|                                                         |                                          |            |
|---------------------------------------------------------|------------------------------------------|------------|
| 12.5                                                    | Generative structure modelling           | 237        |
| 12.5.1                                                  | Diffusion in one page                    | 237        |
| 12.5.2                                                  | AlphaFold3: what changed                 | 237        |
| 12.5.3                                                  | Designing new proteins                   | 240        |
| 12.6                                                    | The course in one line                   | 241        |
| 12.7                                                    | Check your understanding                 | 241        |
| 12.8                                                    | The course notebook                      | 241        |
| 12.9                                                    | The lab                                  | 241        |
| 12.9.1                                                  | Submitting this lab                      | 242        |
| 12.10                                                   | What's next                              | 242        |
| <b>Capstone Project: Secondary Structure Prediction</b> |                                          | <b>243</b> |
|                                                         | Framing                                  | 243        |
|                                                         | The task                                 | 243        |
|                                                         | Training data                            | 244        |
|                                                         | Deliverables                             | 244        |
|                                                         | Grading                                  | 245        |
|                                                         | Timeline                                 | 246        |
|                                                         | Resources                                | 246        |
|                                                         | What's next                              | 246        |
| <b>References</b>                                       |                                          | <b>247</b> |
|                                                         | The open sources this book is built on   | 247        |
|                                                         | Papers and books                         | 247        |
|                                                         | Databases, datasets and software         | 256        |
|                                                         | Further reading and watching, by session | 262        |
|                                                         | Welcome                                  | 262        |
|                                                         | Setup                                    | 262        |
|                                                         | Session 1                                | 262        |
|                                                         | Session 2                                | 264        |
|                                                         | Session 3                                | 264        |
|                                                         | Session 4                                | 265        |
|                                                         | Session 5                                | 266        |
|                                                         | Session 6                                | 266        |
|                                                         | Session 7                                | 267        |
|                                                         | Session 8                                | 267        |
|                                                         | Session 9                                | 269        |
|                                                         | Session 10                               | 269        |
|                                                         | Session 11                               | 270        |
|                                                         | Session 12                               | 270        |
|                                                         | What isn't reused, and why               | 271        |

# Welcome

This is an open, interactive companion to the redesigned KB8029 Bioinformatics course. Rather than teaching classical bioinformatics and machine learning as two separate blocks, this book follows the course’s integrated session-by-session structure — see the [Course Schedule](#) for the full plan, or `course-redesign.md` in the repository root for the rationale behind it.

**This is not a copy of any textbook.** Earlier we used two copyrighted books (Magnus Ekman’s *Learning Deep Learning* for the machine-learning half and Jonathan Pevsner’s *Bioinformatics and Functional Genomics* for the classical half). Our lectures build on that earlier work, but nothing from either book is reproduced here. Every page is written from scratch, drawing on fully open sources — none restricted to non-commercial use — that cover the same material:

- [Dive into Deep Learning](#) (Zhang, Lipton, Li & Smola) — [CC BY-SA 4.0](#). Primary source for every machine-learning session.
- [Wikipedia](#) — [CC BY-SA 4.0](#). Conceptual and algorithmic depth for the bioinformatics sessions.
- [EMBL-EBI Training](#) — [CC BY 4.0](#). The practical, tool-use layer for databases and alignment.
- [bibook](#) (Lukas Käll) — [CC BY 4.0](#). Notebook code for the algorithmic bioinformatics sessions (alignment, MSA, and more).
- [NCBI Bookshelf](#) — public domain (US government work). BLAST and database practicalities.
- [RCSB PDB-101](#) — [CC BY 4.0](#) (educational tier only). Protein structure.
- [A Byte of Python](#) — [CC BY-SA 4.0](#). Python fundamentals and object-oriented programming.

Every page names which source(s) it draws on, and because of the CC BY-SA license, pages adapted from *Dive into Deep Learning* or Wikipedia are themselves shared here under CC BY-SA 4.0.

**Students are never pointed at raw source material directly** — Wikipedia in particular is written to be browsed outward via its own links, which makes “how far should I read?” an unanswerable question. Every session gets one bounded, authored page instead, with sources listed for further reading, never as an inline dependency.

**New to the course?** Start with [Session 0: Introduction to the Course](#): what the course covers, how a session works, and what to do before Session 1. The rules (exam, grading, labs, plagiarism, AI tools) and the contact details of the student office are on [Course Information & Logistics](#).

**Found an error, or something unclear?** The course is new and open access, and we know it is not perfect, so we appreciate all feedback. Students can use the [feedback](#)

[form](#) (university login, anonymous if you like); with a GitHub account you can open an [issue](#) (public); and everyone is welcome to email [arne@bioinfo.se](mailto:arne@bioinfo.se) with the subject “KB8029 feedback”. See [Help us improve the course](#).

## How to use this alongside the course

Most session pages have three parts:

1. **The idea** — a short, from-scratch explanation of the session’s core concept, written to match what’s covered in the lecture.
2. **The course notebook** — the session’s material in code, embedded with its saved output so you can read it here or run it yourself in Colab. Sessions with a lab also have a separate **lab notebook**, linked from the page’s lab section; you answer its questions in the lab quiz on Canvas.
3. **What’s next** — an explicit pointer to where this session’s ideas resurface later in the course. The redesign is built around several continuous threads (most centrally, secondary-structure prediction) rather than disconnected topics, and each page names the connection rather than leaving it implicit.

# Course Schedule

The course has twelve sessions and a capstone project. The table below lists, for each session, its page, its main reading, its biological case study and the size of its lab. Dates, times and rooms are in the Canvas schedule and calendar, which can also be subscribed to as <https://course.bioinfo.se/kb8029.ics>. How to get to SciLifeLab is on the [Session 0](#) page. See [Course Information & Logistics](#) for grading, lab policy, and how the course actually runs.

Three phases: bioinformatics foundations build up to profiles and protein structure (Sessions 3-6); a convergence zone (Sessions 7-9) where the classical material directly motivates and is resolved by neural networks; then a deep-learning arc (Sessions 10-12) that keeps extending the same secondary-structure-prediction thread.

**Every lab, regardless of the length noted below, is administered through Canvas, not this book** — the “Lab” column here just indicates how substantial each session’s lab is, not where to find or submit it. See [Course Information & Logistics](#) for the general lab-quizz policy (50% correct to pass, one redo chance, each lab due the evening before the next lecture) and the course’s Canvas page for each lab’s actual worksheet, deadline, and submission mechanism.

| # | Session                                                                                           | Primary reading (open, linked)                                                                   | Biological case study | Lab                             |
|---|---------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|-----------------------|---------------------------------|
| 0 | <b>Introduction to the course</b><br>— how the course works, where to find everything, setting up | —                                                                                                | —                     | None (setup: VS Code and Colab) |
| 1 | <b>Math prerequisites</b><br>— linear algebra, calculus, probability refresher                    | d2l.ai, <b>Preliminaries</b> (§2.3 Linear Algebra, §2.5 Calculus, §2.6 Probability & Statistics) | —                     | None (non-code problem set)     |

| # | Session                                                                                                          | Primary reading (open, linked)                                                                                                                                                                                                        | Biological case study                                                                                                                                                   | Lab            |
|---|------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| 2 | <b>Programming prerequisites</b><br>— NumPy/Matplotlib recap, notebooks, Biopython, SciPy, scikit-learn, PyTorch | <a href="#">A Byte of Python</a> ; <a href="#">d2l.ai Preliminaries</a><br>§2.1 Data Manipulation, §2.2 Preprocessing, §2.5 Automatic Differentiation; Biopython <a href="#">Tutorial and Cookbook</a>                                | Real published mouse-liver RNA-seq data (Becares et al. 2019) for a real volcano plot; six real human globin genes, fetched live via Bio.Entrez, for the Biopython demo | <b>Full</b>    |
| 3 | <b>Biological Databases</b>                                                                                      | Wikipedia: <a href="#">Biological database</a> , <a href="#">Sequence database</a> ; NCBI: <a href="#">GenBank overview</a> ; practice via EMBL-EBI Training's <a href="#">What is ENA?</a> and <a href="#">The UniProt databases</a> | The <i>HBB</i> (beta-globin) locus                                                                                                                                      | Light exercise |

| # | Session                                                                    | Primary reading (open, linked)                                                                                                                                                                                                                                       | Biological case study                                                                                                                                                                                                | Lab                                                                                                |
|---|----------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| 4 | <b>Sequence Alignment</b>                                                  | Wikipedia: <a href="#">Needleman-Wunsch algorithm</a> , <a href="#">Smith-Waterman algorithm</a> , <a href="#">Substitution matrix</a> (BLO-SUM/PAM); practice: EMBL-EBI <a href="#">Pairwise Sequence Alignment</a> and <a href="#">Multiple Sequence Alignment</a> | Aligning two homologous globins                                                                                                                                                                                      | <b>Full</b> (by-hand DP matrix + traceback, then parameter/E-value sensitivity, then a simple MSA) |
| 5 | <b>Advanced Database Search &amp; Profile Methods (BLAST, PSSMs, HMMs)</b> | Wikipedia: <a href="#">BLAST</a> (biotechnology), <a href="#">Hidden Markov model</a> , <a href="#">Position weight matrix</a> , <a href="#">Sequence profiling tool</a> ; NCBI: <a href="#">BLAST® Help</a> (public domain)                                         | A real BLAST search on an unknown sequence, followed by PSI-BLAST/a Pfam domain search on the same family — framed explicitly as: a profile/PSSM score is a weighted sum over positions, i.e. already a linear model | <b>Full</b> (BLAST and profile/HMM search)                                                         |

| # | Session                  | Primary reading (open, linked)                                              | Biological case study                                                                                                                                                  | Lab                 |
|---|--------------------------|-----------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| 6 | <b>Protein Structure</b> | Wikipedia: <a href="#">Protein structure</a> ; <a href="#">RCSB PDB-101</a> | Hemoglobin's 3D fold in PyMOL: secondary structure (helices, strands) and tertiary structure, the structural background for Session 7's secondary-structure prediction | <b>Full</b> (PyMOL) |

| # | Session                                                                                      | Primary reading (open, linked)                                                                                                                                                                                                                                                                                                 | Biological case study                                                                                                                                                                                                                                                                                                                                                                                                                | Lab         |
|---|----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| 7 | <b>Linear models, gradient descent, the XOR problem &amp; secondary structure prediction</b> | d2l.ai, <a href="#">Linear Neural Networks for Regression</a> ; Wikipedia <a href="#">Perceptron</a> , <a href="#">Perceptrons</a> (book), <a href="#">Protein secondary structure</a> ; Bulyk, Johnson & Church (2002), <i>Nucleic Acids Research</i> 30(5):1255-1261; Rost & Sander (1993), <i>J. Mol. Biol.</i> 232:584-599 | Formalize Session 5's profile score as a perceptron, and predict the secondary structure that Session 6 introduced; show its weights can be <i>learned</i> by gradient descent instead of derived from alignment statistics; two real-world punchlines — a zinc-finger transcription factor's non-additive binding (Bulyk et al.), and Rost & Sander's PHD method solving real secondary-structure prediction the same way, at scale | <b>Full</b> |

| # | Session                                                     | Primary reading (open, linked)                                                                      | Biological case study                                                                                                                                                                                                              | Lab                                              |
|---|-------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| 8 | <b>Machine learning foundations &amp; practical PyTorch</b> | d2l.ai, <a href="#">Linear Neural Networks for Classification</a> + <a href="#">Builders' Guide</a> | Regression vs. classification, evaluation metrics, homology-aware splits (MMseqs2); <b>subcellular localization</b> network defined in PyTorch                                                                                     | <b>Full</b> (evaluating a blood-cell classifier) |
| 9 | <b>How a neural network learns, and why training fails</b>  | d2l.ai, <a href="#">Multilayer Perceptrons</a>                                                      | Training a network on breast-cancer biopsies; why training fails (momentum and Adam); epistasis in the GB1 protein fitness landscape (when a hidden layer pays off); overfitting on protein localization; L1 and L2 regularization | <b>Full</b> (training loop on blood-cell images) |

| #  | Session                                                       | Primary reading (open, linked)                                               | Biological case study                                                                                                                                                                    | Lab                                          |
|----|---------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------|
| 10 | <b>CNNs, and how they took over biology</b>                   | d2l.ai, Convolutional Neural Networks + Modern Convolutional Neural Networks | ImageNet history, cancer histology with transfer learning and an image foundation model (DINOv2), biomedical imaging, DNA (DeepSEA), a learned signal-peptide filter, contact prediction | <b>Full</b> (CNNs on blood-cell images)      |
| 11 | <b>Sequences, language models &amp; SignalP</b>               | d2l.ai, Recurrent Neural Networks + Attention Mechanisms and Transformers    | RNN/LSTM, attention and the Transformer, the road to GPT and BERT, protein language models as foundation models; SignalP 1.0 → 6.0 and its few-shot gain, reproduced with ESM-2          | <b>Full</b> (RNNs and ESM embeddings)        |
| 12 | <b>AlphaFold &amp; generative models of protein structure</b> | Primary papers (AlphaFold 1-3, ColabFold, RFDiffusion, ProteinMPNN)          | From coevolution to AlphaFold; complexes (pDockQ, ipTM, ipSAE); AlphaFold3 and diffusion; the four regions of interaction prediction; protein design                                     | <b>Full</b> (AlphaFold Server, AlphaFold DB) |

| # | Session                                                 | Primary reading (open, linked) | Biological case study               | Lab                |
|---|---------------------------------------------------------|--------------------------------|-------------------------------------|--------------------|
| — | <b>Capstone Project: Secondary Structure Prediction</b> | —                              | Closes the loop opened at Session 6 | The project itself |

## Exam preparation

Past exams, kept as a study resource (not because this course repeats questions from them):

- [Exam Bioinformatics 2025](#)
- [ReExam Bioinformatics 2025](#)
- [Old exam 1](#)
- [Old exam 2](#)
- [Old exam 3](#)

# Course Information & Logistics

## **i** Note

**Sources for this page:** the course's own introduction slides from earlier years, updated for this version of the course. Dates, rooms, deadlines and this year's teaching assistants are on [Canvas](#), not here. For an overview of the course, how a session works, and where to find everything, start with [Session 0](#).

This page holds the rules of the course: what you need to do to pass, how the grade is set, and who to contact.

## Exam and requirements

The course has three mandatory parts:

- **The labs** are pass/fail, and must all be passed to pass the course (see [Labs](#) below). Doing them on time gives bonus points on the exam (see [Bonus points](#) below).
- **The project** is graded 0-100, and you need at least 40% to pass it. It has three graded parts: the code you have written, the final report, and an oral presentation of your code to one of the teaching assistants. See the [Capstone](#) page for the task and the grading criteria.
- **The exam** is a pen-and-paper exam, graded 0-100; you need at least 40% to pass it. You may bring up to five pages (single-sided) of your own notes.

The final grade is 70% from the exam and 30% from the project, and you need 50% in total to pass. The grade scale (not exactly the default one at SU) is:

| Grade       | A     | B     | C     | D     | E     | Fx    | F     |
|-------------|-------|-------|-------|-------|-------|-------|-------|
| Total score | ≥ 90% | ≥ 80% | ≥ 70% | ≥ 60% | ≥ 50% | ≥ 45% | < 45% |

## Bonus points on the exam

Up to 9 points are added to your score on the main exam (0-100), 3 for each of three things. For each, doing all of it gives 3 points, and every one you miss or do late takes away one point.

- **Questions before the lecture: up to 3 points.** Before each lecture from Session 2 on, send in on Canvas what you want explained: a question, or a topic you want more time on. One line is enough. The Canvas survey “Questions before the lecture” closes at 18:00 the day before the lecture, so that the teacher can read your questions and answer the most common ones at the start of the lecture.
- **Being at the lectures: up to 3 points.** Every lecture has a short live quiz on [quiz.bioinfo.se](http://quiz.bioinfo.se), which you answer on your phone after logging in with your university account. Answering it counts as being present (at least three quarters of its questions); whether your answers are right or wrong is never used in grading. Sessions 0 and 1 are one lecture, so there are 12 lectures. Each of the two [Q&A sessions](#) you attend (by answering its check-in question when you arrive, logged in) makes up for one missed lecture.
- **Labs on time: up to 3 points.** A lab is on time when you have passed its lab quiz and filled in its feedback survey by the due date (see [Labs](#)). A late lab still has to be passed to pass the course.

The bonus points count only on the main exam, not on a re-exam, and only once. If you fail the exam, a re-exam is normally offered a few weeks after the course; if you have already passed (grade E or higher), there is no retake to improve the grade. Past exams are on the [Schedule](#) page. Two question-and-answer sessions, one after Session 6 and one after Session 12, prepare you for the exam.

## The Q&A sessions

The Q&A sessions are not lectures. Rereading and listening to the same explanation again feel useful but do little for an exam; trying to recall what you know, and finding out what you cannot do yet, does much more. So each Q&A is a practice exam with immediate feedback:

- **Recall (7 min):** with books closed, write down everything you remember about two or three central topics.
- **Practice exam (37 min):** four exam-style questions, on paper, with the same aids as in the exam: bring your pages of notes.
- **Break (15 min).**
- **Marking in pairs (20 min):** you mark each other’s answers with the marking guide, so you see what a full answer looks like. The teaching assistants help.
- **What the room got wrong (30 min):** an anonymous poll shows which questions were hardest, and the teacher goes through those and your questions.
- **What to study now (8 min):** you write down your two weakest topics.

Nothing is handed in or graded, and the poll is anonymous. The only thing recorded is a logged-in check-in question when you arrive: attending a Q&A makes up for one missed lecture (see [Bonus points](#)). You can post questions anonymously during the session, or ask them on the Canvas forum before it.

## Labs

Most sessions have a lab (see the [Course Schedule](#)): a notebook in this book, with a short online quiz on Canvas.

- **Mandatory.** Every lab quiz must be passed, with at least 50% correct.
- **Due before the next lecture.** Each lab quiz is due at 23:59 the evening before the next lecture (the last one the evening before the exam; when the next lecture is the morning after the lab, you get one more day), so that you come to each lecture with the previous lab done; the dates are on Canvas.
- **Available as soon as published.** You do not have to wait for the lecture.
- **Feedback after each lab.** The last part of every lab is a short feedback survey on Canvas, due with the lab quiz. It is anonymous: Canvas records that you filled it in, but nobody can see what you wrote. It asks for concrete suggestions on the lab and the session's material (what was too long, what background was missing, errors, unclear instructions), so that we can improve the course. Sessions 0 and 1, which have no lab, have the same kind of survey, filled in at the end of their lecture. Errors and suggestions can also be sent at any time with the [feedback form](#).
- **One redo chance** if you fail, normally only the next time the course runs, and the lab may have changed by then.

## Expected workload

The course is full time for four weeks and two days: about 176 hours of work, at eight hours a day. The table below gives our estimate of where that time goes: each session has its lecture (three hours; Sessions 0 and 1 share one), reading the material and doing the lab, and then come the capstone project and the exam.

- **Reading and understanding** means studying the session page carefully, with its worked examples, practice problems, quiz and main videos, and working through the course notebook. Optional videos and boxes are not included.
- **Lab** is the total time for the lab, including its quiz on Canvas. Every lab is designed to fit into the scheduled three-hour lab afternoon, and most students need about two to two and a half hours. Optional extras in the lab notebooks are not included.

| Part of the course      | Lecture | Reading and understanding | Lab    | Total |
|-------------------------|---------|---------------------------|--------|-------|
| 0 Introduction          | 1 h     | 0.5 h                     | –      | 1.5 h |
| 1 Mathematics           | 2 h     | 7.5 h                     | no lab | 9.5 h |
| Setup, before Session 2 | –       | –                         | 0.5 h  | 0.5 h |
| 2 Python                | 3 h     | 4.5 h                     | 2.5 h  | 10 h  |
| 3 Biological databases  | 3 h     | 2.5 h                     | 2 h    | 7.5 h |

| Part of the course                                                                        | Lecture     | Reading and understanding | Lab           | Total          |
|-------------------------------------------------------------------------------------------|-------------|---------------------------|---------------|----------------|
| 4 Sequence alignment                                                                      | 3 h         | 3 h                       | 2.5 h         | 8.5 h          |
| 5 Database search and profiles                                                            | 3 h         | 3 h                       | 2.5 h         | 8.5 h          |
| 6 Protein structure                                                                       | 3 h         | 2.5 h                     | 2.5 h         | 8 h            |
| 7 Linear models and XOR                                                                   | 3 h         | 4.5 h                     | 1.5 h         | 9 h            |
| 8 Machine-learning foundations                                                            | 3 h         | 4.5 h                     | 2 h           | 9.5 h          |
| 9 Training neural networks                                                                | 3 h         | 4.5 h                     | 2 h           | 9.5 h          |
| 10 CNNs                                                                                   | 3 h         | 3.5 h                     | 2.5 h         | 9 h            |
| 11 Sequences and language models                                                          | 3 h         | 4.5 h                     | 2.5 h         | 10 h           |
| 12 AlphaFold                                                                              | 3 h         | 5 h                       | 2.5 h         | 10.5 h         |
| <b>All sessions</b>                                                                       | <b>36 h</b> | <b>50 h</b>               | <b>25.5 h</b> | <b>111.5 h</b> |
| Capstone project: introduction, help session, your own work, report and oral presentation |             |                           |               | 40 h           |
| Exam preparation, including the Q&A sessions                                              |             |                           |               | 20 h           |
| Written exam                                                                              |             |                           |               | 4 h            |
| <b>Whole course</b>                                                                       |             |                           |               | <b>175.5 h</b> |

These numbers are estimates from the amount of material, not measurements, and they depend a lot on your background. If you already know Python well, Session 2 goes quickly, and if the mathematics is new to you, Session 1 takes longer. The sessions take about 110 hours with the lectures, and the [capstone project](#) and the exam about 65 hours together.

## Plagiarism

Plagiarism has unfortunately been an issue in earlier years. You may already have signed a plagiarism policy form; if not, you will be given one. Please set an example of honesty for your peers, since not everyone has fully understood the rules. If you see a problem, or have a suggestion for how to handle it, talk to the teachers. In general we trust everyone: you are honest until proven otherwise.

Plagiarism is copying the work of others, whether from online or printed sources, your peers, or anyone else, without attributing it directly and explicitly. Discussing methods and approaches with other students is encouraged; submitting someone else's answers, code or text as your own is not.

- All written reports are checked for plagiarism.
- Plagiarism can lead to suspension from the university.

If you are unsure where the line is on a specific assignment, ask before you submit, not after.

## AI tools

You may use AI tools during the course, but not during the exam. The final report and the labs must not be written by AI tools, but you may use them as a "science buddy", for example to understand a method or to improve your report. Session 2's "How to use chatbots" section shows how to use them well.

## Schedule

The schedule is on Canvas. As far as possible, teaching is on Mondays, Wednesdays and Fridays, with lectures 09:00-12:00 and labs 13:00-16:00. What each session covers is on the [Course Schedule](#) page.

## Administration

The course is administered by the [Chemistry Section](#) at Stockholm University:

Kemiska sektionens kansli & studentexpedition  
Svante Arrhenius väg 16A, 114 18 Stockholm  
Email: kemi@su.se  
Opening hours: Mon-Wed 9.00-11.30 and 12.30-15.00  
Phone hours: Wed 9.00-11.30 and 12.30-15.00

# Setup: VS Code and Google Colab

All the notebooks and labs in this course run on **Google Colab**, Google's free service for running Python notebooks on its machines, some of which have a GPU. The **labs** are done in **VS Code** on your own laptop, with the notebook running on a Colab machine through Google's Colab extension: this is how notebooks are used in research, and the labs are where you get used to it. The **course notebooks** that go with each session page can also be opened directly in Colab in the browser. Either way, nothing has to be installed on your laptop apart from VS Code and git, and every student works in the same environment.

| What you need                                                                                             | Why                                                                                                                             |
|-----------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| A <b>Google account</b> (a free one is enough)                                                            | Colab runs your code on Google's machines                                                                                       |
| <b>VS Code</b> on your laptop, with the extensions <b>Jupyter</b> , <b>Python</b> and <b>Google Colab</b> | To open, edit and run the lab notebooks                                                                                         |
| <b>git</b>                                                                                                | To get all the course notebooks in one go and keep them up to date, and to keep your own work under version control (Session 2) |

Every lab notebook has an **Open in VS Code** link (on the session page, in Canvas and in the lab quiz), and every course notebook an **Open in Colab** badge. All notebooks fetch their data from the course's GitHub repository, so they run the same way everywhere. Do Parts 1 and 2 before the first lab in Session 2, and check that a notebook runs, so that the lab time is spent on the lab.

## Part 1: Install VS Code, git and the extensions

1. Install [VS Code](#) and [git](#) if you do not already have them.
2. In VS Code, open the Extensions view (Ctrl+Shift+X, or Cmd+Shift+X on a Mac) and install **Python** and **Jupyter** (both by Microsoft) and **Google Colab** (by Google).
3. Check that you can sign in to Google in a browser with the account you want to use for the course.

## Part 2: Get the course notebooks into VS Code (once)

1. Click **Clone the course notebooks into VS Code**. Your browser asks whether to open VS Code: allow it. (Or, in VS Code: Command Palette, Ctrl+Shift+P or Cmd+Shift+P → **Git: Clone** → <https://github.com/ElofssonLab/kb8029-book.git>.)
2. Choose where to save the folder kb8029-book (about 100 MB), and open it when VS Code offers to. The notebooks are in its notebooks folder.
3. Before each lab, get the latest version: Command Palette → **Git: Pull**. The teachers sometimes fix a lab shortly before it is used.

## Part 3: Run a lab notebook on Colab from VS Code

Google's **Colab** extension for VS Code (released November 2025) lets a notebook that is open in VS Code run on a Colab machine:

1. Click the lab's **Open in VS Code** link. The first time, the page that opens asks for the full path of your kb8029-book folder (for example C:\Users\you\kb8029-book or /Users/you/kb8029-book) and remembers it; after that, its button opens the notebook straight in VS Code. You can also open it yourself: in VS Code press Ctrl+P (Cmd+P on a Mac) and type the notebook's name, for example session04-lab.ipynb.
2. Click **Select Kernel** (top right) → **Colab** → **Auto Connect** (or choose a runtime with a GPU), and sign in with your Google account when asked.
3. Run the first cells. Most notebooks start with an install cell, which installs the few packages that Colab does not already have (for example Biopython); this takes a few seconds each time you connect to a new machine.
4. When you need a GPU (mainly Sessions 9-12), check that you got one:

```
import torch
print(torch.cuda.is_available(), torch.cuda.get_device_name(0) if torch.cuda.is_ava
```

Your changes are saved in your own copy of the notebook in kb8029-book. If **Git: Pull** complains that a file you changed would be overwritten, keep your version under another name (for example session04-lab-mine.ipynb) and pull again.

Things to know:

- **The code runs on Google's machine.** It cannot see files on your laptop, even though the notebook file is open in VS Code. The course notebooks download what they need from the course's GitHub repository, so they work anyway. For your own files, upload them, or mount Google Drive (Command Palette → **Colab: Mount Google Drive to Server...**).

- **Colab machines are temporary.** Everything on them, including installed packages and files the notebook wrote, disappears when the session ends. Your notebook itself, with its code and output, is saved on your laptop by VS Code. Save other results you want to keep (download them, or write them to Drive).
- **The free GPU is limited.** A session is closed after a period of inactivity or when your quota is used up, and a GPU is not always available. Ask for a GPU only when the notebook needs one, and when you are done, release the machine with **Colab: Remove Server**.
- **No terminal tools.** Everything in the labs is done from Python: the sequence searches and alignments in Sessions 4 and 5 call EMBL-EBI's web services (HMMER, Clustal Omega, InterPro), and in Session 6 PyMOL runs inside the notebook and structure searches go to the Foldseek web service.

## Part 4: Course notebooks in the browser

The course notebooks that go with each session page have an **Open in Colab** badge. Click it, choose **Runtime → Change runtime type → T4 GPU** if the notebook needs a GPU, and run the cells. This works from any computer. **File → Save a copy in Drive** keeps your own copy with your changes; without it, your edits are lost when you close the tab. You can also open a course notebook from your kb8029-book folder in VS Code, exactly as a lab.

## Troubleshooting

**The Open in VS Code button does nothing, or VS Code says the file does not exist.** Check the folder path on the page (it must be the kb8029-book folder itself), and that the clone in Part 2 worked. Opening the notebook by name with Ctrl+P always works.

**VS Code does not offer Colab under Select Kernel.** Check that the Google Colab extension is installed and enabled in a *local* VS Code window, and reload the window (Command Palette → **Developer: Reload Window**).

**ModuleNotFoundError.** The runtime is new and the install cell has not been run in it. Run the notebook from the top.

**FileNotFoundError for a file you can see on your laptop.** The code runs on Colab, not on your laptop: upload the file, or read it from Drive.

**No GPU, or torch.cuda.is\_available() is False.** You were given a machine without a GPU, or the free GPU quota is used up for now. Choose a GPU runtime explicitly, try again later, or run the notebook on the CPU: the course notebooks are small enough for that, only slower.

**The connection drops in the middle of a long cell.** Colab closes idle and long-running free sessions. Reconnect, run the notebook from the top, and save intermediate results to Drive if a step takes long.

## Further reading

- Google Developers Blog, [Google Colab is coming to VS Code](#) (2025)
- Google, [Colab FAQ](#)

# Session 0: Introduction to the Course

## **i** Note

**Sources for this page:** the course's own introduction slides from earlier years, updated for this version of the course. Dates, rooms, deadlines and this year's teaching assistants are on Canvas, not here.

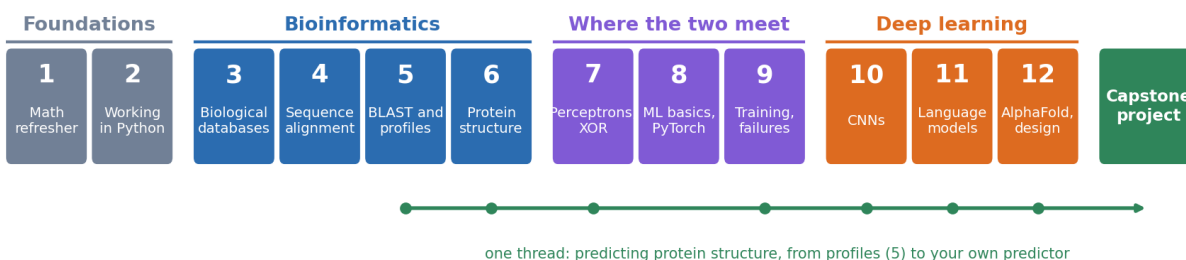
## Reproducibility

**How many bioinformaticians does it take to change a light bulb?**

One, but first they write a pipeline for it, and then it only works on their own laptop.

## What this course is about

KB8029 brings two subjects together: **bioinformatics**, the computational analysis of biological sequences and structures, and **machine learning**, in particular the neural networks behind today's methods. They are not taught as two separate halves. The classical bioinformatics methods motivate the machine learning, and the machine learning is applied to the same biological problems.



Every session: a page in this book, a notebook, a lecture with live quizzes and discussion notebooks, and (most sessions) a lab submitted in Canvas.

Figure 1: The course at a glance. Sessions 1-2 refresh the mathematics and programming the rest of the course uses. Sessions 3-6 cover classical bioinformatics: databases, alignment, database searches with profiles, and protein structure. In Sessions 7-9 the two subjects meet: a profile score turns out to be a simple neural network, and the networks are trained properly. Sessions 10-12 cover modern deep learning, up to AlphaFold. One problem, predicting protein structure from sequence, runs through the second half and ends in your capstone project.

By the end of the course you should be able to

- find, retrieve and compare biological sequences and structures using the main databases and tools;
- explain how the classical methods work: alignment, substitution matrices, profiles and database searches;
- build, train and evaluate neural networks in PyTorch, and judge whether a result is real;
- explain how modern deep-learning methods for proteins work, from convolutional networks and protein language models to AlphaFold;
- design, train and evaluate your own secondary-structure predictor (the [capstone project](#)).

The full plan is on the [Course Schedule](#) page.

## How a session works

Each of the twelve sessions has the same parts:

| Part                       | Where                                                           | What it is for                                                                     |
|----------------------------|-----------------------------------------------------------------|------------------------------------------------------------------------------------|
| <b>The page</b>            | this book, e.g. <a href="#">Session 1</a>                       | the material of the session, with self-study questions and a short quiz at the end |
| <b>The course notebook</b> | this book, linked from each page, with an “Open in Colab” badge | the same material in code; read it, then run and change it                         |

| Part                | Where                                                                     | What it is for                                                                                                      |
|---------------------|---------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| <b>The lecture</b>  | on site                                                                   | explanations, live coding, and short <b>discussion problems</b> you work on in groups (notebooks opened by QR code) |
| <b>Live quizzes</b> | <a href="https://quiz.bioinfo.se">quiz.bioinfo.se</a> , on your phone     | quick questions during the lecture, to check what has landed                                                        |
| <b>The lab</b>      | a separate lab notebook in this book, questions on <a href="#">Canvas</a> | most sessions; hands-on work with real data, submitted as a short online quiz                                       |

How much time each session takes, split into reading and lab, is in the table [Expected workload](#).

Read the page before the lecture: the lectures assume you have seen the material once, and use the time for the parts that are hard to learn alone. **Lectures are not recorded.**

## Where to find everything

| What                                                                       | Where                                                                   |
|----------------------------------------------------------------------------|-------------------------------------------------------------------------|
| Course pages and notebooks (this book)                                     | <a href="https://course.bioinfo.se">course.bioinfo.se</a>               |
| All course pages as one PDF                                                | <a href="#">KB8029-course-notes.pdf</a>                                 |
| Schedule, rooms, deadlines, announcements, lab quizzes, discussion forum   | <a href="#">Canvas</a> (the course appears there once you are admitted) |
| Live quizzes during lectures                                               | <a href="https://quiz.bioinfo.se">quiz.bioinfo.se</a>                   |
| Setting up VS Code and Colab                                               | <a href="#">Setup</a>                                                   |
| The session-by-session plan                                                | <a href="#">Course Schedule</a>                                         |
| Exam, grading, labs, plagiarism, AI tools, weekly schedule, student office | <a href="#">Course Information &amp; Logistics</a>                      |
| The capstone project                                                       | <a href="#">Capstone project</a>                                        |
| Past exams                                                                 | <a href="#">Exam preparation</a>                                        |
| Further reading for the whole course                                       | <a href="#">References</a>                                              |

## People

The course has two teachers, one for machine learning and one for bioinformatics, and teaching assistants who run the lab sessions. This year's teachers and teaching

assistants are listed on Canvas.

Ask questions in the Canvas discussion forum, where everyone can see the answers. The teaching assistants help during the scheduled lab hours; do not expect immediate help at other times.

## Help us improve the course

Everything in this course, the pages, the notebooks, the labs and the slides, is open access: written from scratch and free for anyone to read and reuse. This is also the first year of this version of the course. We have written it with care, but it is not perfect. You will find errors, explanations that do not work for you, and notebooks that break when a library changes. You are the first readers, so you are the ones who find them, and we really appreciate it when you tell us. Every message is read by the teachers, and each one makes the course better, for you and for the students after you.

- **With a university account:** use the [feedback form](#) and log in with your university account. You can choose the session, say where the problem is, and send it anonymously if you like. The form is also linked at the bottom of every page.
- **With a GitHub account:** open an [issue on the course's GitHub page](#). Issues are public, so everyone can see the problem, the discussion and the fix; they are not anonymous. This works well for errors in the text and for notebooks that do not run.
- **Without either:** email [arne@bioinfo.se](mailto:arne@bioinfo.se) with the subject "KB8029 feedback".

Feedback never affects your grades. The form is for improving the material; questions about the content belong in the Canvas discussion forum, where everyone sees the answer. The official course evaluation at the end of the course is separate, so please fill that in as well.

## The rules, in brief

To pass, you need to pass all the **labs**, the **capstone project** (a secondary-structure predictor of your own: a Colab notebook, a short report, and an oral presentation of your code to a teaching assistant) and the **exam**. Your work must be your own, and AI tools may help you understand but must not write your labs or report. The grading, the grade scale, and the rules on plagiarism and AI tools are on the [Course Information & Logistics](#) page.

## At SciLifeLab

The lectures and labs are at SciLifeLab in Solna:

- **Address:** Science for Life Laboratory (SciLifeLab), Tomtebodavägen 23A, 171 65 Solna ([map](#)).
- **Public transport:** bus 3 (or others) to Tomtebodavägen-Karolinska inst. (Solna).
- **Rooms:** most lectures and labs are in Air&Fire (Gamma 2); a few events are in other rooms in Gamma 2, Gamma 7 or Alfa 2. The room of each lecture and lab is in the Canvas schedule.
- **Teachers' offices:** Alfa, floor 6.
- **Directions** from the research group's page: <https://bioinfo.se/contact/>.

You need a SciLifeLab access card to enter and leave the building; when and where to collect it is announced by email or on Canvas.

- Use your card **every time** you pass the speed gates, on the way in and on the way out.
- The gates let **one person through at a time**. To let a visitor in, use your card twice, with a few seconds in between, or the alarm goes off.
- Never force your way through a gate.

## Computers

All notebooks and labs run on **Google Colab**, which gives free access to a GPU. The labs are done in **VS Code** on your laptop, with the notebook running on a Colab machine; the course notebooks can also be opened in Colab in the browser. So the only things you need are a laptop with VS Code and git, and a Google account. The [Setup](#) page takes you through it. Even the structure viewing in Session 6 is done with PyMOL inside a notebook, so nothing else has to be installed.

If you do not have a laptop that can do this, tell us now, not on the day of a lab.

## Before Session 1

1. Accept the course invitation on [Canvas](#), and check the schedule there. To get the lectures, labs and lab deadlines in your own calendar, subscribe to <https://course.bioinfo.se/kb8029.ics> (Google Calendar: Other calendars → From URL; Outlook: Add calendar → Subscribe from web); it updates itself when the schedule changes.
2. Open [quiz.bioinfo.se](https://quiz.bioinfo.se) on your phone and check that you can log in.
3. Read the [Session 1](#) page, the mathematics refresher, and try its practice problems.

## Before Session 2

Session 2 has the first lab, and it runs on Colab. Before it, work through the [Setup](#) page: install VS Code, git and the Google Colab extension, clone the course notebooks into VS Code, and make sure you can run a lab notebook on Colab from VS Code.

Setting up takes about half an hour, and it is easier to sort out problems before the lab than during it.

## **What's next**

[Session 1](#) refreshes the mathematics the course relies on: vectors and matrices, derivatives and gradients, and probability, likelihood and entropy. [Session 2](#) does the same for programming in Python.

# 1 Session 1: Math Prerequisites

## **i** Note

**Sources for this page:** d2l.ai, *Dive into Deep Learning*, [Preliminaries](#) (§2.3 Linear Algebra, §2.5 Calculus, §2.6 Probability and Statistics) — CC BY-SA 4.0; Wikipedia, [Dot product](#), [Norm \(mathematics\)](#), [Matrix norm](#), [Matrix multiplication](#), [Transpose](#), [Cross product](#), [Unit vector](#), [Eigenvalues and eigenvectors](#), [Principal component analysis](#), [Tensor](#), [Gradient](#), [Chain rule](#), [Bayes' theorem](#), [P-value](#), [Student's t-test](#), [Multiple comparisons problem](#), [False discovery rate](#), [Volcano plot \(statistics\)](#), [Exponential function](#), [Logarithm](#), [Sigmoid function](#), [Integral](#), [Variance](#), [Standard deviation](#), [Normal distribution](#) — CC BY-SA 4.0. Text below is written from scratch, not copied, following these ideas; standard results (the geometric dot-product formula, the chain rule, Bayes' theorem, the t-test formula) are stated in the conventional way found in any of these sources. The volcano-plot figure uses simulated data, generated for this book.

This session is not “all the mathematics you might need”. It is the mathematical vocabulary we use in this course, worked through with real numbers, and each idea comes with a pointer to where it shows up later. This session has no programming exercises and no lab; programming starts in Session 2, whose lab is the first of the course. The notebook linked at the bottom nevertheless lets you check this session's answers in code. You set up VS Code and Colab ([Setup](#)) before Session 2, whose lab is the first to use it.

The session has three parts:

1. **Vectors and matrices:** vectors, dimensions and shapes, the dot product, matrix multiplication (and the transpose), normalisation, and an intuitive look at eigenvectors and PCA.
2. **Calculus for machine learning:** derivatives, partial derivatives, the gradient, gradient descent and the chain rule.
3. **Probability and statistics:** a short recap, correlation (Pearson and Spearman), conditional probability, logs and log-odds, likelihood, p-values and multiple testing, and an RNA-seq example.

Each part ends with practice problems, a quick self-check and one short video (not made by us). The other videos listed there are optional: most of this material will be familiar, and they are there if you want to refresh a topic.

## 1.1 Learning goals

By the end of this session you should:

- work comfortably with vectors, matrices, dot products and dimensions;
- understand derivatives, gradients and why gradient descent works;
- understand conditional probability and how  $P(A \mid B)$  differs from  $P(B \mid A)$ ;
- understand logs, likelihood and why bioinformatics uses log-scores;
- understand entropy as average surprise, and cross-entropy as its link to likelihood;
- understand p-values and why multiple testing requires FDR control.

## 1.2 Vectors and matrices

A more villainous definition

“It’s a mathematical term, represented by an arrow with both direction and magnitude. Vector! That’s me, because I commit crimes with both direction and magnitude.” — Vector, in *Despicable Me* (2010)

He’s not wrong: that’s the geometric picture of a vector this session builds on.

### 1.2.1 Vectors

A vector is just an ordered list of numbers. You already know how to add two vectors (add the corresponding entries) and how to scale one (multiply every entry by the same number). What needs a closer look is how big a vector is. For a single number,  $|x|$  measures its distance from zero, that is, its size regardless of sign. Vectors and matrices generalise this idea, just as they generalise everything else in this session.

For a vector, the natural generalisation is its **length** (also called its **norm**), written  $\|\mathbf{v}\|$ . It is a direct extension of the Pythagorean theorem to more than two dimensions:

$$\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2}$$

**Worked example.** For  $\mathbf{v} = (3, 4)$ :

$$\|\mathbf{v}\| = \sqrt{3^2 + 4^2} = \sqrt{25} = 5$$



Figure 1.1: 5 is literally the hypotenuse of a 3-4-5 right triangle.

### 1.2.2 Dimensions and shapes

The number of entries in a vector is its **dimension**. The vector  $(3, 4)$  is 2-dimensional, and the 20-number amino acid composition of a protein is a 20-dimensional vector. A matrix is a rectangular grid of numbers, and its **shape** is its number of rows and columns, so a matrix with  $n$  rows and  $m$  columns has shape  $(n, m)$ , also written  $n \times m$ . For everything you will do in NumPy and PyTorch, one habit matters more than any formula: **always know the shape of every matrix and vector**. In biology, a matrix is almost always a data table, as in these two examples:

- **Gene expression.** Measure 20,000 genes in 50 patient samples, and you have a matrix of shape  $(50, 20000)$ . Row  $i$  is one sample's expression profile; column  $j$  is one gene across all the samples.
- **Protein features.** Describe 1,000 proteins by their amino acid composition (the fraction of each of the 20 amino acids), and you have a matrix  $X$  of shape  $(1000, 20)$ : one row per protein, one column per feature.

Vectors and matrices are the first two members of a bigger family called **tensors**. A single number is a 0-dimensional tensor, a vector is 1-dimensional and a matrix is 2-dimensional, and a tensor can continue to 3 dimensions (a stack of matrices) or more. A batch of 32 protein sequences of length 100, with each position one-hot encoded over 20 amino acids, is therefore a tensor of shape  $(32, 100, 20)$ . Session 2 introduces tensors properly, as the one data structure PyTorch uses for everything, and the shape rules below carry over unchanged.

### 1.2.3 The dot product

Once we have vectors, matrices and tensors, we need to calculate with them, that is, apply **operations** to them. The first operation we discuss is the **dot product**:

$$\mathbf{w} \cdot \mathbf{x} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

The dot product simply multiplies the corresponding entries and adds everything up. It is still worth naming, because it shows up constantly. It is a **weighted sum**, in which each  $x_i$  contributes to the total in proportion to its weight  $w_i$ .

**Worked example.** For  $\mathbf{w} = (2, -1)$  and  $\mathbf{x} = (1, 4)$ :

$$\mathbf{w} \cdot \mathbf{x} = (2)(1) + (-1)(4) = 2 - 4 = -2$$

Besides this arithmetic definition, the dot product has a geometric meaning that is worth keeping in mind:

$$\mathbf{w} \cdot \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos \theta$$

Here  $\theta$  is the angle between the two vectors and  $\|\cdot\|$  is the length defined above. The formula gives the *sign* of a dot product for free, without computing anything. The dot

product is positive when the angle between the vectors is less than  $90^\circ$  (they point in broadly the same direction), zero when they are exactly perpendicular, and negative when the angle exceeds  $90^\circ$  (they point in broadly opposite directions). In the worked example above,  $\mathbf{w}$  and  $\mathbf{x}$  are about  $103^\circ$  apart, just past perpendicular, which is why their dot product came out negative:

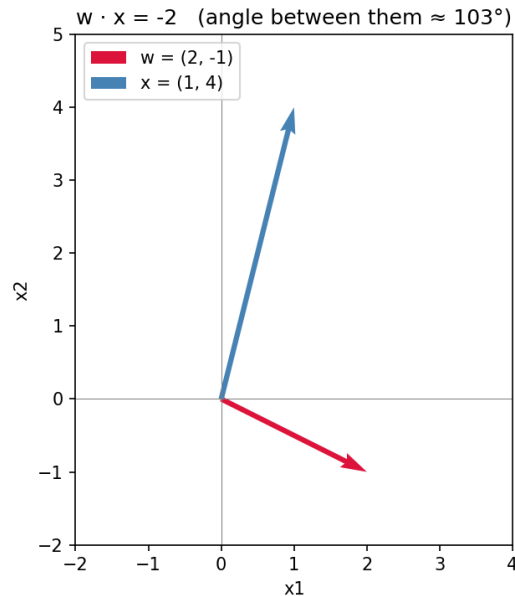


Figure 1.2: The vectors  $\mathbf{w} = (2, -1)$  and  $\mathbf{x} = (1, 4)$  are about  $103^\circ$  apart, so their dot product is negative.

Why does this one computation deserve so much attention before the course has properly started? The reason is that it reappears, unchanged, in two very different-looking places later in the course. A **profile score** (Session 5), the standard way of asking “how well does this sequence match a known family?”, multiplies each position’s observed value by a position-specific weight and sums the results. A **perceptron** (Session 7), the simplest kind of artificial neuron, computes its output in the same way, as a dot product between a weight vector and an input vector. When we get there, you will therefore already know both the arithmetic and the geometric intuition, and the only new part will be what to *do* with the result.

#### 1.2.4 Matrix multiplication

Matrix multiplication extends the dot product from one pair of vectors to many. A matrix can be read as a stack of vectors (its rows, or equally its columns), so a matrix-vector product is simply a batch of dot products, one per row of the matrix. For example,

$$\begin{pmatrix} 2 & -1 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 4 \end{pmatrix} = \begin{pmatrix} (2)(1) + (-1)(4) \\ (0)(1) + (3)(4) \end{pmatrix} = \begin{pmatrix} -2 \\ 12 \end{pmatrix}$$

The first entry of the result is the dot product from the worked example above, and the second is a new dot product between the second row of the matrix and the same vector.

This matters as soon as we have real data. Take a classifier that predicts, say, whether a protein is secreted. It has one weight per feature, so its weight vector  $\mathbf{w}$  has shape  $(20,)$ . The product  $X\mathbf{w}$  then computes the dot product of every row of  $X$  with  $\mathbf{w}$ :

$$\underbrace{X}_{(1000, 20)} \underbrace{\mathbf{w}}_{(20,)} = \underbrace{X\mathbf{w}}_{(1000,)}$$

**One matrix-vector product gives 1,000 scores, one per protein,** and no loop is needed. This is how Session 7's perceptron and Session 8's classifiers are written: every example is stacked as a row of  $X$ , and the scores are  $X\mathbf{w}$ .

Whether two matrices can be multiplied depends only on their shapes. **The inner dimensions must match, and the outer ones give the shape of the result:**

$$(n \times m)(m \times k) \rightarrow (n \times k)$$

- With three classes instead of one (say secreted, nuclear or cytoplasmic, as in Session 8), the weights become a matrix  $W$  of shape  $(20, 3)$ , and  $XW$  has shape  $(1000, 3)$ , three scores per protein.
- A weight vector of the wrong length, say 21, cannot be multiplied, because  $(1000 \times 20)(21)$  has mismatched inner dimensions. PyTorch then stops with the error message `size mismatch, got input (1000), mat (1000x20), vec (21)`, and NumPy gives a similar complaint about mismatched dimensions. This is the most common error in this course's code, and reading the shapes is how you fix it.

**Worked example.** Three proteins, two features each (say, the fraction of hydrophobic residues and of charged residues), and a weight vector  $\mathbf{w} = (2, -1)$ :

$$X = \begin{pmatrix} 0.5 & 0.1 \\ 0.2 & 0.4 \\ 0.3 & 0.3 \end{pmatrix}, \quad X\mathbf{w} = \begin{pmatrix} (0.5)(2) + (0.1)(-1) \\ (0.2)(2) + (0.4)(-1) \\ (0.3)(2) + (0.3)(-1) \end{pmatrix} = \begin{pmatrix} 0.9 \\ 0.0 \\ 0.3 \end{pmatrix}$$

Shapes:  $(3, 2)$  times  $(2,)$  gives  $(3,)$ , one score per protein.

**The transpose.** The transpose  $X^\top$  swaps rows and columns. Entry  $(i, j)$  of  $X$  becomes entry  $(j, i)$  of  $X^\top$ , so a  $(1000, 20)$  matrix becomes  $(20, 1000)$ :

$$\begin{pmatrix} 0.5 & 0.1 \\ 0.2 & 0.4 \\ 0.3 & 0.3 \end{pmatrix}^\top = \begin{pmatrix} 0.5 & 0.2 & 0.3 \\ 0.1 & 0.4 & 0.3 \end{pmatrix}$$

You will meet the transpose constantly, because it is how shapes are made to fit:

- **The dot product** is often written  $\mathbf{w}^\top \mathbf{x}$ , a row vector times a column vector:  $(1 \times m)(m \times 1) \rightarrow (1 \times 1)$ , a single number.
- $X^\top X$  has shape  $(20, 1000)(1000, 20) \rightarrow (20, 20)$ . It compares every feature with every other feature, summed over all proteins. For data with the column means subtracted,  $X^\top X / (n - 1)$  is the **covariance matrix** of the features, the matrix PCA works on (below).
- **Transposing a product reverses the order:**  $(AB)^\top = B^\top A^\top$ . The shapes confirm it.  $AB$  is  $(n \times k)$ , so its transpose must be  $(k \times n)$ , and only  $B^\top A^\top$ ,  $(k \times m)(m \times n)$ , gives that.

### 1.2.5 Normalisation

The length of a vector often gets in the way when we only care about its direction. Dividing a vector by its own length removes it and produces a **unit vector**, which points in the same direction but has length exactly 1:

$$\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

**Worked example.** For  $\mathbf{w} = (2, -1)$  from above, the length formula gives  $\|\mathbf{w}\| = \sqrt{5} \approx 2.236$ , so

$$\hat{\mathbf{w}} = \left( \frac{2}{\sqrt{5}}, \frac{-1}{\sqrt{5}} \right) \approx (0.894, -0.447)$$

Normalising removes the factor  $\|\mathbf{w}\|$  from the geometric dot-product formula above and leaves only the angle. The dot product of two **unit vectors** is therefore their **cosine similarity**, a common way of comparing feature or embedding vectors when direction matters more than magnitude, and an important ingredient of most modern machine-learning methods. Normalisation also matters in practice. Two vectors that point in the same direction but are written down at different scales (say, one is exactly ten times the other) give very different raw dot products, even though they represent the same underlying match. Normalising both to unit vectors first makes their scores directly comparable. (This is the surprise in this session's in-class discussion problem, if you have done it.)

The length of a matrix is defined in the same way. If we treat every entry as part of one long vector, the length formula gives the matrix's **Frobenius norm**,

$$\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2}$$

a single number that measures how “big” a matrix is overall. It is useful throughout the rest of the course, for instance to measure how much two matrices differ (the norm of their difference), or how large a set of learned weights has grown during training.

### 1.2.6 Eigenvectors and PCA, intuitively

A matrix  $A$  transforms any vector it multiplies, and usually changes both its length and its direction. For most matrices, however, a handful of special directions keep their orientation and are only stretched or shrunk. Such a vector  $\mathbf{v}$  is an **eigenvector** of  $A$ , and the factor it is scaled by is the corresponding **eigenvalue**  $\lambda$ :

$$A\mathbf{v} = \lambda\mathbf{v}$$

**Example.** For  $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ , the vector  $(1, 1)$  is an eigenvector with eigenvalue 3. One matrix-vector product confirms this:  $A(1, 1) = (2+1, 1+2) = (3, 3) = 3 \times (1, 1)$ . The other eigenvector is  $(1, -1)$ , with eigenvalue 1. Geometrically,  $A$  stretches anything along  $(1, 1)$  by a factor of 3 and leaves  $(1, -1)$  unchanged. Every other vector is a mixture of the two, so it is turned towards the  $(1, 1)$  axis.

You will not need to compute eigenvalues by hand in this course, because a computer does it (`np.linalg.eigh`, as in the notebook). What matters is the idea and where it is used. **Principal component analysis (PCA)** finds the directions of greatest variance in a dataset, and these directions are the eigenvectors of the covariance matrix  $X^T X / (n - 1)$  from the section above. The eigenvalues say how much of the variance each direction explains. PCA is the standard first look at a gene-expression matrix. If every sample is plotted along the first two eigenvectors, samples with similar expression cluster together.

**i** Optional: the cross product (used later, for AlphaFold)

For vectors in three dimensions, the **cross product**  $\mathbf{a} \times \mathbf{b}$  produces a *new vector*, not a number, that is perpendicular to both  $\mathbf{a}$  and  $\mathbf{b}$ :

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

For example,  $(2, -1, 3) \times (1, 4, 0) = ((-1)(0) - (3)(4), (3)(1) - (2)(0), (2)(4) - (-1)(1)) = (-12, 3, 9)$ .

Its length equals  $\|\mathbf{a}\| \|\mathbf{b}\| \sin \theta$ , with the *sine* where the dot product has the *cosine*. It is therefore largest when the two vectors are perpendicular and zero when they are parallel. The cross product exists only in three dimensions. In structural biology it gives the direction perpendicular to a plane, such as the plane of three backbone atoms, and it is often used when transforming and rotating protein structures (Sessions 6 and 12).

**Practice.** Compute  $(1, 2, 0) \times (0, 1, 3)$ . *Answer:*  $((2)(3) - (0)(1), (0)(0) - (1)(3), (1)(1) - (2)(0)) = (6, -3, 1)$ .

**Practice problems.**

1. Compute the length of  $\mathbf{v} = (1, 2, 2)$  and the Frobenius norm of  $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ .
2. Compute  $(2, -1, 3) \cdot (1, 4, 0)$ .
3. Normalise  $\mathbf{v} = (3, 4)$ .
4. An expression matrix  $X$  holds  $500 \text{ samples} \times 20 \text{ genes}$ , and  $\mathbf{w}$  has one weight per gene. What are the shapes of  $X$ ,  $\mathbf{w}$ ,  $X\mathbf{w}$ ,  $X^\top$  and  $X^\top X$ ? Can you compute  $X\mathbf{u}$  for a vector  $\mathbf{u}$  with 500 entries? And  $X^\top \mathbf{u}$ ?
5. Compute  $X\mathbf{w}$  for  $X = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{pmatrix}$  and  $\mathbf{w} = (1, 2, 3)$ , and write down  $X^\top$ .
6. Check that  $(1, -1)$  is an eigenvector of  $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ , and read off its eigenvalue.

### **i** Solutions

1.  $\|\mathbf{v}\| = \sqrt{1^2 + 2^2 + 2^2} = \sqrt{9} = 3$ ;  $\|A\|_F = \sqrt{1^2 + 0^2 + 0^2 + 1^2} = \sqrt{2} \approx 1.414$ .
2.  $(2)(1) + (-1)(4) + (3)(0) = 2 - 4 + 0 = -2$ .
3.  $\|\mathbf{v}\| = \sqrt{3^2 + 4^2} = 5$ , so  $\hat{\mathbf{v}} = (0.6, 0.8)$ .
4.  $X$ :  $(500, 20)$ ;  $\mathbf{w}$ :  $(20, )$ ;  $X\mathbf{w}$ :  $(500, )$ , one score per sample;  $X^\top$ :  $(20, 500)$ ;  $X^\top X$ :  $(20, 20)$ .  $X\mathbf{u}$  is impossible, since the inner dimensions 20 and 500 don't match.  $X^\top \mathbf{u}$  works:  $(20, 500)(500, ) \rightarrow (20, )$ , one number per gene.
5.  $X\mathbf{w} = ((1)(1) + (0)(2) + (2)(3), (0)(1) + (1)(2) + (1)(3)) = (7, 5)$ , and  $X^\top = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{pmatrix}$ .
6.  $A(1, -1) = (2-1, 1-2) = (1, -1) = 1 \times (1, -1)$ : same direction, eigenvalue 1.

### Check your understanding:

The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)

### 1.2.7 Further reading

- Wikipedia, [Norm \(mathematics\)](#) and [Matrix norm](#)
- Wikipedia, [Dot product](#) — the algebra in more generality than needed here (complex vectors, function spaces)
- Wikipedia, [Matrix multiplication](#) and [Transpose](#)
- Wikipedia, [Unit vector](#) and [Cross product](#) (optional)
- Wikipedia, [Eigenvalues and eigenvectors](#) and [Principal component analysis](#)
- Wikipedia, [Tensor](#) — the general mathematical object; Session 2 covers the practical, code-first version
- d2l.ai, [Linear Algebra](#) — matrices, shapes, transposes and norms at the same level as above, with code

### 1.2.8 Further watching

[Dot products and duality](#) (3Blue1Brown, Essence of Linear Algebra, ch. 9, 14 min) animates the geometric picture of the dot product above:

Video: [Dot products and duality, 3Blue1Brown](#)

**Optional videos**, if you want to refresh more of this part:

- [Cross products](#) (3Blue1Brown, Essence of Linear Algebra, ch. 10, 9 min), for the optional cross-product box;
- [Eigenvectors and eigenvalues](#) (3Blue1Brown, Essence of Linear Algebra, ch. 14, 17 min).

## 1.3 Calculus for machine learning

Derivatives

**What's the integral of  $(1/\text{cabin})d(\text{cabin})$ ?**

A natural log cabin!

### 1.3.1 Derivatives

A derivative tells you how much a function's output changes for a small change in its input; it is the slope of the tangent line at a point. If you have seen derivatives before, that is the whole idea. What is worth reviewing is what happens once there is more than one input, but first we need a few standard derivatives.

The table below lists the derivatives that come up constantly in this course. They are worth memorising rather than re-deriving every time:

Table 1.1: Derivatives used throughout the course.

| $f(x)$                       | $f'(x)$                    |
|------------------------------|----------------------------|
| constant $c$                 | 0                          |
| $x^n$                        | $nx^{n-1}$                 |
| $e^x$                        | $e^x$                      |
| $\ln(x)$                     | $1/x$                      |
| $\sin(x)$                    | $\cos(x)$                  |
| $\cos(x)$                    | $-\sin(x)$                 |
| $\sigma(z) = 1/(1 + e^{-z})$ | $\sigma(z)(1 - \sigma(z))$ |

The last row deserves special attention. It is the derivative that Session 9's back-propagation needs for every sigmoid-activated unit in a network. It is usually written

in terms of  $\sigma(z)$  itself, rather than expanded in  $e^{-z}$ , because  $\sigma(z)$  has already been computed on the forward pass, so the derivative costs nothing extra.

A derivative is zero where a function is momentarily flat, that is, at a peak, a trough, or an inflection point with a horizontal tangent.  $\sin(x)$  and its derivative  $\cos(x)$  make this directly visible:

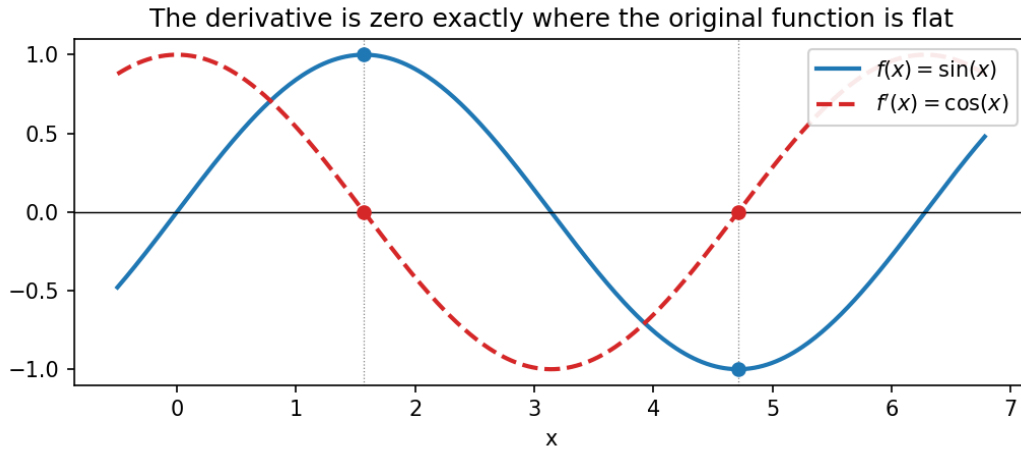


Figure 1.3:  $f(x) = \sin(x)$  (solid) and  $f'(x) = \cos(x)$  (dashed). Wherever  $\sin(x)$  reaches a peak or trough (dotted vertical lines),  $\cos(x)$  crosses exactly zero.

### 1.3.2 Partial derivatives and the gradient

Most functions in machine learning have many inputs, so we need derivatives with respect to each of them. A **partial derivative** of a function of several variables, written  $\partial y / \partial x_i$ , is computed by treating every variable except  $x_i$  as a constant and differentiating as usual. Stacking all the partial derivatives of a function into one vector gives its **gradient**:

$$\nabla y = \left( \frac{\partial y}{\partial x_1}, \dots, \frac{\partial y}{\partial x_n} \right)$$

**Worked example.** For  $f(x_1, x_2) = 3x_1^2 + 2x_1x_2$ :

$$\frac{\partial f}{\partial x_1} = 6x_1 + 2x_2, \quad \frac{\partial f}{\partial x_2} = 2x_1$$

(For  $\partial f / \partial x_1$ ,  $x_2$  is held constant, so  $2x_1x_2$  differentiates like  $2x_2 \cdot x_1$ , giving  $2x_2$ . For  $\partial f / \partial x_2$ ,  $x_1$  is held constant, so  $3x_1^2$  contributes nothing and  $2x_1x_2$  differentiates to  $2x_1$ .) At the point  $(1, 2)$ :

$$\nabla f(1, 2) = (6(1) + 2(2), 2(1)) = (10, 2)$$

Geometrically, the gradient at a point tells you two things at once: the *direction* in which the function increases fastest, and, through its length, *how steeply* it increases. If you point yourself in the opposite direction, you are therefore heading downhill as fast as possible from where you stand.

### 1.3.3 Gradient descent

Stepping against the gradient is the whole idea behind **gradient descent**, which Session 7 uses to train the perceptron introduced above. Suppose we have a function that measures how wrong a model's predictions are (a *loss function*). If we repeatedly take a sufficiently small step in the direction opposite to its gradient, the loss decreases, at least locally. Concretely, the update rule is

$$x_{n+1} = x_n - \eta f'(x_n)$$

where  $\eta$  (eta) is a small positive **learning rate** that controls the step size. We can watch this converge, one step at a time, for the one-variable function  $f(x) = (x - 3)^2$ , whose derivative is  $f'(x) = 2(x - 3)$ , starting from  $x_0 = 0$  with  $\eta = 0.3$ :

Table 1.2: Gradient descent on  $f(x) = (x - 3)^2$ , starting from  $x_0 = 0$  with  $\eta = 0.3$ .

| $n$ | $x_n$ | $f(x_n)$ | $f'(x_n)$ | $x_{n+1} = x_n - 0.3f'(x_n)$ |
|-----|-------|----------|-----------|------------------------------|
| 0   | 0.000 | 9.00     | -6.00     | 1.800                        |
| 1   | 1.800 | 1.44     | -2.40     | 2.520                        |
| 2   | 2.520 | 0.23     | -0.96     | 2.808                        |
| 3   | 2.808 | 0.037    | -0.384    | 2.923                        |

Two things stand out in the table. First,  $x_n$  closes in on 3, the true minimum, where  $f'(x) = 0$ . Second, the *steps themselves* shrink as  $x_n$  gets closer, simply because the derivative shrinks, so nobody has to decrease  $\eta$  by hand. From Session 7 on, you will see the same iterations inside a real model's training loop. The only difference is that many weights are updated at once, using the gradient vector instead of a single derivative.

### 1.3.4 The chain rule

The idea from calculus that matters most for this course is the **chain rule**. For a composite function  $y = f(g(x))$ , the derivative is

$$\frac{dy}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$$

In words, the derivative of a function of a function is the product of their individual derivatives. This looks unremarkable, but it is the one idea that backpropagation (Session 9) is built from. A neural network’s output is a function of a function of a function of its weights. The chain rule lets us compute how each weight, however deeply buried, should change, by multiplying together the derivatives of every function it passes through on the way to the output.

**i** Optional: integrals, briefly (for AUROC on Session 8)

Integration is differentiation run backwards: given a derivative, it finds the function the derivative came from. Geometrically, the **definite integral**  $\int_a^b f(x) dx$  is the area under the curve of  $f$  between  $a$  and  $b$ . The **indefinite integral**  $\int f(x) dx$  recovers a whole family of functions that differ only by a constant  $C$ . Since the derivative of a constant is always zero (see the table above), integration alone can never tell which constant the original function had.

**Worked example.** For the gradient-descent table above, we differentiated  $f(x) = (x - 3)^2$  into  $f'(x) = 2(x - 3)$ . Integrating goes the other way and should recover  $f$  itself, up to a constant:

$$\int 2(x - 3) dx = (x - 3)^2 + C$$

This is the  $f(x)$  the worked example started from. It illustrates, informally, the **Fundamental Theorem of Calculus**: integration and differentiation are inverse operations.

Integrals show up in two places later in the course, and neither requires solving an integral by hand. First, a continuous probability density is not itself a probability. Only the *area under* a stretch of it, its integral over that range, gives an actual probability. Second, from Session 8 on, the “area under the ROC curve” (AUROC) and the “area under the precision-recall curve” (AUPR) are two of the course’s standard metrics for evaluating classifiers. Both are integrals, but they are computed numerically (by the trapezoidal rule, which sums thin strips under a finite set of points) rather than solved in closed form, because a real ROC curve is not a tidy algebraic function.

**Practice problem.** For  $f(x) = 4x^3$ , find  $f'(x)$  using the table above, then integrate your answer and confirm that you recover  $f(x)$  itself, up to a constant. *Solution:*  $f'(x) = 12x^2$ , by the power rule  $\frac{d}{dx}x^n = nx^{n-1}$  with  $n = 3$ , times the constant factor 4:  $\frac{d}{dx}(4x^3) = 4 \cdot 3x^2 = 12x^2$ . Integrating back:  $\int 12x^2 dx = 4x^3 + C$ , which is  $f(x)$  up to the constant integration can never recover.

**Practice problem.** For  $f(x_0, x_1) = 3x_0^2 + 2x_0x_1$ , compute  $\nabla f$  at the point  $(1, 2)$ .

**i** Solution

$\partial f / \partial x_0 = 6x_0 + 2x_1 = 6(1) + 2(2) = 10$ ;  $\partial f / \partial x_1 = 2x_0 = 2(1) = 2$ . So  $\nabla f(1, 2) = (10, 2)$ .

## Check your understanding:

The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)

### 1.3.5 Further reading

- Wikipedia, [Gradient](#) and [Chain rule](#) — further into multivariate calculus (Jacobians, the generalised chain rule) than this course needs
- Wikipedia, [Integral](#) — the formal definition and techniques of integration, beyond the informal treatment above
- d2l.ai, [Calculus](#) — the same gradient-descent idea, with code

### 1.3.6 Further watching

*Visualizing the chain rule and product rule* (3Blue1Brown, Essence of Calculus, ch. 4, 16 min):

Video: [Visualizing the chain rule and product rule, 3Blue1Brown](#)

**Optional videos**, if you want to refresh more of this part (3Blue1Brown, Essence of Calculus):

- *The paradox of the derivative* (ch. 2, 17 min), on what “instantaneous rate of change” actually means;
- *Derivative formulas through geometry* (ch. 3, 18 min), on where the formulas in the table of derivatives come from;
- *Integration and the fundamental theorem of calculus* (ch. 8, 21 min);
- *What does area have to do with slope?* (ch. 9, 13 min), a second view of why integration and differentiation are inverses.

## 1.4 Probability and statistics

### Probability

I surveyed many people who had played Russian roulette. Seems like the probability of dying is actually 0%.

Probability in general is covered by the biostatistics course. This part gives the vocabulary that the rest of this course uses. It starts with a short recap and **correlation**, continues with **conditional probability**, **log probabilities and log-odds**, **likelihood**, and **p-values with multiple testing**, and ends with an RNA-seq example.

### 1.4.1 A short recap: expectation and variance

The **expectation** of a random variable is its probability-weighted average,  $\mathbb{E}[X] = \sum_i x_i P(X = x_i)$ . For a fair die it is 3.5.

The expectation says nothing about how spread out the values are. That is measured by the **variance**,  $\text{Var}(X) = \mathbb{E}[(X - \mu)^2]$ , the average squared distance from the mean. The **standard deviation**  $\sigma = \sqrt{\text{Var}(X)}$  puts the spread back in the units of  $X$  (for the die,  $\sigma \approx 1.71$ ). For a normal distribution, about 95% of the values lie within  $2\sigma$  of the mean.

### 1.4.2 Correlation: Pearson and Spearman

Often we want to know whether two variables go together, for example the mRNA and protein levels of a gene. A **correlation coefficient** measures how strongly they do, on a scale from  $-1$  (one goes up as the other goes down) through  $0$  to  $+1$  (they go up together). The two common coefficients differ in what “going together” means:

- **Pearson’s**  $r$  measures how well the points fit a *straight line*: the covariance of  $x$  and  $y$  divided by the product of their standard deviations,

$$r = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2}}$$

- **Spearman’s**  $\rho$  is Pearson’s  $r$  computed on the *ranks* of the values instead of the values themselves. It measures whether  $y$  tends to increase when  $x$  does (a monotonic relationship), whether or not along a straight line, and a single extreme value cannot dominate it.

**Worked example.** Five genes have mRNA levels  $(1, 2, 3, 4, 100)$  and protein levels  $(2, 4, 6, 8, 10)$ . Every increase in mRNA goes with an increase in protein, so Spearman’s  $\rho = 1$ . Pearson’s  $r$ , however, is only  $0.72$ , because the one extreme value does not lie on the line through the others. With  $5$  instead of  $100$ , it would be exactly  $1$ .

Three cautions apply to any correlation, and all of them are used in Session 2:

- **Plot first.** Both coefficients are  $0$  for a U-shaped relationship, and a single number can hide what a scatter plot shows at once.
- **A correlation has a p-value too**, and it depends on the number of points. A strong-looking correlation from five points can easily arise by chance.
- **Correlation is not causation.** Two variables can move together because both depend on a third.

### 1.4.3 Conditional probability

Many questions in biology are about probabilities under a condition, for example the probability that a protein has a signal peptide *given* that it is a membrane protein. The probability of  $A$  *given* that  $B$  has happened, written  $P(A | B)$ , restricts attention to the cases where  $B$  is true:

$$P(A | B) = \frac{P(A \text{ and } B)}{P(B)}$$

**Worked example.** A fair die is rolled. Given that the result is greater than 3, what is the probability that it is even? Only 4, 5 and 6 remain, and two of them are even, so  $P(\text{even} | X > 3) = 2/3$ , not the unconditional  $1/2$ .

**The direction matters.**  $P(A | B)$  and  $P(B | A)$  are different numbers. Suppose that among 200 proteins, 60 are membrane proteins and 30 of those have a signal peptide, while 20 of the other 140 have one too. Then  $P(\text{signal peptide} | \text{membrane}) = 30/60 = 0.5$ , but  $P(\text{membrane} | \text{signal peptide}) = 30/50 = 0.6$ . Mixing the two up is one of the most common errors in reading statistics, and it is the error behind misreading p-values (below).

Conditional probabilities are everywhere in this course. A profile or hidden Markov model (Session 5) gives the probability of a residue *given* the position in the family, and a classifier (Session 8) gives the probability of a class *given* the input.

#### **i** Optional: Bayes' rule

Bayes' rule turns one conditional probability into the other. It follows from applying the definition of conditional probability in both directions:

$$P(A | B) = \frac{P(B | A) P(A)}{P(B)}$$

**Example.** A condition affects 1% of a population. A diagnostic test detects it in 95% of the people who have it,  $P(\text{positive} | \text{disease}) = 0.95$ , and gives a false positive in 5% of the people who do not. If someone tests positive, what is the probability that they have the condition? Positive results come from both groups, so  $P(\text{positive}) = (0.95)(0.01) + (0.05)(0.99) = 0.059$ , and

$$P(\text{disease} | \text{positive}) = \frac{(0.95)(0.01)}{0.059} \approx 0.16$$

Despite the 95% sensitivity of the test, a positive result therefore means only about a 16% chance of having the condition. The condition is rare, so the false positives from the large healthy group outnumber the true positives. Bayes' rule combines a **prior** (1% have the condition) with **evidence** (the test). Maximum likelihood, below, in contrast uses only the likelihood and no prior.

#### 1.4.4 Log probabilities and log-odds

The probability of a whole sequence is a product of per-position probabilities, and products of many small numbers quickly become impossibly small. Under a uniform model, in which A, C, G and T are equally likely, any particular 1,000-base sequence has the probability  $0.25^{1000} \approx 10^{-602}$ . This is smaller than the smallest number a computer normally stores, so the computer may round it to zero. The **logarithm** solves the problem, because it turns multiplication into addition:

$$\log(xy) = \log(x) + \log(y)$$

The log-probability of the sequence is then a sum of 1,000 per-base terms,  $1000 \times \log_{10}(0.25) \approx -602$ , which is an ordinary number. (Logarithms to different bases,  $\ln$ ,  $\log_2$  and  $\log_{10}$ , differ only by a constant factor.)

**Log-odds scores.** In bioinformatics we usually compare two models. Is this sequence better explained by a family model or by the background frequencies? The **log-odds score** answers this question. It is the log of the ratio of the sequence's probabilities under the two models, and for independent positions it is a *sum* of per-position terms:

$$S = \log \frac{P(\text{sequence} \mid \text{model})}{P(\text{sequence} \mid \text{background})} = \sum_i \log \frac{P(x_i \mid \text{model})}{P(x_i \mid \text{background})}$$

A positive score means that the model explains the sequence better, and a negative one that the background does. Substitution matrices (Session 4, where BLOSUM entries are scaled log-odds) and profile scores (Session 5, where a PSSM column is a list of log-odds) work in this way, which is why their scores are added along an alignment.

**Worked example.** A GC-rich model gives G and C the probability 0.35 each, and A and T 0.15 each, while the background gives every base 0.25. Per base, in bits ( $\log_2$ ), G or C scores  $\log_2(0.35/0.25) = +0.49$ , and A or T scores  $\log_2(0.15/0.25) = -0.74$ . The sequence GCGCAT therefore scores

$$S = 4 \times 0.49 + 2 \times (-0.74) \approx +0.47 \text{ bits}$$

so it is  $2^{0.47} \approx 1.4$  times more likely under the GC-rich model than under the background.

**Logs of p-values.** Very small p-values are also shown on a log scale. The transformation  $-\log_{10}(p)$  turns  $p = 0.001$  into 3 and  $p = 10^{-6}$  into 6, and it is the y-axis of the volcano plot below.

**Back from log-odds to probability.** The **sigmoid function**,  $\sigma(z) = 1/(1 + e^{-z})$ , undoes a (natural) log-odds: if  $z = \ln(p/(1 - p))$ , then  $\sigma(z) = p$ . Logistic regression (Session 8) models the log-odds of a class as a weighted sum of the inputs and uses the sigmoid to turn it into a probability. Applied to an arbitrary score, the sigmoid still gives a number between 0 and 1, but that number is a calibrated probability only if the model was trained for it. (The derivative of the sigmoid is in the table in Section 1.3.)

### 1.4.5 Likelihood

So far we have computed the probability of data under a given model. A probability model tells you how likely the data are, given the model's parameters:  $P(\text{data} \mid \text{parameters})$ . Read the other way round, as a function of the parameters for data you have already observed, the same number is called the **likelihood**:

$$L(\text{parameters}) = P(\text{data} \mid \text{parameters})$$

#### Worked example.

Suppose we observe a 10-base fragment in which seven positions are G/C and three are A/T. We model each position as G/C with probability  $p$ , independently of the others. Because the positions are independent, the probability of the whole fragment is the product of the per-base probabilities:

$$L(p) = p^7(1 - p)^3$$

For  $p = 0.5$ ,  $L = 0.5^{10} \approx 0.00098$ . For  $p = 0.7$ ,  $L = 0.7^7 \times 0.3^3 \approx 0.0022$ , about 2.3 times higher, so the data are better explained by a GC-rich model. No value of  $p$  does better than 0.7, the observed fraction. Choosing the parameters that make the observed data most probable is called **maximum likelihood**, and it is how most models in this course are trained. Because a likelihood is a product, it is computed as a sum of logs, the **log-likelihood** (previous section). Many important loss functions later in the course, including **cross-entropy** (next section), are **negative log-likelihoods**, so minimising the loss maximises the likelihood.

### 1.4.6 Entropy: average surprise

The log of a probability also measures how *surprising* an outcome is. An outcome with probability  $p$  carries a **surprise** of  $-\log_2 p$  bits. A certain outcome ( $p = 1$ ) carries 0 bits, a fair coin flip 1 bit, and one of four equally likely bases 2 bits. The **entropy** of a distribution is its average surprise:

$$H(P) = - \sum_x P(x) \log_2 P(x)$$

Entropy therefore measures how unpredictable, or how *variable*, the outcomes are:

Table 1.3: The entropy of some distributions of bases and amino acids.

| Distribution                                         | Entropy |
|------------------------------------------------------|---------|
| one base always (a fully conserved alignment column) | 0 bits  |
| A or G, half each                                    | 1 bit   |

| Distribution                                     | Entropy                                                 |
|--------------------------------------------------|---------------------------------------------------------|
| the GC-rich model above (0.35, 0.35, 0.15, 0.15) | 1.88 bits                                               |
| all four bases equally likely                    | 2 bits, the maximum for DNA                             |
| all 20 amino acids equally likely                | $\log_2 20 \approx 4.32$ bits, the maximum for proteins |

The difference from the maximum, for example  $2 - H$  for a DNA column, is the column's **information content**. It says how much knowing the family tells you about that position, and it is what the height of a stack in a sequence logo shows (Session 4).

**Cross-entropy.** Entropy assumes that we know the true distribution. Suppose instead that the data really come from a distribution  $P$ , but we describe them with a model  $Q$ . The average surprise we then experience is the **cross-entropy**:

$$H(P, Q) = - \sum_x P(x) \log_2 Q(x)$$

The cross-entropy is never smaller than  $H(P)$ , and it equals  $H(P)$  only when  $Q = P$ . The extra cost,  $H(P, Q) - H(P)$ , is the **relative entropy** (Kullback-Leibler divergence) of Session 4.

**The link to likelihood.** Take the 10-base fragment above, with observed frequencies 0.7 G/C and 0.3 A/T, and a model that gives G/C the probability  $p$ . The cross-entropy is  $-(0.7 \log_2 p + 0.3 \log_2 (1-p))$ , which is 1.00 bit for  $p = 0.5$  and 0.88 bits for  $p = 0.7$ , the lowest possible value. This is exactly the log-likelihood per base with the sign changed, since  $\log_2 L(0.5) = -10$  bits for ten bases and  $\log_2 L(0.7) = \log_2 0.0022 \approx -8.8$ . **Minimising cross-entropy is therefore the same as maximum likelihood**, and this is why cross-entropy is the loss used to train classifiers from Session 8 on.

**Maximum entropy.** Entropy can also be used to choose a model. Among all distributions that agree with what you have measured, you pick the one with the highest entropy, because it assumes nothing more than the measurements. Session 12 meets this idea in the contact-prediction method DCA.

The same formula, with a different constant and natural logarithms, is the entropy of statistical physics, which is where Shannon borrowed the name from. The hydrophobic effect that drives protein folding (Session 6) is largely an effect of this physical entropy, namely the entropy of the surrounding water.

### 1.4.7 p-values and hypothesis testing

Statistical testing starts from a **null hypothesis** ( $H_0$ ), the assumption that there is no real effect and that any observed difference is due to chance. A **p-value** is the probability of seeing a result at least as extreme as the one actually observed, *if the null hypothesis were true*. By convention,  $p < 0.05$  is often treated as “statistically

significant”, but this threshold is a convention, not a law of nature. What the threshold guarantees is the following. If the null hypothesis is true and the assumptions of the test hold, a test with a significance threshold of 0.05 rejects the null hypothesis with a probability of 5%.

It follows that a p-value is a conditional probability,  $P(\text{data this extreme} \mid H_0)$ , and *not*  $P(H_0 \mid \text{data})$ . Confusing the two is the direction error from the conditional-probability section. A small p-value tells you that the data would be surprising if there were truly no effect. It does not, by itself, tell you the probability that there is no effect.

**The t-test, briefly.** The biostatistics course covers the t-test in depth, so here we only recall the idea. The t-test is a classical test for comparing the means of two groups. Welch’s version,  $t = (\bar{x}_1 - \bar{x}_2) / \sqrt{s_1^2/n_1 + s_2^2/n_2}$ , is usually preferred when equal variances cannot be assumed. A larger difference between the means, or less noisy data, gives a larger  $|t|$  and therefore a smaller p-value. For example, four control measurements (5.1, 4.9, 5.3, 5.0) against four treatment measurements (7.2, 6.8, 7.5, 7.0) give  $t \approx -11.9$  and  $p \approx 0.0001$ .

#### 1.4.8 The multiple testing problem

A p-value threshold of 0.05 means that, if the null hypothesis is true and the test’s assumptions hold, there is a 5% chance of a false positive *for that one test*. That is a reasonable risk to accept once. The situation is very different when the same test runs on every gene in a genome. If we simulate 2000 genes, none of which is actually differentially expressed, and test each one at  $p < 0.05$ , we get roughly  $2000 \times 0.05 = 100$  “significant” genes purely by chance. The result is a wall of false positives, not a list of real findings.

The solution is a **multiple testing correction**. Bonferroni’s correction is the simplest: it divides the significance threshold by the number of tests. For genomics, where thousands of tests are routine, it is often needlessly conservative. The standard alternative is to control the **false discovery rate (FDR)**. Instead of the chance of *any* false positive, the FDR is the *expected proportion* of false positives among everything called significant. The Benjamini-Hochberg procedure is the common way to control it. It does so under specified assumptions (independent or positively correlated tests), and it stays informative even with tens of thousands of tests.

#### 1.4.9 Worked example: RNA-seq differential expression and the volcano plot

The following example puts these ideas together on a realistic problem. We compare gene expression between a treatment and a control condition across 2000 genes, of which 50 are simulated to be genuinely differentially expressed, while the rest are pure noise. For each gene, a t-test gives a p-value. The **fold change** measures the size of the effect, independently of its statistical significance. It says how many times higher or lower the expression is in the treatment than in the control, and it is usually reported as the  $\log_2$  of that ratio.

A **volcano plot** shows both quantities at once, with the  $\log_2$  fold change on the x-axis and  $-\log_{10}(p\text{-value})$  on the y-axis. Smaller p-values, that is, stronger evidence, therefore plot higher up. The interesting genes are those that are both far from the centre (a big effect) and high up (strong evidence), and they give the plot the shape of its name:

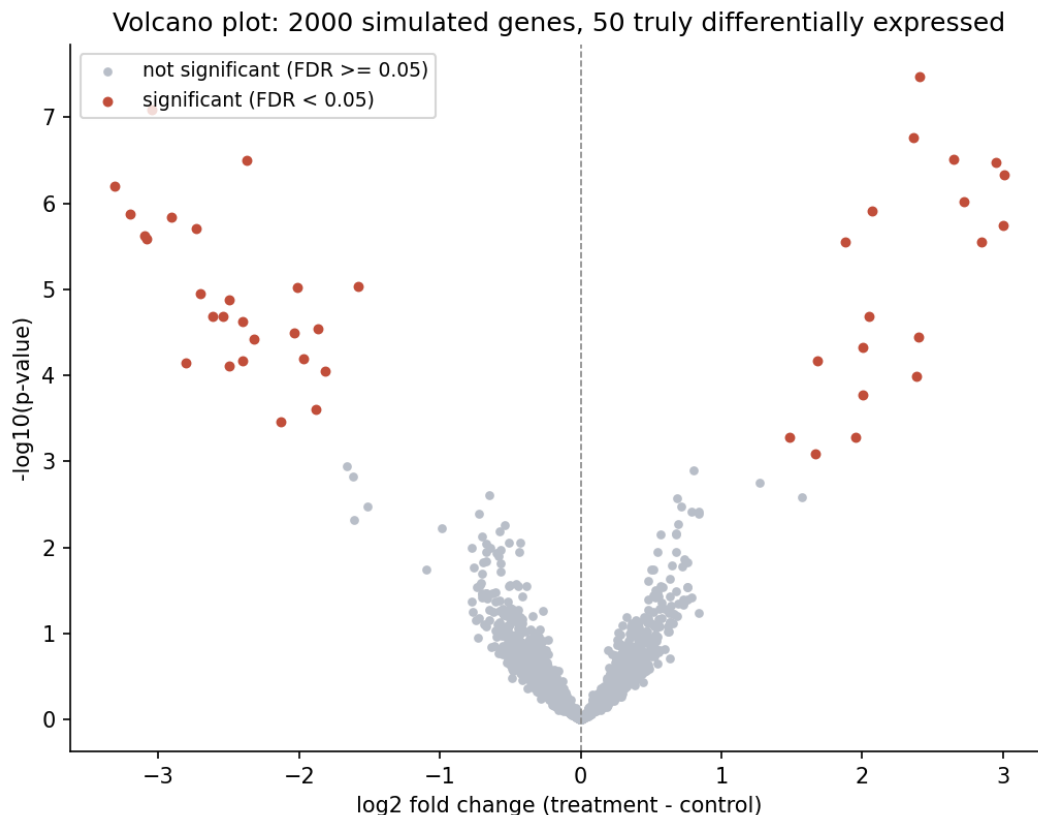


Figure 1.4: Volcano plot of the 2000 simulated genes:  $\log_2$  fold change against  $-\log_{10}(p\text{-value})$ .

With only the naive threshold  $p < 0.05$  (Welch's t-test for each gene), 135 genes in this simulated data are flagged as "significant". Of those, 85 are false positives among the 1950 genes with no real effect, close to the  $\sim 97.5$  expected by chance. With the Benjamini-Hochberg procedure at an FDR of 5% instead, 44 genes are flagged, and all 44 are among the 50 genes that are really differentially expressed. In this particular test there are no false positives, so the procedure recovers the large majority of the true effects while it stops the flood of false positives that the naive threshold let through. Note, however, that an FDR threshold of 5% does **not** guarantee zero false discoveries. That none of the 44 discoveries is false here is a property of this particular simulation.

### Practice problems.

1. Among 200 proteins, 60 are membrane proteins and 30 of those have a signal peptide; 20 of the other 140 have one too. What are  $P(\text{membrane} \mid \text{signal peptide})$

- and  $P(\text{signal peptide} \mid \text{not membrane})$ ?
2. A 4-base fragment has 3 G/C and 1 A/T. Compute its likelihood under  $p_{GC} = 0.5$  and under  $p_{GC} = 0.75$ . Which explains it better?
  3. With the GC-rich model of the worked example, what is the log-odds score of the sequence AT, in bits? Which model does it favour?
  4. If you test 100 genes with no real difference at  $p < 0.05$  and apply no correction, about how many false positives should you expect by chance alone?
  5. A gene has a p-value of  $10^{-6}$ . What is its height on a volcano plot's y-axis,  $-\log_{10}(p)$ ?
  6. An alignment column of a DNA motif has A in 70% of the sequences and C, G and T in 10% each. What is its entropy, and its information content, in bits?

### **i** Solutions

1. 50 proteins have a signal peptide, 30 of them membrane proteins:  $P(\text{membrane} \mid \text{signal peptide}) = 30/50 = 0.6$ .  $P(\text{signal peptide} \mid \text{not membrane}) = 20/140 \approx 0.14$ .
2.  $L(0.5) = 0.5^4 = 0.0625$ ;  $L(0.75) = 0.75^3 \times 0.25 \approx 0.105$ . The GC-rich value,  $p = 0.75$  (the observed fraction), explains it better.
3.  $2 \times \log_2(0.15/0.25) = 2 \times (-0.74) \approx -1.47$  bits: negative, so the background model explains AT better.
4.  $100 \times 0.05 = 5$  false positives expected, even though every one of the 100 genes is truly non-differentially-expressed.
5.  $-\log_{10}(10^{-6}) = 6$ .
6.  $H = -(0.7 \log_2 0.7 + 3 \times 0.1 \log_2 0.1) \approx 0.36 + 1.00 = 1.36$  bits; information content  $2 - 1.36 \approx 0.64$  bits. Less variable than a random column (2 bits), far from fully conserved (0 bits).

### **Check your understanding:**

The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)

### **1.4.10 Further reading**

- Wikipedia, [Pearson correlation coefficient](#) and [Spearman's rank correlation coefficient](#)
- Wikipedia, [Conditional probability](#) and [Likelihood function](#)
- Wikipedia, [Logarithm](#), [Log probability](#) and [Sigmoid function](#)
- Wikipedia, [P-value](#) and [Welch's t-test](#)
- Wikipedia, [Multiple comparisons problem](#), [False discovery rate](#), and [Volcano plot \(statistics\)](#)
- Wikipedia, [Bayes' theorem](#) (optional)
- d2l.ai, [Probability and Statistics](#) — connects this directly to how machine-learning models are evaluated

### 1.4.11 Further watching

[What Is A P-Value? - Clearly Explained](#) (Steven Bradburn, 8 min):

Video: [What Is A P-Value? - Clearly Explained, Steven Bradburn](#)

**Optional videos**, if you want to refresh more of this part:

- [False Discovery Rates, FDR, clearly explained](#) (StatQuest with Josh Starmer, 18 min);
- from 3Blue1Brown, for the Bayes box and the recap: [Bayes theorem, the geometry of changing beliefs](#), [But what is the Central Limit Theorem?](#), [Binomial distributions](#) and [Why “probability of 0” does not mean “impossible”](#).

## 1.5 The course notebook

`notebooks/session01-math-in-code.ipynb` recomputes every worked example and practice problem above in code, so that you can confirm that the code and the mathematics agree before Session 2 introduces the code side properly. For linear algebra, it covers the vector length, the dot product, matrix shapes and the transpose (1,000 proteins  $\times$  20 features), normalisation, eigenvalues found by the computer (with a small PCA) and the optional cross product. For calculus, it covers the gradient (by hand and by automatic differentiation) and gradient descent. For probability and statistics, it covers Pearson and Spearman correlation, conditional probability (exactly and by simulation), the optional Bayes example, log-odds scores, the GC likelihood, the t-test, and the simulated RNA-seq and volcano-plot analysis. The version below is a static, read-only copy. Open it in [Google Colab](#) via the badge at the top of the notebook page to edit and run it yourself.

## 1.6 What’s next

Session 2 takes every idea above and writes it in code the idiomatic way: the dot product becomes a line of PyTorch, and the gradient becomes something the framework computes for you automatically.

## 2 Session 2: Working like a computational biologist in Python

### Note

**Sources for this page:** Swaroop, C. H., *A Byte of Python*, <https://python.swaroopch.com/> — CC BY-SA 4.0, commercial use permitted. Python fundamentals. The scientific-stack sections name NumPy, pandas, Matplotlib, seaborn, SciPy, scikit-learn, Biopython and PyTorch as tools, with usage demonstrated in original example code; no text is reused from their documentation. Which tools get a light refresher is informed by the prerequisite DA7066 course's compendium (Arvestad, Helanow, Mörtberg & Sahlin), an internal planning reference. The dataset explored on this page is real published differential-expression data: Becares, N. et al. (2019), "[Impaired LXR \$\alpha\$  Phosphorylation Attenuates Progression of Fatty Liver Disease](#)", *Cell Reports* 26(4):984-995, loaded from its public copy shared as example data by [VolcanoR](#) (Goedhart, J. & Luijsterburg, M.S., 2020, "[VolcanoR is a web app for creating, exploring, labeling and sharing volcano plots](#)", *Scientific Reports* 10:20560). The globin records are fetched live from NCBI; every number on this page was read back from the notebook's output.

### Important

**Set up first.** This course runs its notebooks on Google Colab. The lab is done in VS Code, with the notebook running on a Colab machine through Google's Colab extension; the course notebook can also be opened in Colab in the browser. If you have not done so yet, follow [Setup](#): install VS Code, git and the Colab extension, clone the course notebooks, and check that you can run a notebook on a Colab machine.

### Bioinformatics

**Why do bioinformaticians prefer using Unix?** Because they can't live without grep.

You already know Python from the prerequisite course, so this session is not about new syntax. Instead, it is about the four things a computational biologist does with Python every day, using the libraries you already know:

1. **Represent data** with the right object: arrays, DataFrames, biological sequence objects and, later, tensors.
2. **Explore data** with summaries, filtering and plots, before trusting any number.

3. **Use existing tools**, meaning libraries, documentation and online databases, rather than reinventing them.
4. **Solve problems when code does not work**, using error messages, documentation, AI assistants and tests of your own assumptions.

To keep these four things concrete, the page follows one real dataset (a published RNA-seq experiment on mouse liver) and one real gene family (the human globins) all the way from representation to a small finished analysis. The mini challenge in Section 2.12 then leads straight into the lab.

## 2.1 Learning goals

By the end of this session you should be able to:

- choose between a list, a NumPy array, a pandas DataFrame, a Biopython Seq/SeqRecord and a PyTorch tensor, and say what each is for;
- explore an unfamiliar table: its shape, columns, summaries, filters and plots, and notice what the summaries tell you about how the data were made;
- plot first and test second, and pick a test that fits what the plot shows;
- read biological records from a local file or from NCBI;
- write a small class of your own, with `__init__`, attributes and methods, and recognise the same shape in a PyTorch model;
- get unstuck: read an error message, inspect objects, use the documentation and an AI assistant, and verify the answer;
- keep versions of your work with git.

## 2.2 Notebooks: what they are and why we use them

Every notebook in this book runs on Google Colab: the labs from VS Code, the course notebooks also in the browser, as described in [Setup](#). Before we use them, it is worth knowing what a notebook is and where its pitfalls lie.

A **notebook** (a `.ipynb` file, the Jupyter format) mixes three things in one document. It holds **text** (explanations, like this page), **code cells**, and the **output** of each cell (numbers, tables, plots, error messages), shown directly underneath it. You run one cell at a time, in whatever order you choose.

This makes notebooks well suited to exploring data, for three reasons:

- **Run a small piece, look, adjust, re-run** just that piece, without starting the whole analysis from the top.
- **Results stay next to the code that made them**, so you, a teacher or a colleague can see what was done and what came out.
- **Text explains the reasoning**, which a plain script does not.

The price of this flexibility is **hidden state**. A notebook’s variables live in the memory of a running Python process (the *kernel*), not on the page. Running cells out of order, re-running one cell several times, or editing a cell without re-running it all leave the memory different from what the page shows. The notebook has a small demonstration (Section 2.10), in which a cell that adds 1 to counter gives a new number every time you re-run it, although its code never changes. This is one of the most common sources of confusion for new notebook users, and the remedy is simple:

**! Important**

**Before submitting or sharing a notebook: Restart Kernel → Run All.**

This clears the memory and runs every cell from top to bottom, in order. If the notebook still runs and gives the same results, what the page shows is what the code does.

## 2.3 Scientific Python objects

Different objects represent different kinds of data, and each gives you the operations that make sense for its kind. The examples below, which you can run in the notebook, hold the same three transcript lengths (in nucleotides) in increasingly specialised objects:

```
lengths = [628, 577, 576]          # a list: a general container
print(lengths * 2)                 # [628, 577, 576, 628, 577, 576]: repeats the list

import numpy as np
arr = np.array(lengths)            # a NumPy array: numbers, with maths
print(arr * 2, arr.mean(), arr.shape) # [1256 1154 1152] 593.67 (3,)

import pandas as pd
genes = pd.DataFrame({"gene": ["HBB", "HBA1", "HBA2"], "length": lengths})
print(genes[genes["length"] > 600]) # a table with named columns and mixed types
```

Table 2.1: Python objects for biological data, what they represent and what they are useful for.

| Object           | Represents                                                                          | Useful operations                                                     |
|------------------|-------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| list             | any collection of things                                                            | append, loop over, index                                              |
| NumPy ndarray    | numbers of one type, in any number of dimensions (Session 1’s vectors and matrices) | elementwise maths, means, matrix products, boolean masks              |
| pandas DataFrame | a table: named columns, possibly of different types, one row per observation        | select and filter by name, group, summarise, read and write CSV files |

| Object                    | Represents                                                                          | Useful operations                                                                         |
|---------------------------|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| Biopython Seq / SeqRecord | a biological sequence / a sequence with its identifier, description and annotations | reverse complement, translate, GC content, parse and write FASTA or GenBank (Section 2.7) |
| PyTorch Tensor            | an array that can live on a GPU and remember how it was computed                    | everything an array does, plus automatic gradients (Session 7 onwards)                    |

Choosing the right object therefore matters. On a list, `lengths * 2` silently does something you did not mean, whereas on an array it does the maths you expect and multiplies every number by 2.

**Looking ahead: tensors and autograd.** A PyTorch **tensor** is the array of deep learning, and its extra trick is **automatic differentiation**. PyTorch records every operation on a tensor, and `.backward()` replays them in reverse, applying Session 1's chain rule at each step. As a result, Session 1's gradient practice problem, done by hand there, takes only a few lines of PyTorch code:

```
import torch
x0 = torch.tensor(1., requires_grad=True)
x1 = torch.tensor(2., requires_grad=True)
f = 3 * x0**2 + 2 * x0 * x1
f.backward()
print(x0.grad, x1.grad)                                # tensor(10.) tensor(2.): Session 1's answer
```

From Session 7 onwards you therefore never write a derivative by hand again. You call `.backward()` instead.

## 2.4 Exploring a biological dataset

With the objects in place, we can turn to real data. The first dataset comes from a mouse-liver RNA-seq experiment (Becares et al., 2019). All the mice were fed a high-fat/high-cholesterol diet, and the livers of mice carrying a mutant form of the transcription factor LXR $\alpha$  (S196A, which cannot be phosphorylated at that site) are compared with those of wild-type mice, three per group. The data have already been analysed with DESeq2, so here we explore the resulting table. It contains one row per gene, with the gene's mean expression (`baseMean`), `log2` fold change, p-value and adjusted p-value.

```
url = "https://raw.githubusercontent.com/JoachimGoedhart/VolcaNoseR/master/Becares-diff
df = pd.read_csv(url)
df.shape          # (3001, 9): 3,001 genes, 9 columns
df.columns        # Gene, baseMean, log2_FoldChange, FC, lfcSE, stat, pvalue, padj, m
```

```
df.head()          # the first five rows
df.describe()      # count, mean, spread and range of every numeric column
```

**Summaries tell you how the data were made.** `describe()` shows that the largest p-value in the file is 0.12, and the largest adjusted p-value 0.50. Under the null hypothesis raw p-values are spread evenly between 0 and 1, so a genome-wide table would reach close to 1, and this file almost certainly does not contain all genes. It is most likely a pre-filtered list of the 3,001 genes with  $p < 0.12$ , and any conclusion about “all genes” drawn from it would be wrong. Always look at the summaries before you look at a result.

**Selecting and filtering** use conditions on columns:

```
significant = df[df["padj"] < 0.05]          # 656 genes
strong = df[(df["padj"] < 0.05) & (df["log2_FoldChange"].abs() > 1)] # 171 genes: 46 u
df.sort_values("padj").head()                # the most signif
```

The most significant gene is *Osbpl3* (adjusted  $p = 8 \times 10^{-18}$ , down 3.1-fold). Among the significant genes, the largest increase is in *Cyp2c69* (log2 fold change 2.39, 5.2-fold up).

**NumPy calculations** work directly on columns. Mean expression, for example, spans five orders of magnitude, from 10 to 954,964 (albumin, *Alb*), with a median of 329. It is therefore usually worked with as `np.log10(df["baseMean"])`.

## 2.5 Seeing the data

A summary is only a handful of numbers, whereas a plot shows all 3,001 genes at once. Three plots answer three different questions:

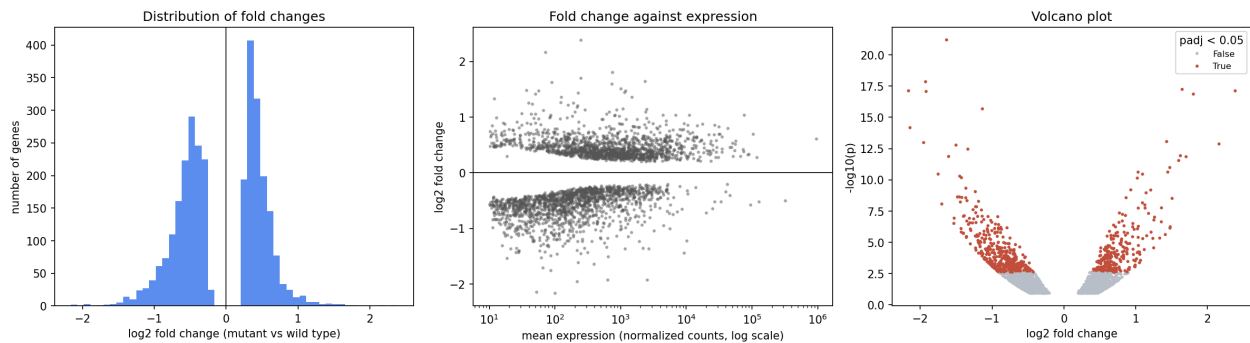


Figure 2.1: Three views of the same 3,001 genes, drawn in the notebook. Left: histogram of log2 fold changes. There is a gap around zero, because genes with very small changes rarely reach  $p < 0.12$ , so the filtering removed them. Slightly more genes go down than up. Middle: fold change against mean expression, with expression on a log scale. Lowly expressed genes show the largest changes in both directions. Right: volcano plot of effect size against significance; genes with adjusted  $p < 0.05$  are coloured (later courses discuss this in much more detail).

```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(1, 3, figsize=(15, 4))
axes[0].hist(df["log2_FoldChange"], bins=50)
axes[1].scatter(df["baseMean"], df["log2_FoldChange"], s=5)
axes[1].set_xscale("log")
df["significant"] = df["padj"] < 0.05
sns.scatterplot(data=df, x="log2_FoldChange", y="minus_log10_pvalue",
                hue="significant", s=8, ax=axes[2])
```

# distribution of one variable  
# relationship between two variables  
# skewed data: use a log scale  
# seaborn: plot straight lines

- A **histogram** shows the distribution of one variable.
- A **scatter plot** shows the relationship between two.
- **seaborn** draws from a DataFrame by column name, and colours points by a third column (hue=) in one argument.
- **Plots reveal how the data were made, too.** The empty band around zero in the left and middle panels is the pre-filtering from Section 2.4 seen again. Genes whose expression barely changed rarely reach  $p < 0.12$ , so they are not in the file. A complete experiment would have its tallest histogram bar at zero.
- Label your axes with units, and use a **log scale** for data that span orders of magnitude, such as expression levels. On a linear axis, the middle panel would show 3,001 points squeezed against the left edge, plus a few outliers.

## 2.6 Asking quantitative questions

**Plot first, test second.** Suppose we ask whether the size of a gene's fold change is related to how highly the gene is expressed. A test on the raw numbers says no:

```
from scipy import stats
stats.pearsonr(df["baseMean"], df["log2_FoldChange"].abs())    # r = -0.01, p = 0.43
```

The middle plot, however, says yes, because lowly expressed genes have larger fold changes. The two disagree because Pearson's correlation measures a *linear* relationship, and on the raw scale a handful of hugely expressed genes (albumin at 954,964) dominate it. The solution is to test what the plot shows:

```
stats.pearsonr(np.log10(df["baseMean"]), df["log2_FoldChange"].abs())    # r = -0.30, p = 
stats.spearmanr(df["baseMean"], df["log2_FoldChange"].abs())            # rho = -0.42, p =
```

Spearman's correlation uses only the ranks, so the scale does not matter to it. Had you run the first test without plotting, you would have reported "no relationship". (Why lowly expressed genes show larger fold changes is itself a lesson. With few reads per gene, the estimates are noisier. Session 8 returns to noise and evaluation.)

**Existing tools for standard questions.** SciPy and scikit-learn implement the standard methods, so you do not have to:

- `stats.ttest_ind(control, treatment, equal_var=False)` is Session 1's Welch t-test. For Session 1's control (5.1, 4.9, 5.3, 5.0) and treatment (7.2, 6.8, 7.5, 7.0) it gives  $t \approx -11.9$ ,  $p \approx 0.0001$ . SciPy's default, `equal_var=True`, is Student's test, which assumes equal variances.
- `scipy.cluster.hierarchy.linkage` builds a tree by repeatedly merging the closest clusters. This is the algorithm behind Session 4's guide tree.
- scikit-learn has ready-made models (LinearRegression, LogisticRegression, `train_test_split`, MLPClassifier, ...). Sessions 7-9 build these by hand so that you see the mechanics, but in practice you would call the library.

### Check your understanding:

The interactive quiz for this section is in the online version: [course.bioinfo.se](http://course.bioinfo.se)

## 2.7 Biological objects

Besides tables of numbers, a computational biologist works with sequences. Biopython's Seq is a sequence that knows biology, so you never have to write a complement table yourself. As an example, we use human beta-globin (*HBB*, accession NM\_000518.5). The first ten codons and the corresponding amino acids can be examined like this:

```

from Bio.Seq import Seq
from Bio.SeqUtils import gc_fraction

hbb = Seq("ATGGTGCATCTGACTCCTGAGGAGAAGTCT")
len(hbb)                # 30
gc_fraction(hbb)        # 0.5
hbb.reverse_complement() # AGACTTCTCCTCAGGAGTCAGATGCACCAT
hbb.translate()          # MVHLTPEEKS

```

A **SeqRecord** wraps a **Seq** together with what a database knows about it: `record.id`, `record.description`, and `record.features`, the annotated regions such as the coding sequence (CDS). Here `record.seq` and `record.features` are **attributes**, and `record.seq.translate()` is a **method**. A **SeqRecord** is thus an **object**, built from a **class** that Biopython defines, and it bundles data with the functions that act on it. The next section shows how to write such a class yourself.

## 2.8 Your own objects: classes

Biopython's **SeqRecord** is one example of a pattern that runs through the whole scientific Python stack, and you will write such classes yourself. Every PyTorch model from Session 8 onwards is a class, so it is worth learning the pattern properly now, on a biological example, before it turns up in a neural network.

A **class** is a blueprint, and each **object** made from it is one instance with its own data. The data are the object's **attributes**, and the functions that belong to the class and act on those data are its **methods**. A class for proteins could look like this:

```

from Bio.SeqUtils import molecular_weight

class Protein:
    """A protein: a name and an amino-acid sequence."""

    def __init__(self, name, sequence):
        self.name = name                # attributes: the data each object carries
        self.sequence = sequence

    def length(self):
        return len(self.sequence)      # a method: a function that uses the object's o

    def composition(self, residues):
        """The fraction of the sequence made of the given residues."""
        return sum(self.sequence.count(aa) for aa in residues) / self.length()

    def weight(self):
        return molecular_weight(self.sequence, seq_type="protein")

```

```

beta = Protein("HBB", "MVHLTPEEKSAVTALWGK...") # the full 147 residues of UniProt P68
beta.length() # 147
beta.composition("AVLIMFWC") # 0.442, the hydrophobic fraction
beta.weight() # 15998 (daltons)

```

Two names need explaining. `__init__` is run once, automatically, when an object is created: `Protein("HBB", ...)` makes a new, empty object and passes it to `__init__` together with the two arguments, and `__init__` stores them as attributes. **self** is that object. Every method receives the object it was called on as its first argument, so `beta.length()` is really `Protein.length(beta)`, and inside the method `self.sequence` means *this* protein's sequence. A hundred `Protein` objects share the same methods but each has its own name and sequence.

Methods whose names begin and end with two underscores are **special methods**, which Python calls for you. `__init__` is the first you meet. Defining `__len__` makes `len(beta)` work, `__repr__` decides what `print(beta)` shows, and `__call__` makes the object itself callable like a function. The notebook adds the first two to `Protein`.

**Inheritance** lets a new class start from an existing one. An `Enzyme` is a `Protein` that also has an EC number, so it inherits everything `Protein` has and adds one attribute:

```

class Enzyme(Protein):
    def __init__(self, name, sequence, ec_number):
        super().__init__(name, sequence) # let Protein set name and sequence
        self.ec_number = ec_number

lyz = Enzyme("LYZ", "KVFERCELARTLKRGL...", "3.2.1.17") # human lysozyme, mature chain
lyz.weight() # 14701: a method inherited from Protein

```

`super().__init__(...)` runs the parent class's `__init__`, so that the attributes it sets up exist in the new object too. Forgetting it is a common bug: the object is created without error, and fails later when a method looks for `self.sequence`.

**Looking ahead: a PyTorch model is a class.** This is exactly the shape of every neural network in Sessions 8-12. A model inherits from `nn.Module`, calls `super().__init__()`, creates its layers as attributes in `__init__`, and describes the computation from input to output in a method called `forward`:

```

import torch.nn as nn

class TinyNet(nn.Module):
    def __init__(self):
        super().__init__() # nn.Module sets itself up first
        self.layer = nn.Linear(3, 1) # a layer is an attribute

    def forward(self, x):
        return self.layer(x)

```

```
model = TinyNet()
model(torch.ones(1, 3))          # runs forward
```

Note that you call `model(x)`, not `model.forward(x)`. Calling an object runs its special method `__call__`, which `nn.Module` defines to do some bookkeeping and then call your `forward`. You never need to know more about that bookkeeping than that it is there, but knowing the pattern makes a PyTorch model readable: the attributes are what the model *has*, its layers and their weights, and `forward` is what it *does*. Because the layers are attributes, `nn.Module` can find all their weights for you, which is what training them in Session 9 relies on.

## 2.9 Data coming from somewhere

Real data arrive either from files on disk or from databases over the network. In both cases Python gives you a **file handle**, an object you read from. You open it with `with`, which closes it again automatically when the block ends, even if an error happens inside:

```
from Bio import SeqIO

with open("globins.fasta") as handle:          # a local FASTA file
    for record in SeqIO.parse(handle, "fasta"):
        print(record.id, len(record.seq))
```

**Compressed files** need no separate unpacking. Sequence data usually come as `.fasta.gz` or `.fastq.gz`, and `gzip.open(path, "rt")` gives a handle that `SeqIO.parse` reads directly:

```
import gzip
with gzip.open("reads.fasta.gz", "rt") as handle:
    records = list(SeqIO.parse(handle, "fasta"))
```

Archives in `.zip` and `.tar.gz` format hold several files (the capstone project's data, for example, come as a `.zip`). They are opened with the standard-library modules `zipfile` and `tarfile`, which list the contents with `.namelist()` or `.getnames()` and unpack them with `.extractall()`.

**Remote data** work in the same way. `pd.read_csv(url)` read the table of Section 2.4 straight from the web, and for NCBI, `Bio.Entrez` returns a handle:

```
from Bio import Entrez, SeqIO
Entrez.email = "your.name@student.su.se"      # NCBI requires a contact address

with Entrez.efetch(db="nucleotide", id="NM_000518.5", rettype="gb", retmode="text") as handle:
    record = SeqIO.read(handle, "genbank")
    print(record.description, len(record.seq))  # Homo sapiens hemoglobin subunit
```

NCBI allows about three requests per second without an API key. You should therefore fetch many records in one request, or pause between requests, rather than hammer the server in a fast loop.

### Check your understanding:

The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)

## 2.10 When you don't know how to do something

Everyone's code fails, all the time, so the useful question is how to notice and fix the failures. There are three kinds of mistake, and they are not equally easy to catch:

1. **A Python error.** Python stops and tells you what went wrong, for example a `KeyError` when a column name is misspelled.
2. **Correct code, wrong analysis.** The code runs, but it does not answer your question, for example because you plotted GC content on the x-axis when you meant length, or filtered on the p-value when you meant the adjusted p-value. Nothing fails.
3. **A biologically nonsensical result.** The code runs and the numbers look reasonable, but they cannot be right. Computing a globin's protein length as its mRNA length divided by three, for example, gives 654 amino acids for cytoglobin, while globins are about 150 amino acids long. (The mRNA includes untranslated regions, and only the coding sequence is translated.)

### Remember

**Code running without an error does not mean the analysis is correct.**

AI assistants are very good at removing mistakes of the first kind. They do much less to stop the second and third kinds, which you catch only by asking whether the result answers your question and whether it is biologically plausible. For the first kind, experienced programmers follow a routine for getting unstuck:

**read the error → inspect → documentation → AI assistant (or google) → verify.**

**Read the error message, from the bottom.** The last line says *what* went wrong, and the arrow above it says *where* in your code. Asking for a column that does not exist, for example, gives:

```
----> 1 df["log2FC"].mean()
KeyError: 'log2FC'
```

On a `DataFrame`, a `KeyError` means “no column with that name”.

**Inspect the objects involved.** Here `df.columns` shows that the column is called `log2_FoldChange`. The same few tools answer most questions:

- `type(x)`: what kind of object is this?
- `x.shape`, `len(x)`, `df.dtypes`: how big is it, and what is in it?
- `df.head()`: what does it actually look like?
- `help(x)` or `x?` in a notebook: what can it do?

**Read the documentation.** Every library on this page has a user guide and an API reference, and the documentation is the authority on what a function does in *your* version. Old tutorials, for example, still write from `Bio.SeqUtils` import `GC`, which fails in current Biopython (1.86 when this page was written):

```
ImportError: cannot import name 'GC' from 'Bio.SeqUtils'
```

The documentation names its replacement, `gc_fraction`.

**Ask an AI assistant, if it is useful, and ask well.** Assistants are good at explaining an error message, suggesting which function to use, and writing a first draft. Give them what a human helper would need:

- the full error message;
- a small example of your data (`df.head()`);
- the library versions;
- what you expected, and what you tried.

#### How to use chatbots

Never ask only “fix my code.” Tell the assistant what you **intended the code to do**, show the relevant data/object, and ask it to **explain the proposed change**.

Afterwards, verify the result independently whenever possible.

**Then verify.** Assistants can invent functions and arguments that do not exist, use outdated APIs (like `GC` above, because they learned from the same old tutorials), or write code that runs but computes the wrong thing. The last case is the dangerous one, because nothing fails. Treat the answer as a suggestion, and check it:

- run it on a tiny example whose answer you know;
- check shapes and ranges;
- look up any function you do not recognise.

What is allowed in graded work is set out in the course rules on Canvas.

**Test your assumptions in code.** An `assert` stops the notebook when something you believe turns out to be false:

```
assert df["pvalue"].between(0, 1).all(), "p-values must lie between 0 and 1"
assert not df["Gene"].duplicated().any(), "one row per gene"
assert len(record.seq) == 628, "HBB mRNA should be 628 nt"
```

**The notebook trap: stale state.** A notebook's variables live in memory, independently of where the cells sit on the page. If you run cell 3, then cell 7, and then edit cell 3 without re-running it, cell 7 still uses the *old* value. The page then says one thing while the memory holds another. When a result surprises you, use **Restart Kernel → Run All**. It clears the memory and runs every cell from top to bottom, and it is the only way to be sure that the output matches the code (see the box at the start of the page).

## 2.11 A tool outside Python: git

One tool outside Python is part of everyday work. **git** keeps the history of a folder (a *repository*) as a series of snapshots (*commits*). You can therefore see what changed, go back to a version that worked, and share work through GitHub. In this course you run git on your own laptop, in VS Code's terminal: clone the course repository there, open its notebooks in VS Code, and run them on a Colab machine. Keep in mind that the files you commit are the ones on your laptop, while anything a notebook writes while it runs ends up on the Colab machine (see [Setup](#)).

```
git clone https://github.com/ElofssonLab/kb8029-book.git # a copy of the course notebook
git status                                             # what has changed since the last commit?
git add analysis.ipynb                               # choose what goes into the next snapshot
git commit -m "Add volcano plot"                     # take the snapshot, with a message saying why
git log --oneline                                     # the history
git pull                                              # fetch others' changes (git push sends yours to yo
```

Commit small, working steps, with messages that say *why*. Do not commit large data files, passwords or keys; list them in a `.gitignore` file instead. For further reading, see the Software Carpentry lessons [The Unix Shell](#) and [Version Control with Git](#) (CC BY 4.0).

## 2.12 Mini challenge: a small analysis, start to finish

The mini challenge puts the pieces together on the globin family, the paralogous genes that Session 4 aligns:

1. **Fetch** the six human globin mRNAs from NCBI: *HBB* (NM\_000518.5), *HBA1* (NM\_000558.5), *HBA2* (NM\_000517.6), *MB* (NM\_001382809.1), *NGB* (NM\_021257.4) and *CYGB* (NM\_134268.5).
2. **Represent** them as a DataFrame: one row per gene, with the mRNA length, GC fraction and the length of the protein, from the record's CDS feature.
3. **Calculate** something: how much of each mRNA is coding sequence?
4. **Plot** something: protein length against mRNA length.
5. **Explain** what you found, in two or three sentences.

The notebook has a worked solution, which finds the following:

Table 2.2: The globin genes of the mini challenge, from the notebook's worked solution.

| Gene        | mRNA (nt) | GC fraction | Protein (aa) | Untranslated (nt) |
|-------------|-----------|-------------|--------------|-------------------|
| <i>HBB</i>  | 628       | 0.51        | 147          | 184               |
| <i>HBA1</i> | 577       | 0.64        | 142          | 148               |
| <i>HBA2</i> | 576       | 0.64        | 142          | 147               |
| <i>MB</i>   | 1,286     | 0.55        | 154          | 821               |
| <i>NGB</i>  | 1,778     | 0.62        | 151          | 1,322             |
| <i>CYGB</i> | 1,962     | 0.62        | 190          | 1,389             |

The proteins are all about the same size (142-190 amino acids, one globin fold each), but the mRNAs differ more than threefold in length. Almost all of the difference is untranslated sequence (UTRs), which ranges from 147 nt in *HBA2* to 1,389 nt in *CYGB*. A transcript's length therefore says little about its protein. With six genes, Pearson's correlation between mRNA and protein length is  $r = 0.78$ , but  $p = 0.07$ , so a strong-looking correlation from few points is only weak evidence (the second discussion problem).

### Check your understanding:

The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)

## 2.13 The course notebook

`notebooks/session02-working-in-python.ipynb` follows this page section by section and runs everything above. It covers the objects, the exploration and plots of the Becares data, the tests, Biopython, the Protein and Enzyme classes and a first PyTorch model, reading local, compressed and remote data, a debugging walk-through, and the solution to the mini challenge. Open it in [Google Colab](https://colab.research.google.com/), or download it and run it on a Colab machine from VS Code.

## 2.14 The lab

The lab notebook, [notebooks/session02-lab.ipynb](https://notebooks.google.com/notebooks/session02-lab.ipynb), continues from the mini challenge, with less help at each step. Lines to complete are marked with `todo(...)`, which stops the cell with a message saying what is missing. The lab has three parts:

1. **Fill in:** complete the marked lines in working code. You explore the same RNA-seq table with other columns and thresholds, plot first and test second, run Seq operations on a mystery fragment, fetch its record from NCBI into a local file and back, and finish a small Transcript class for that record.

2. **Adapt:** change working code to answer new questions. You find the most strongly down-regulated gene, fix a cell with three realistic bugs (one of which runs without an error), and apply the mini challenge's function to six more globin genes.
3. **Solve:** one open challenge, with a specification and hints only. Across all twelve globins, does a longer mRNA mean a longer protein? You explain what you found in a few sentences, including what to do with the one gene that stands out.

Three more challenges follow as **optional extras**, which are not part of the quiz. In the first you download a gene (human leptin) with Entrez and Biopython and plot GC content along its mRNA. The second explores the RNA-seq table with seaborn, and the third is a statistical analysis, plot first and test second, of whether up-regulated genes are expressed at a different level than down-regulated ones.

**If time is short, take the core path first:** Part A and B2 (fix three bugs). They are the core of the session (a data table, plots, sequences, NCBI, your own class, and debugging), and their questions are worth more than half the quiz points. Then do B1, B3 and Part C, which build on them, and the optional extras last.

Answer the Session 2 lab quiz on Canvas with your notebook's own output.

### 2.14.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## 2.15 Further watching

The animations in the 3Blue1Brown videos used throughout this course are made with **Manim**, an open-source Python library written by Grant Sanderson, the creator of 3Blue1Brown. The two videos below show Python used for something very different from data analysis.

[\*Animate Like 3Blue1Brown - No Coding Required \(Manim Animation Using A.I.\)\*](#) (Sycamore), generating Manim animations with an AI assistant. Watch it with Section 2.10 in mind: how do you check that what the assistant produced is what you wanted?

Video: [Animate Like 3Blue1Brown - No Coding Required \(Manim Animation Using A.I.\), Sycamore](#)

[\*How I animate 3Blue1Brown | A Manim demo with Ben Sparks\*](#) (3Blue1Brown), a long demonstration of how the videos are built, scene by scene, in code:

Video: [How I animate 3Blue1Brown, A Manim demo with Ben Sparks](#)

## 2.16 What's next

Session 3 uses Biopython's database access at full scale, to retrieve and cross-reference real sequences across biological databases. Tensors and autograd return in Session 7's gradient-descent lab, and in Session 8 the classes of Section [2.8](#) return as PyTorch models.

## 3 Session 3: Biological Databases

### Note

**Sources for this page:** Petras Kundrotas’s KB8029 lecture “Databases of Sequence Information” (Feb 2026) — this page covers its databases and worked examples, reorganized around five questions and adapted for this book, not copied verbatim; Wikipedia, [Biological database](#), [Sequence database](#), [International Nucleotide Sequence Database Collaboration](#), [GenBank](#), [UniProt](#), [RefSeq](#), [Accession number \(bioinformatics\)](#), [Genome browser](#), [UCSC Genome Browser](#), [Ensembl genome database project](#), [PubMed](#), [PubMed Central](#), [MEDLINE](#), [BioRxiv](#), [Boolean search](#) — CC BY-SA 4.0; NCBI, [GenBank overview](#), [About RefSeq](#), and [RefSeq FAQ](#) — public domain (US government work); EMBL-EBI Training, [What is ENA?](#) and [The UniProt databases](#) — CC BY 4.0. Every accession number, record and search-result count quoted below was retrieved live from the NCBI, UniProt, InterPro, RCSB, AlphaFold DB and IntAct APIs (2026-10-04); the notebook contains the code that produced them. The UniProt protein-existence levels follow UniProt’s [documentation](#).

### Naming things

**Why do databases trust accession numbers more than gene names?** Ask the gene *Pokemon*, renamed *Zbtb7* in 2005 after Pokémon’s owners objected.

## 3.1 Learning goals

By the end of this session you should be able to:

- follow one gene or protein across databases, from its genomic and transcript records to its protein entry, family, structure, interactions and literature, using the cross-references between them;
- tell apart a gene symbol, a database accession, an accession with its version, and a database-specific entry name, and explain why one biological entity has many linked records;
- say where a particular claim in a record comes from: data **submitted** by the people who produced it, **curated** by database staff, **predicted** by a program, or **experimentally supported**, and recognise the signs of each (RefSeq NM/NP vs. XM/XP, Swiss-Prot vs. TrEMBL, UniProt’s protein-existence level, an experimental structure vs. a predicted model);

- build a PubMed query with Boolean operators and quotation marks, and predict its result before running it.

Rather than listing databases one by one, the session follows one protein, human beta-globin (*HBB*), through five questions: **Where is the sequence? What is known about the protein? Which family does it belong to? Is there a structure? What does it interact with?** Each database then appears as the answer to one of these questions.

## 3.2 Why coordinated, worldwide databases

Sequencing produces far more data than any single lab, or any single country, can curate. UniProtKB's unreviewed section (TrEMBL) alone holds about 149 million protein entries. A private copy of "the" database would therefore go stale the moment anyone else submitted a sequence, so the databases have to be shared.

The field's answer is the **International Nucleotide Sequence Database Collaboration (INSDC)**, which joins GenBank (NCBI, USA), the European Nucleotide Archive (ENA, EMBL-EBI, Europe) and DDBJ (Japan). The three exchange all new data every day and follow shared annotation standards. A sequence submitted to any one of them can therefore be found through all three, under the same accession. Each presents it in its own way, though; GenBank and ENA, for example, use different flat-file formats and different web pages. Access has been free and unrestricted from the start, and since the 1990s most journals have required new sequences to be deposited in an INSDC database before publication. This is why a sequence you look up today reflects essentially everything published on it.

### 3.2.1 Further reading

- Wikipedia, [Biological database](#) and [International Nucleotide Sequence Database Collaboration](#)
- EMBL-EBI Training, [What is ENA?](#)

## 3.3 Two ideas you need for every database

### 3.3.1 Identifiers: names, accessions, versions and cross-references

The most useful skill of this session is to know what kind of identifier you are holding, because one gene has many. For *HBB* alone, the table lists seven:

Table 3.1: Seven identifiers for the human *HBB* gene and its products.

| Identifier  | Kind                 | What it means                                                                                                                            |
|-------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| HBB         | approved gene symbol | assigned by the HGNC (HUGO Gene Nomenclature Committee) for human genes, and widely used across databases (which may also list synonyms) |
| 3043        | NCBI Gene ID         | NCBI's number for the gene record                                                                                                        |
| NM_000518   | accession            | one RefSeq transcript record; unique only within RefSeq                                                                                  |
| NM_000518.5 | accession.version    | the fifth version of that record; the version changes whenever the sequence changes                                                      |
| P68871      | UniProt accession    | the stable identifier of the UniProt entry                                                                                               |
| HBB_HUMAN   | UniProt entry name   | a readable label for the same entry, which can change                                                                                    |
| 2HHB        | PDB ID               | one experimental structure; <i>HBB</i> appears in hundreds of them                                                                       |

An accession is unique only within its own database. The same gene has different accessions in RefSeq, Ensembl, UniProt and the PDB, and the records are tied together by explicit **cross-references**. The RefSeq protein record NP\_000509.1, for example, states that it was translated from NM\_000518.5, and the UniProt entry P68871 lists both. One biological entity thus has many linked records.

Two habits follow from this. First, to move between records, follow an explicit cross-reference when one exists. Otherwise, search with a stable identifier or the approved symbol (and the organism), and check the organism and sequence of what you find. Second, when you report a sequence, give the accession *with* its version, because NM\_000518 alone does not say which revision you used. The same holds for genome coordinates, which only mean something together with the genome **build** (GRCh37/hg19 or GRCh38/hg38). The lab shows the same gene at different coordinates in the two builds.

RefSeq accessions also encode the kind of record in their prefix:

Table 3.2: RefSeq accession prefixes and the kind of record they denote.

| Prefix | Record                                | Example      |
|--------|---------------------------------------|--------------|
| NC_    | genomic: a complete chromosome        | NC_000011.10 |
| NM_    | mRNA, not predicted (many curated)    | NM_000518.5  |
| NP_    | protein, not predicted (many curated) | NP_000509.1  |
| NR_    | non-coding RNA, not predicted         | —            |

| Prefix    | Record                                                                        | Example        |
|-----------|-------------------------------------------------------------------------------|----------------|
| XM_ / XP_ | mRNA / protein <b>predicted</b><br>by NCBI's<br>genome-annotation<br>pipeline | XP_011536997.1 |

### 3.3.2 Where does a record's information come from?

Databases differ less in *what* they store than in *who vouches for it*. A claim in a record can come from four sources, listed below from the least to the most checked. These sources describe where a particular piece of information comes from, not four separate kinds of record. One Swiss-Prot entry, for example, mixes curated, experimental and predicted annotation, and a PDB entry is both submitted and experimental.

- **Submitted (archival).** INSDC stores sequences close to how the submitters deposited them, together with their annotation. Nobody else revises it, so the quality varies, and the same gene may appear in many overlapping records.
- **Predicted.** A program inferred it. An XM\_/XP\_ record says so itself: "MODEL REFSEQ: This record is predicted by automated computational analysis." Most TrEMBL entries are translations of predicted coding sequences, annotated automatically. An AlphaFold model is a predicted structure.
- **Curated.** Database staff have read the evidence and merged the records into one reviewed entry. RefSeq is a non-redundant collection built from INSDC data (it is not itself part of INSDC). Its NM\_/NP\_ records are the non-predicted class, many of them curated or supported by transcript and protein evidence. Its XM\_/XP\_ records, which sit side by side with them, are computational models. Swiss-Prot entries are reviewed by UniProt curators.
- **Experimentally supported.** Somebody measured it. A PDB entry holds coordinates fitted to experimental data, an IntAct interaction comes from a published experiment, and UniProt marks the proteins whose existence has been observed directly.

UniProtKB makes the distinction between curated and predicted explicit, because it has **two sections**. **Swiss-Prot** holds about 576,000 entries reviewed by curators, in which a person has checked the function, domains, modifications and literature, with one entry per protein. **TrEMBL** holds about 149 million unreviewed, automatically annotated entries. Swiss-Prot is not a subset of TrEMBL. The two are separate sections, and when curators review a TrEMBL entry, it is integrated into Swiss-Prot and leaves TrEMBL. Every entry also has a **protein-existence level**, graded from the strongest evidence to the weakest: 1, experimental evidence at protein level; 2, at transcript level; 3, inferred from homology; 4, predicted; 5, uncertain. *HBB* (P68871) is at level 1.

The discussion problems practise both ideas. You name the level and kind of four RefSeq records from their prefixes, and then find a UniProt entry from an accession or from the gene symbol.

### 3.3.3 Further reading

- Wikipedia, [Accession number \(bioinformatics\)](#), [RefSeq](#) and [UniProt](#)
- NCBI, [About RefSeq](#) and [RefSeq FAQ](#), with the full prefix list
- EMBL-EBI Training, [The UniProt databases](#)

## 3.4 Following one protein through five questions

### 3.4.1 Where is the sequence?

The first question is where the sequence is, and the answer depends on the level. Databases are organised by the central dogma, so DNA, RNA and protein each have their own records. For *HBB*, NCBI Gene (3043) places the gene on chromosome 11 (11p15.4). The chromosome itself is NC\_000011.10, the curated transcript is NM\_000518.5 and its protein is NP\_000509.1. The raw submitted sequences behind them sit in INSDC, while RefSeq and **Ensembl** provide the annotated, non-redundant view of the genome, with the gene models marked out.

To see a gene in its genomic context, you use a **genome browser**. The UCSC Genome Browser and Ensembl show the sequence and its annotation as a function of position along a chromosome, for a chosen organism and genome build.

### 3.4.2 What is known about the protein?

**UniProt** is the central protein resource. P68871 (HBB\_HUMAN) is a Swiss-Prot entry, so its function, the diseases its variants cause (such as sickle-cell anaemia), its modifications and its literature have been checked by curators. It also links to RefSeq, Ensembl, InterPro, the PDB, AlphaFold DB, IntAct and many more. In practice, the reviewed UniProt entry is therefore the best first stop for “what is known about this protein?”.

A third UniProt component, **UniParc**, stores every unique protein sequence once, whichever database it came from, under an identifier that is never reused. It is how you recognise that records in different databases hold the same sequence.

#### 3.4.2.1 Further watching

[Welcome to UniProt](#) (UniProt’s own introduction to the resource):

Video: [Welcome to UniProt, UniProt](#)

### 3.4.3 Which family does it belong to?

A family or domain database tells you which group a sequence belongs to, which is often the fastest way to learn something about a new or poorly characterised protein. **InterPro** (EMBL-EBI) integrates protein family, domain and site signatures from over a dozen member databases. One of them is **Pfam**, whose own website was retired in 2023 and merged into InterPro. P68871 matches five InterPro entries, nested from broad to specific:

Table 3.3: The five InterPro entries that match  $\beta$ -globin (P68871), from broad to specific.

| InterPro accession | Name                                       |
|--------------------|--------------------------------------------|
| IPR009050          | Globin-like superfamily                    |
| IPR012292          | Globin/Protoglobin                         |
| IPR000971          | Globin                                     |
| IPR050056          | Hemoglobin and related oxygen transporters |
| IPR002337          | Hemoglobin, beta-type                      |

Most of these signatures are profile hidden Markov models, the method that Session 5 explains. For non-coding RNA, **Rfam** plays the same role. It uses covariance models, which score the conserved base-pairing pattern together with the sequence, because an RNA's function depends on its fold. *HBB* is protein-coding and has no Rfam entry; Rfam's tRNA family, for comparison, is RF00005.

### 3.4.4 Is there a structure?

The question of structure has two very different kinds of answer:

- **Experimental structures** are stored in the **Protein Data Bank (PDB)**. Entry 2HHB is human deoxyhaemoglobin, two alpha and two beta chains, determined by X-ray diffraction at **1.74 Å resolution** (Fermi et al. 1984). Resolution describes the experimental data: the smaller the number, the finer the detail the data could resolve.
- **Predicted models** are stored in the **AlphaFold Protein Structure Database**, for essentially every protein in UniProt. Each residue carries a confidence score, **pLDDT** (0-100). Above 90, the model is expected to be accurate down to the side chains, and above 70 the backbone is usually right. Below 50, the model confidence is very low, and intrinsic disorder is one common reason (Session 12). The model for P68871 has a mean pLDDT of 97.

A resolution and a pLDDT are therefore not the same kind of number. Resolution measures how much detail the experiment could see in a real molecule, whereas pLDDT is the model's own estimate of how right its prediction is. For *HBB*, which has hundreds of experimental structures, the model adds little. AlphaFold DB matters most for the many proteins that have none. Session 6 treats these coordinates as data, aligning structures, measuring them and searching for structural relatives, and Session 12 shows how AlphaFold predicts them.

### 3.4.5 What does it interact with?

Interaction databases, too, differ in the kind of evidence they hold. They are listed here from the most to the least direct evidence:

- **IntAct** (EMBL-EBI, part of the IMEx consortium) curates individual physical interactions from published experiments, such as that of *HBB* with HBZ (hemoglobin subunit zeta).
- **Complex Portal** (IntAct's sister resource) curates stable complexes with their composition. CPX-2158, "Hemoglobin HbA complex", consists of two HBA1, two HBB and four haem groups, the same object whose atoms 2HHB shows.
- **STRING** combines curated interactions, co-expression, conserved gene neighbourhood, predictions and text mining into one score per protein pair. It answers a broader question, "which proteins are functionally associated?", with a mixture of experimental and predicted evidence. **FunCoup** (Stockholm University) builds a similar network in a different way, and a later course in this programme covers it.

### 3.4.6 The dossier

Put together, the answers to the five questions form a short **database dossier** for *HBB*. Each row is a different database's record of the same biology, and each is reached from the previous one by a cross-reference:

Table 3.4: The databases that answer each question about  $\beta$ -globin, and the main evidence behind each answer.

| Question                         | Database               | Record                                                   | Main evidence for the claim                          |
|----------------------------------|------------------------|----------------------------------------------------------|------------------------------------------------------|
| Where is the sequence?           | NCBI Gene; RefSeq      | 3043; NC_000011.10<br>→ NM_000518.5 →<br>NP_000509.1     | curated (RefSeq)                                     |
| What is known about the protein? | UniProt (Swiss-Prot)   | P68871 (HBB_HUMAN),<br>protein existence<br>level 1      | curated; existence<br>experimental                   |
| Which family?                    | InterPro               | IPR000971 (Globin)<br>and four more                      | signatures built by<br>curators, matches<br>computed |
| Is there a structure?            | PDB; AlphaFold DB      | 2HHB (X-ray, 1.74 Å);<br>AF-P68871-F1<br>(mean pLDDT 97) | experimental;<br>predicted                           |
| What does it interact with?      | IntAct; Complex Portal | HBB-HBZ;<br>CPX-2158 (HbA)                               | experimental;<br>curated                             |

| Question               | Database | Record                                                                                   | Main evidence for the claim |
|------------------------|----------|------------------------------------------------------------------------------------------|-----------------------------|
| Where is it described? | PubMed   | PMID 6726807<br>(Fermi et al. 1984, <i>J Mol Biol</i> 175:159),<br>the paper behind 2HHB | literature                  |

The notebook fetches every one of these records live, and the lab ends with a dossier of your own, for human myoglobin.

### 3.4.7 Further reading

- EMBL-EBI, [InterPro](#), [PDB](#), [AlphaFold DB](#), [IntAct](#) and [Complex Portal](#)
- Rfam, [RF00005 \(tRNA\)](#); STRING, [string-db.org](#); FunCoup, [funcoup.org](#)

## 3.5 Searching: know your level, predict, then check

A search at the wrong level buries what you want. NCBI’s or EBI’s everything-at-once search for “hemoglobin beta” mixes genome, gene, transcript and protein records, whereas the same words in NCBI Gene or UniProt give a short list with the record you want near the top. Once you know the level, you should therefore use that database’s own search. When you need many records, or want a search to be reproducible, use the programmatic interfaces instead (NCBI’s E-utilities through Biopython’s `Bio.Entrez`, and the UniProt and PDB REST APIs), as the notebook does.

The same applies to the literature. **PubMed**, built by NCBI on the NLM’s MEDLINE database (started 1971), indexes the biomedical literature, with just over **41 million** records. Its free full-text archive, **PubMed Central (PMC)**, holds about **12.6 million**, roughly a third. A PubMed hit is therefore not necessarily a free paper. When PMC has no copy, a preprint server such as **bioRxiv** often does.

A good habit for every database is to **predict what a query will return, run it, and check whether the result makes sense**. PubMed’s Boolean operators make this easy, because they behave like operations on sets:

Table 3.5: PubMed queries combined with Boolean operators.

| Query                  | Result count | Prediction                        |
|------------------------|--------------|-----------------------------------|
| hemoglobin             | 288,577      | —                                 |
| malaria                | 125,442      | —                                 |
| hemoglobin AND malaria | 4,458        | both terms present                |
| hemoglobin OR malaria  | 409,561      | 288,577 + 125,442 – 4,458 (union) |

| Query                  | Result count | Prediction                      |
|------------------------|--------------|---------------------------------|
| hemoglobin NOT malaria | 284,119      | 288,577 – 4,458<br>(difference) |

**Quotation marks** are the second tool. The query hemoglobin beta is read as hemoglobin AND beta, that is, both words anywhere, in any order. The query "hemoglobin beta" requires the exact phrase, and it usually returns far fewer records. The third discussion problem asks you to predict the counts for a new pair of terms before running the searches.

### 3.5.1 Further watching

[PubMed Overview](#) (Johns Hopkins Welch Medical Library):

Video: [PubMed Overview, Welch Medical Library](#)

### 3.5.2 Further reading

- Wikipedia, [Genome browser](#), [PubMed](#), [PubMed Central](#) and [Boolean search](#)

#### **i** Other resources worth knowing

These resources are not needed for this course, but you will meet them:

- **Variants:** dbSNP and ClinVar (NCBI; ClinVar adds clinical significance), the European Variation Archive (EMBL-EBI), and the GWAS Catalog, which records statistical associations between variants and traits from genome-wide association studies. OMIM (Johns Hopkins) is the classic reference for genes and Mendelian disease.
- **Cancer genomics:** TCGA, through NCI's Genomic Data Commons, and COSMIC (Sanger Institute) for somatic mutations in cancer.
- **Systems:** PRIDE (mass-spectrometry proteomics data), Reactome (curated pathways) and BioModels (computational models of pathways).

## 3.6 The course notebook

notebooks/session03-databases-in-code.ipynb builds the *HBB* dossier with live code. It retrieves the gene, mRNA and protein records from NCBI through Bio.Entrez, the UniProt entry with its cross-references and protein-existence level, the InterPro matches, PDB entry 2HHB through two different APIs (with its primary citation) and the AlphaFold model, the IntAct and Complex Portal records, and STRING. It then counts the Swiss-Prot and TrEMBL entries and runs the Boolean PubMed searches

above, checking the set arithmetic in code. Open it in [Google Colab](#) via the badge at the top of the notebook page.

## 3.7 Check your understanding

*The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)*

## 3.8 The lab

The lab follows the same path on other genes and proteins. It covers identifiers and genome builds, predicted and curated records, a reviewed UniProt entry and its protein-existence level, InterPro domains, an experimental structure and an AlphaFold model, complexes and networks, and a literature search. It ends with a **database dossier** for human myoglobin, in which you also say what kind of information each record holds.

### 3.8.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current worksheet, the deadline and how to submit.

## 3.9 What's next

Session 4 picks up where the cross-references leave off. Once you can retrieve two related sequences, the next question is how to line them up and score how similar they are, and that is sequence alignment.

## 4 Session 4: Sequence Alignment

### **i** Note

**Sources for this page:** Petras Kundrotas’s KB8029 lecture “Sequence Alignment” (Feb 2026) — this page follows its structure and worked examples, adapted and shortened for this book, not copied verbatim; three figures below (filling one DP cell; the completed Needleman-Wunsch traceback; linear vs. affine gap penalties) are taken directly from that lecture’s slides, and the globin gene tree was computed for this course; Dayhoff, Schwartz & Orcutt (1978), *Atlas of Protein Sequence and Structure*, pp. 345-352 (origin of the PAM matrices); Henikoff & Henikoff (1992), *PNAS* 89:10915-10919 (origin of the BLOSUM matrices); Wikipedia, [Needleman-Wunsch algorithm](#), [Smith-Waterman algorithm](#), [Gap penalty](#), [Dot plot \(bioinformatics\)](#), [Standard score](#), [Kullback-Leibler divergence](#), [Sequence homology](#), [Point accepted mutation](#), [BLOSUM](#), [Multiple sequence alignment](#) — CC BY-SA 4.0; the notebook’s core algorithm code is adapted from Lukas Käll’s [bibook](#) — CC BY 4.0. The guide-tree worked-example figure is original to this book, not from the lecture. The globin sequence logo is drawn for this book from the Pfam Globin family (PF00042) seed alignment, via [InterPro](#) (CC0), with [logomaker](#); sequence logos were introduced by Schneider & Stephens (1990), *Nucleic Acids Research* 18:6097-6100; Wikipedia, [Sequence logo](#) — CC BY-SA 4.0.

### Homology

Two sequences walk into a bar. The bartender asks: “Are you two related?”

They answer: “Depends on your gap penalty.”

### 4.1 Learning goals

By the end of this session you should be able to:

- explain why sequence alignment needs a dedicated algorithm rather than brute-force comparison;
- work a small global alignment by hand using dynamic programming;
- explain the difference between global and local alignment;
- explain why an affine gap penalty is usually preferred over a linear one;
- explain what distinguishes the PAM and BLOSUM families of substitution matrices;
- explain why a high alignment score is not by itself significant;

- read a sequence logo, and explain how the entropy of each column of a multiple alignment shows which positions are conserved;
- read a simple phylogenetic tree, and tell orthologs from paralogs by speciation and duplication.

Session 3 ended with two related sequences retrieved from the databases. This session asks the next question: **which residues correspond to each other, and how much evidence does the alignment give for shared ancestry?** An alignment is a hypothesis about that correspondence. It proposes which residues descend from the same ancestral position, so substitutions appear as aligned pairs, and insertions and deletions as gaps. The score of the alignment then says how plausible this proposed history is *under the chosen scoring scheme*. That is not the same as recovering the true evolutionary history, and much of this session is about the choices that go into the score.

## 4.2 Why alignment needs an algorithm

By definition, two sequences are **homologous** if and only if they share a common evolutionary origin. We cannot observe that ancestry directly, so we have to *infer* it from statistical evidence, usually the similarity (or identity) of the two sequences. Both measures require an alignment. **Identity** is the fraction of aligned positions with the same residue. **Similarity** is defined more loosely and also counts residues with shared physicochemical properties (two hydrophobic residues, say). High identity or similarity is therefore *evidence* for homology, not proof of it.

Since homology is defined by a common ancestor, it follows that homologs come in two kinds, depending on the event that separated them:

- **orthologs** are homologs separated by a **speciation** event, such as myoglobin in human and in mouse;
- **paralogs** are homologs separated by a **gene duplication**, such as  $\alpha$ - and  $\beta$ -globin, which descend from a duplicated ancestral globin gene.

The two definitions refer to events, not to genomes, so paralogs need not be in the same genome. Human  $\alpha$ -globin and mouse  $\beta$ -globin are paralogs too, because the duplication that separated the  $\alpha$ - and  $\beta$ -globin genes happened before human and mouse diverged. Telling the two kinds apart therefore requires the order of events, and the section “From alignment to tree” below shows how a gene tree makes that order visible.

To measure identity, we first have to decide which residues to compare. When two sequences have the same length, this is trivial: we place one under the other and count the matches. Real sequences usually differ in length, because insertions and deletions have accumulated over evolutionary time. Where we place one sequence under the other then changes the answer dramatically. Two example globin sequences that are 21.5% identical when aligned correctly can look as low as 4% identical if placed naively. Finding the right placement is therefore the real problem.

Could we simply try every possible placement and keep the best one? Unfortunately not. For two sequences of length  $n$ , allowing gaps, the number of possible alignments grows so fast that for  $n = 100$  it is already around  $10^{59}$ , far too many to search exhaustively. This is why **dynamic programming** algorithms are used for this problem. They find the guaranteed-optimal alignment without ever checking most of the possibilities. The trick is that many candidate alignments share the same beginnings. Dynamic programming therefore solves the best alignment of every pair of prefixes once, stores it, and reuses it. Before we turn to the algorithm, it helps to see what a good alignment looks like.

#### 4.2.1 Further reading

- Wikipedia, [Sequence homology](#): orthology, paralogy and the other relatedness terms, in more depth than needed here

### 4.3 Visualising alignments: dot plots

The simplest way to see whether two sequences are related is a **dot plot**, and it needs no algorithm at all. We put one sequence along each axis of a grid and mark a dot wherever the row and column residues match. A run of matches then lines up as a diagonal line, which the eye picks out immediately. A long alignment shows up as one long diagonal, a repeat as several parallel diagonals, and an inversion as a diagonal running in the opposite direction. A dot plot gives no score and no optimal alignment, so it is only a fast qualitative screen. It is still used today to visualise the output of real alignments (BLAST's web output includes one), because a picture answers "is there a match here at all, and roughly where?" faster than a score does. To get an actual alignment, and a score for it, we need dynamic programming.

#### 4.3.1 Further reading

- Wikipedia, [Dot plot \(bioinformatics\)](#) — filtering/windowing techniques for noisy real sequences, beyond the simple version above

### 4.4 Dynamic programming: Needleman-Wunsch and Smith-Waterman

Dynamic programming builds a matrix  $V$  with one extra row and column (for handling the sequence starts), fills it in using a simple rule, and then traces back through it to read off the best alignment. Each cell  $V(i, j)$  holds the score of the best alignment of the first  $i$  residues of one sequence with the first  $j$  residues of the other. It is filled as the best of three options: extending the alignment diagonally with a match or mismatch, or opening a gap from above or from the left.

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + s(x_i, y_j) \\ V(i-1, j) + g \\ V(i, j-1) + g \end{cases}$$

Here  $s(x_i, y_j)$  is a substitution score (see below) and  $g$  is a gap penalty. In practice, filling one cell means computing all three candidates and keeping the largest. The figure shows this for cell  $V(1, 1)$  of a pair of 13- and 12-residue sequences, with a gap penalty of  $g = 1$ :

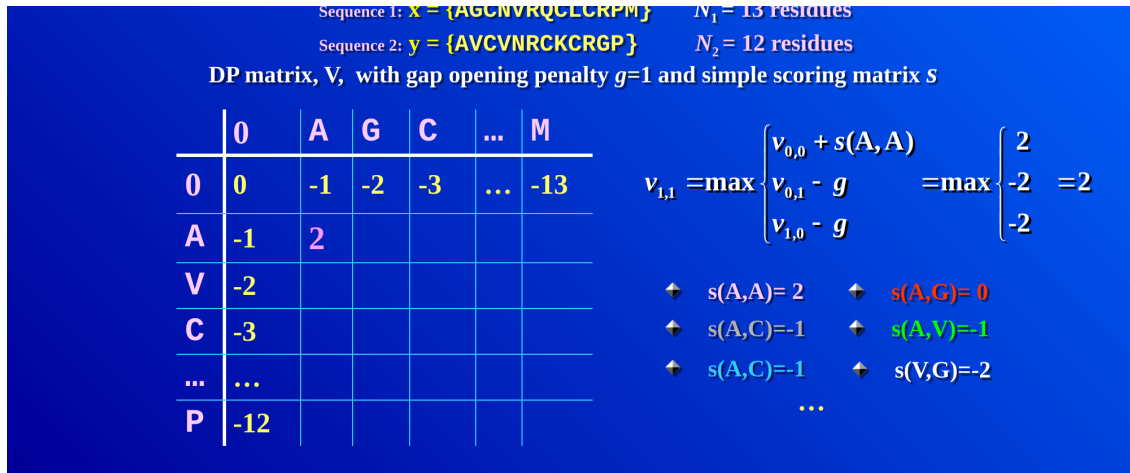


Figure 4.1: Filling one cell,  $V(1, 1)$ , of the dynamic-programming matrix: all three candidates are computed and the largest is kept (gap penalty  $g = 1$ ).

Every other cell is filled in the same way, and only the numbers that go into the three candidates change. **Worked example.** The following case is small enough to do completely by hand. We align  $x = \text{GCAT}$  against  $y = \text{GAC}$ , with match  $= +1$ , mismatch  $= -1$  and gap  $= -1$ :

Table 4.1: The dynamic-programming matrix for aligning  $x = \text{GCAT}$  with  $y = \text{GAC}$  (match  $+1$ , mismatch  $-1$ , gap  $-1$ ). The traceback path of the optimal alignment is in bold.

|          | G  | A        | C        |
|----------|----|----------|----------|
| <b>0</b> | -1 | -2       | -3       |
| <b>G</b> | -1 | <b>1</b> | 0        |
| <b>C</b> | -2 | <b>0</b> | 1        |
| <b>A</b> | -3 | -1       | <b>1</b> |
| <b>T</b> | -4 | -2       | 0        |

Tracing back from the bottom-right corner (score 0) along the cells in bold in Table 4.1 gives the optimal global alignment:

G C A T  
G - A C

Here G matches G, C is deleted, A matches A, and T is mismatched with C, for a total score of  $1 - 1 + 1 - 1 = 0$ .

The same procedure works at the scale where it is actually used. The figure below shows two protein sequences of 12 and 13 residues, with the completed matrix and its full traceback path highlighted:

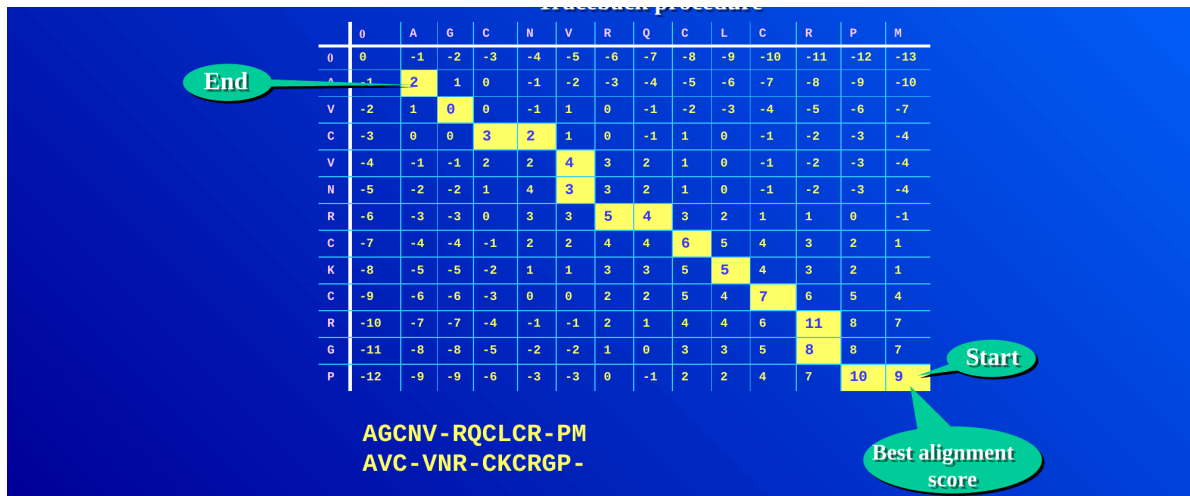


Figure 4.2: The completed Needleman-Wunsch matrix for two protein sequences of 12 and 13 residues, with the traceback path highlighted.

This algorithm, **Needleman-Wunsch**, produces a *global* alignment, in which the entire length of both sequences is forced into the result, gaps and all. Often, however, we want only the best-matching sub-region, that is, a *local* alignment. **Smith-Waterman** achieves this with one change to the same algorithm. Negative values in the matrix are reset to zero, and the traceback starts from the highest-scoring cell anywhere in the matrix (not necessarily the bottom-right corner) and stops as soon as it reaches a zero.

Which of the two to use is a biological choice, not just a mechanical one. **Global** alignment is the right choice when we expect the two sequences to correspond along most of their length, as two myoglobins do. **Local** alignment is the right choice when they may share only a part, such as one domain common to two large multidomain proteins, or a short motif inside a long sequence. Forcing a global alignment on such sequences buries the shared region under a long tail of meaningless gaps. How the gaps are penalised is therefore the next question.

#### 4.4.1 Further reading

- Wikipedia, [Needleman-Wunsch algorithm](#) and [Smith-Waterman algorithm](#) — pseudocode and complexity analysis beyond what's needed here

## 4.5 Gap penalties: linear vs. affine

So far  $g$  has been a single flat number charged per gap position, and this choice matters more than it might seem. A **linear** gap penalty,  $g(n) = n \times g_{\text{op}}$  for  $n$  consecutive gaps, charges the same total whether the gaps are spread out or bunched together. Evolutionary insertions and deletions often affect contiguous stretches of residues. It is therefore usually more realistic to charge more for opening a gap than for extending one. A linear penalty, in contrast, can produce implausible alignments that scatter single gaps throughout and break up motifs that are conserved as a block.

The fix is an **affine** gap penalty. It charges a larger cost  $g_{\text{op}}$  to *open* a new gap and a much smaller cost  $g_{\text{ext}}$  to *extend* an already open one ( $g_{\text{ext}} \ll g_{\text{op}}$ ):

$$g(n) = g_{\text{op}} + (n - 1) \times g_{\text{ext}}$$

With this penalty, a single long gap costs less than the same number of gap positions scattered as many short gaps, which is the behaviour we want. The figure below makes this concrete. It shows the same two sequences aligned once with a linear penalty (top pair), where the gaps are scattered and break up the shared “QQW” motif, and once with an affine penalty (bottom pair), where the gaps are consolidated into contiguous stretches and “QQW” stays together:

The slide lists the following concepts:

- Simple substitution matrices ( $20 \times 20$ )
- Position-specific scoring matrix ( $N_1 \times N_2$ )
- Gap penalty function,  $g(n)$** 
  - Linear function**  
 $g(n) = n \times g_{\text{op}}, \quad g_{\text{op}} \sim s(x_i, x_j)$
  - Affine gap penalty function**  
 $g(n) = g_{\text{op}} + (n - 1) \times g_{\text{ext}}, \quad g_{\text{ext}} \ll g_{\text{op}}$
  - Position-dependent gap penalty function**  
 $g(n, i) = \left\lceil g_{\text{op}} + (n - 1) \times g_{\text{ext}} \right\rceil \times \exp(n_i), \quad n_i = 0 \text{ or } 1$

Sequence alignment examples:

**Linear penalty (top pair):**

```

ADRLYGG--T-W-QQWAGW
AD---GGAATQQW--KL--
  
```

**Affine penalty (bottom pair):**

```

ADRLYGGTW-QQW--AGW
AD---GGAATQQWKL---
  
```

Figure 4.3: The same two sequences aligned with a linear gap penalty (top pair) and with an affine gap penalty (bottom pair).

### 4.5.1 Further reading

- Wikipedia, [Gap penalty](#) — covers this in more depth, including the position-dependent variant shown at the bottom of the figure above, which is beyond what this course needs

## 4.6 Substitution matrices: PAM and BLOSUM

The other ingredient of the alignment score is the substitution score  $s(x_i, y_j)$ . These scores are **log-odds scores**, the logarithms of probability ratios from Session 1. Roughly, the score of residues  $a$  and  $b$  is the log of how often they are aligned in related proteins, divided by how often they would pair by chance given how common each residue is. A **positive** score therefore means that the pair is seen more often among related sequences than expected by chance, and a **negative** score that it is seen less often. Conservative substitutions (K and R, I and V) consequently score positively. Session 5's profiles use the same idea, one column at a time. Two families of substitution matrices dominate, and they differ in the data and the model they are built from:

- **PAM** (Point Accepted Mutations; Dayhoff, Schwartz & Orcutt, 1978) is based on **global** alignments of **closely related** proteins and uses an *explicit* evolutionary model. The sequences are arranged into phylogenetic trees, ancestral sequences are inferred, and substitutions are counted along the branches of the tree. Matrices for larger distances (PAM250) are then extrapolated from the one for very short distances.
- **BLOSUM** (BLOCKS SUBstitution Matrix; Henikoff & Henikoff, 1992) is instead based on **local**, ungapped alignments of **distantly related** proteins. Substitution frequencies are counted directly from conserved blocks of real multiple alignments, with no phylogenetic tree and no extrapolation. Each BLOSUM matrix (BLOSUM45, BLOSUM62, BLOSUM80, ...) is therefore derived independently of the others.

The two families are easy to confuse, because their numbering runs in *opposite* directions. A **higher** PAM number means a **more distant** comparison, so PAM250 suits more divergent sequences than PAM100. A **higher** BLOSUM number, in contrast, means a **closer** comparison, so BLOSUM80 suits more similar sequences than BLOSUM45.

### 4.6.1 Further reading

- Wikipedia, [Point accepted mutation](#) and [BLOSUM](#) — the full derivation of each matrix family, beyond the conceptual level above

## 4.7 Is the score significant?

Once a substitution matrix and gap penalties are chosen, every alignment gets a score. A high score alone, however, does not prove that two sequences are homologous. The raw score depends on the sequence lengths, the substitution matrix and the gap penalties, so there is no universal threshold for “good enough”. One intuitive way to calibrate it is to compare the observed score with the scores of the same sequences

shuffled, or of unrelated sequences, aligned in the same way. If the real score is far above almost all of them, chance is an unlikely explanation.

For local alignments this comparison can be made precise. The best scores of unrelated sequences follow a known statistical distribution, an extreme-value distribution rather than a normal one. Database-search tools use this distribution to estimate an **E-value**, the number of hits this good expected by chance in a database of this size. Session 5 explains BLAST and E-values properly.

#### 4.7.1 Further reading

- Wikipedia, [Extreme value theory](#), in its general statistical form, beyond its use here

### 4.8 Multiple sequence alignment

So far we have aligned two sequences, but a protein family usually has many members, and aligning more than two sequences at once is harder than it looks. Exact dynamic programming extends to  $N$  sequences, but its cost grows exponentially with  $N$ , so it becomes impractical for more than a few short sequences. Practical methods therefore combine three ideas:

- **Progressive alignment** builds a **guide tree** from all pairwise distances, aligns the closest pair first, and then adds sequences (or already aligned groups) one at a time. Early decisions are never revisited, so an early mistake can persist (ClustalW is the classic example).
- **Iterative refinement** starts from a progressive alignment and repeatedly re-aligns parts of it, keeping the changes that improve the score (as in MUSCLE).
- **Consistency** uses the agreement among all the pairwise alignments as extra information. If A aligns with B and B with C at a position, a multiple alignment in which A also matches C there is preferred (as in T-Coffee).

Real tools often combine several of these ideas, and which works best depends on the number, length and divergence of the sequences.

The guide tree for progressive alignment is built by the kind of hierarchical clustering used on any distance matrix. Every sequence starts as its own cluster, the closest pair of clusters is merged repeatedly, and the process stops once only one cluster remains. The merge order is the order in which the sequences are aligned. The figure shows a small worked example: five toy sequences that are almost identical except for sequence 4, their pairwise Hamming-distance matrix, and the resulting guide tree. Sequence 4 merges in only at the very last step, which reflects how clearly it differs from the other four.

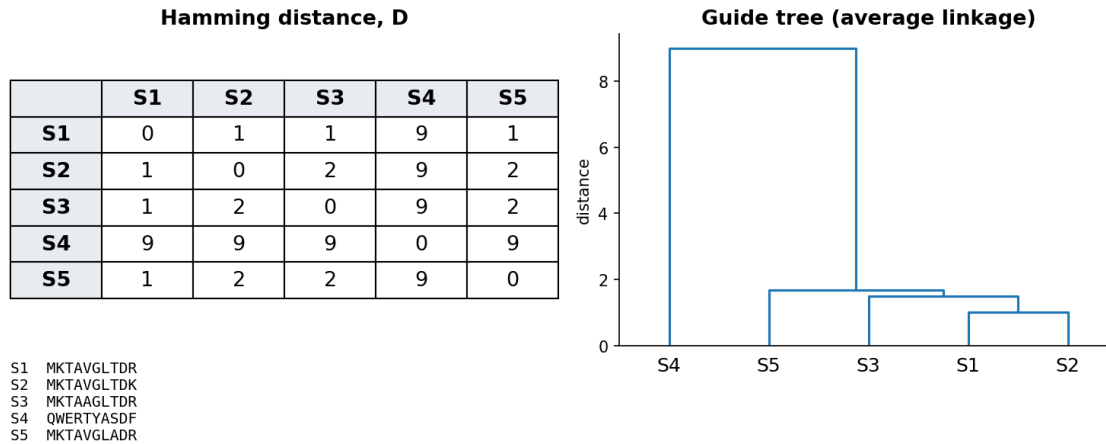


Figure 4.4: A guide tree for five toy sequences: the pairwise Hamming distances and the resulting order of merges.

A guide tree is only an *ordering device* for the alignment and not necessarily a phylogenetic tree. It is usually built by a quick method (UPGMA) that assumes all lineages evolve at the same rate, so a fast-evolving sequence can end up in the wrong place. The third discussion problem shows such a case. To infer how the sequences are actually related, we need a method built for phylogeny. The two kinds of tree look alike but answer different questions:

Table 4.2: A guide tree and a phylogenetic tree.

|                | Guide tree                                           | Phylogenetic tree                                |
|----------------|------------------------------------------------------|--------------------------------------------------|
| Question       | In which order should the sequences be aligned?      | How are the sequences related by descent?        |
| Built from     | Quick pairwise distances, often before any alignment | A multiple alignment, with an evolutionary model |
| Typical method | UPGMA (equal rates assumed)                          | Neighbour joining, maximum likelihood            |
| What it is     | A computational shortcut                             | A hypothesis about evolution                     |
| Branch lengths | Not meant to be read                                 | The amount of change                             |

## 4.9 From alignment to tree

A **phylogenetic tree** proposes how sequences are related by descent. The quickest route from an alignment to a tree uses the same ingredient as a guide tree, a matrix of pairwise distances, but combines it with a method built for phylogeny. **Neighbour joining** is such a method, because it allows lineages to evolve at different rates, which UPGMA does not. A tree is read as follows:

- The **leaves** are the sequences, and the internal **nodes** are inferred ancestors.

- The **branch lengths** show the amount of change (here, the fraction of differing residues).
- A **clade** is an ancestor with all its descendants.
- A tree is either **rooted** or **unrooted**. In a rooted tree the root marks the oldest ancestor, so the order of events can be read off. Distance methods give unrooted trees, and a root is added either with an **outgroup**, a sequence known to have branched off first, or, as here, at the midpoint of the longest path.

The notebook uses these ideas to build a tree of  $\alpha$ -globin,  $\beta$ -globin and myoglobin from human, mouse and chicken:

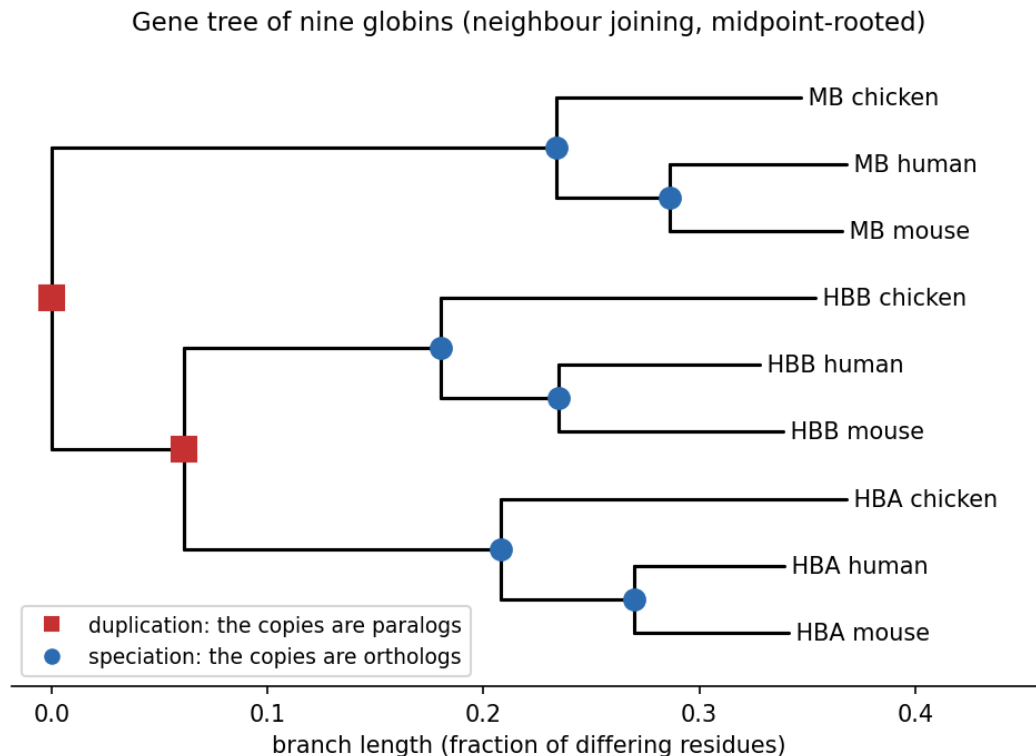


Figure 4.5: A gene tree of nine globins (neighbour joining on the fraction of differing residues in pairwise BLOSUM62 alignments, midpoint-rooted). Red squares: gene duplications, where both branches contain the same species; the copies below them are paralogs. Blue circles: speciations; the copies are orthologs.

This tree is a **gene tree**, and comparing it with the **species tree** (in which chicken branched off before human and mouse) tells the two kinds of homolog apart. The first split separates myoglobin from the haemoglobins, and the second separates  $\alpha$ - from  $\beta$ -globin. Both splits are **duplications**, because each branch still contains all three species, so the genes must have been copied before birds and mammals diverged. Every node below them follows the species tree, with human and mouse joining first, so those nodes are **speciations**. Human and mouse  $\alpha$ -globin are therefore orthologs (human  $\alpha$ -globin differs from mouse  $\alpha$ -globin at 14% of positions, and from chicken  $\alpha$ -globin at 30%), while human  $\alpha$ - and  $\beta$ -globin are paralogs (54% different).

Two cautions apply. First, a tree does not by itself say how much each branch can be trusted. This is usually measured by **bootstrap** support: the alignment columns are resampled many times, the tree is rebuilt each time, and we count how often each branch reappears. Second, real phylogenetics uses models of how residues substitute, together with maximum likelihood (for example IQ-TREE or FastTree), whereas this course only needs the concepts. The fact that sequences are related by a tree comes back in Session 8's homology-aware splits and in Session 12's coevolution.

#### 4.9.1 Further reading

- Wikipedia, [Phylogenetic tree](#), [Neighbor joining](#) and [UPGMA](#)
- Saitou & Nei (1987), [The neighbor-joining method: a new method for reconstructing phylogenetic trees](#), *Mol Biol Evol* 4:406-425

### 4.10 Reading an alignment: entropy and sequence logos

A multiple alignment of a whole family is too large to read row by row, so we need a way to summarise it. What matters is how each **column** varies. A position where every sequence has the same residue has been conserved through evolution, usually because it matters for structure or function. A position with many different residues, in contrast, is free to change.

Session 1's **entropy** puts a number on this variation. For the residue frequencies  $p_a$  in a column, the entropy is

$$H = - \sum_a p_a \log_2 p_a$$

It is 0 bits for a fully conserved column and at most  $\log_2 20 \approx 4.32$  bits for a protein column where all 20 amino acids are equally common ( $\log_2 4 = 2$  bits for DNA). The column's **information content** is the distance from that maximum,  $\log_2 20 - H$ , so it is high for conserved columns and near zero for variable ones. (This is the simple version, which assumes that all amino acids are equally common in the background. Logo programs often correct for background frequencies and for small samples.)

A **sequence logo** (Schneider & Stephens 1990) turns these numbers into a picture by drawing each column as a stack of letters. The height of the stack is the column's information content, and the height of each letter within the stack is its frequency. A tall stack with one big letter is therefore a conserved position, and a low stack of many small letters is a variable one. In the logo below, gaps also lower the stack, because the height is multiplied by the fraction of sequences that have a residue in that column.

The logo of the Pfam globin family, 73 globins from bacteria, plants, worms, insects and vertebrates, shows how few positions are conserved across the whole family:

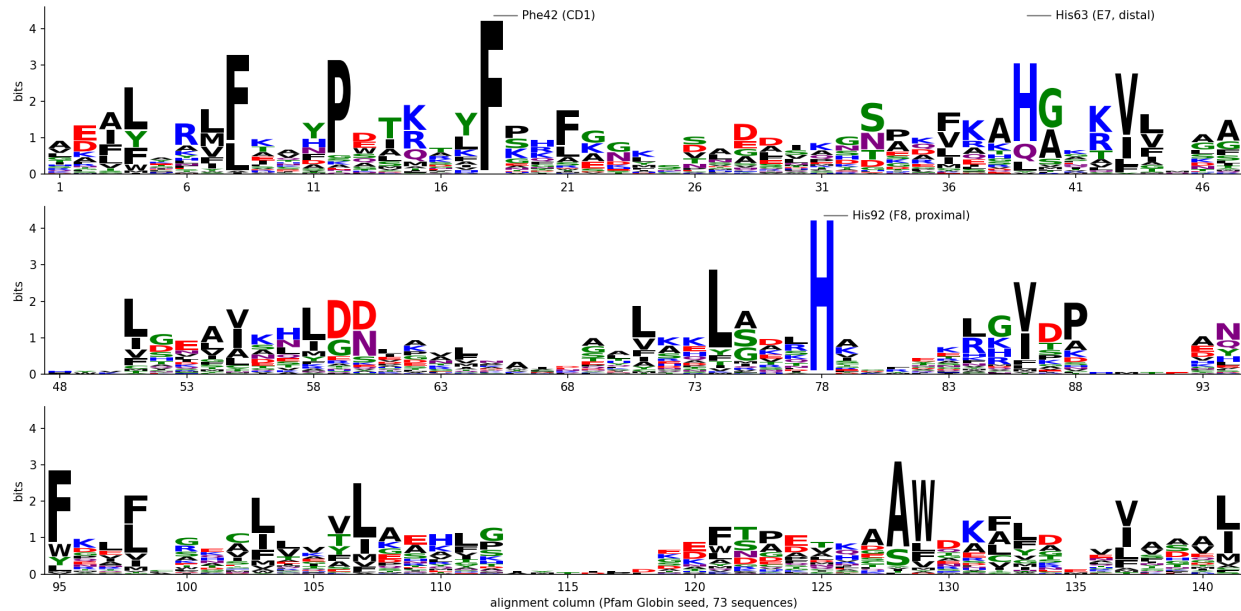


Figure 4.6: Sequence logo of the Pfam Globin (PF00042) seed alignment: 73 sequences, 141 columns. Stack height is the information content in bits; letter height is the residue's frequency. Colours by chemistry: hydrophobic black, polar green, basic blue, acidic red, amide purple. The three labelled columns are numbered as in human beta-globin (HBB, mature chain).

Only two columns are identical in all 73 sequences. One is **His92** (F8, the “proximal” histidine, whose side chain binds the haem iron), and the other is **Phe42** (CD1, which packs against the haem). A third, **His63** (E7, the “distal” histidine, on the oxygen-binding side of the haem), is His in 73% of the sequences. The average column carries only 1.2 bits. The rest of the fold is held together by positions that stay hydrophobic (the black L, V, I and F stacks, often with several residues sharing a column) rather than by identical residues. An alignment can therefore point to the residues a protein cannot do without. It also shows how the same globin fold is kept even though most pairs of sequences in this alignment share fewer than 20% of their residues.

**A motif without a family: signal peptides.** Logos are not only for families of homologs. Many proteins that are secreted, or that end up in a membrane or in an organelle of the secretory pathway, begin with a **signal peptide**. This short N-terminal sequence, typically 15-30 residues long, sends the new chain into the secretory pathway, and it is cut off once it has done its job. The signal peptides of unrelated proteins share no ancestor and cannot be aligned residue by residue, but they share a design. The first collections, of a few dozen sequences around 1980, already showed a positively charged N-terminus, then a hydrophobic stretch, then a more polar end. The three parts are now called the **n-region**, the **h-region** and the **c-region**. We can draw the design as a logo if we line the signal peptides up on the one position they all have, the **cleavage site**, instead of aligning them by homology:

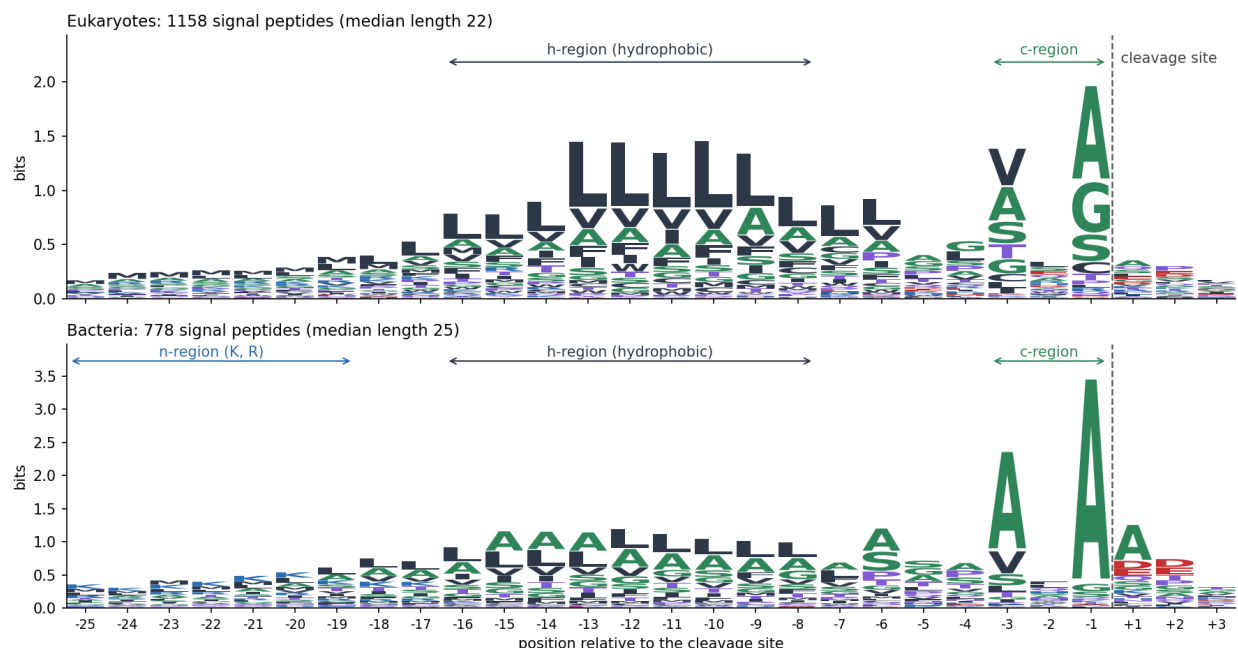


Figure 4.7: Sequence logos of experimentally supported signal peptides from UniProt (the Session 11 dataset), aligned on the cleavage site (dashed line). Positions  $-25$  to  $-1$  are the end of the signal peptide,  $+1$  to  $+3$  the start of the mature protein. Letters are coloured by property: hydrophobic dark grey, small (A, G, S) green, positive blue, negative red, other polar purple. The two logos have different y-scales.

The logo shows two kinds of signal. In the h-region no column is conserved, but every column leans the same way, towards hydrophobic residues and above all leucine. What matters there is a property of a whole stretch, not a residue at a given position. Near the cut, in contrast, two single positions stand out. Positions  $-3$  and  $-1$  carry small, uncharged residues (mostly alanine, also glycine and serine), while position  $-2$ , between them, is free. This is the **( $-3$ ,  $-1$ ) rule**, found in 1983 in a collection of 78 signal peptides (von Heijne, *Eur. J. Biochem.* 133:17). The enzyme that cuts, signal peptidase, needs small side chains at these two positions. The positively charged n-region is blurred in the logo, because signal peptides differ in length, so the left-hand columns mix residues from different regions. It is clearest in bacteria (K and R).

Like a family's logo, this description can be turned into a score (Session 5's PSSM). In 1986, von Heijne built a position weight matrix for positions  $-13$  to  $+2$  around the cleavage site (*Nucleic Acids Res.* 14:4683). The signal peptide comes back several times in the course. Session 7 compares this weight matrix with a neural network, Session 8 uses the hydrophobicity of the h-region as a feature, and Session 11 lets recurrent networks find the regions by themselves.

Two practical notes apply to logos. With few sequences, the frequencies are noisy and a column can look conserved by chance, so logo programs often apply a small-sample correction. Closely related sequences are also over-represented in most alignments, so real tools weight the sequences as well.

A multiple alignment is therefore more than a picture, because each column gives a distribution of amino acids. Session 5 turns these distributions into position-specific scores.

#### 4.10.1 Further reading

- Wikipedia, [Multiple sequence alignment](#) — additional methods and applications, e.g. profile HMMs, which Session 5 picks up in full
- von Heijne (1990), [The signal peptide](#), *J. Membr. Biol.* 115:195, a short review of the signal-peptide design
- Wikipedia, [Sequence logo](#); Schneider & Stephens (1990), “Sequence logos: a new way to display consensus sequences,” *Nucleic Acids Research* 18:6097-6100

### 4.11 The course notebook

`notebooks/session04-alignment-in-code.ipynb` implements Needleman-Wunsch and Smith-Waterman from scratch, adapted from Lukas Käll’s open-access [bibook](#) (CC BY 4.0). It starts with amino-acid sequences and a simple identity scoring scheme, in which a match scores +1 and any mismatch or gap −1. This is the  $V(i, j)$  recurrence above, with the identity matrix standing in for  $s(x_i, y_j)$ .

Rather than presenting a finished matrix, the notebook fills a small worked example one cell at a time, with the reasoning printed out. The example is a real HBB fragment aligned against a hypothetical mutated variant of it. You then complete a partially filled matrix yourself before the real one is revealed, and the traceback is walked through in the same step-by-step way. Both halves of the algorithm thus appear as a sequence of small decisions rather than as a finished picture.

The notebook then replaces the flat identity scheme with a real substitution matrix, the PAM250 matrix from the “Substitution matrices” section above, loaded from Biopython rather than typed in by hand. With it, the notebook compares global and local alignment of two real globin fragments (from HBB and myoglobin) that share one conserved motif but differ elsewhere. This makes two things directly visible: the practical difference between the two algorithms, and the effect of a graded substitution matrix compared with a flat identity scheme. Finally, the notebook builds the globin gene tree of the section “From alignment to tree” by neighbour joining. Open it in [Google Colab](#) via the badge at the top of the notebook page to edit and run it yourself.

### 4.12 Check your understanding

The questions below give immediate feedback and cover the sections above. They are in the same style as the course’s live in-class peer-instruction quizzes, and the full question bank is in `quizzes/session04.md` in the repository.

*The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)*

## 4.13 The lab

The lab has four steps, all reached through [notebooks/session04-lab.ipynb](#):

1. **By hand:** fill in a small Needleman-Wunsch dynamic-programming matrix and trace back the optimal alignment yourself, on paper. This time you use a real substitution matrix (BLOSUM62) instead of the course notebook's flat identity scheme, on two real protein fragments that are not used anywhere else on this page. The lab notebook checks your work one row at a time rather than computing it for you.
2. **In code:** see how the score and the alignment from Step 1 change with different gap penalties and substitution matrices. A real search of the Swiss-Prot database, sent to EMBL-EBI's HMMER server through its web API, then shows the same kind of sensitivity in a genuine E-value. With the wrong matrix, real relatives are not found at all.
3. **In code:** build and visualise a small, real multiple sequence alignment. This makes the progressive strategy from the section above concrete, on three short real windows from proteins used throughout this course.
4. **By hand, then in code:** work out the entropy of three columns of that alignment by hand. Then draw the sequence logo of the Pfam globin family, map its most conserved columns onto human HBB, and (optionally) see how few sequences it takes to be fooled by chance.

When all four steps are done, answer the Session 4 lab quiz on Canvas using your own results.

### 4.13.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## 4.14 What's next

Sequence alignment infers which residues correspond from the sequences alone, and Session 6 asks the same question directly in 3D, with structure alignment. Before that, Session 5 takes up two problems that this session leaves open. First, exact dynamic programming such as Needleman-Wunsch does not scale to searching a new sequence against an entire database, and that is what BLAST's heuristic is for. Second, once we have many aligned sequences rather than two, we can build a *profile* from them.

## 5 Session 5: Advanced Database Search & Profile Methods

### Note

**Sources for this page:** Petras Kundrotas’s KB8029 lecture “Advanced Database Search” (Feb 2026) — this page follows its structure and adapts several of its worked examples (independently recomputed and verified, not copied), not copied verbatim; Altschul, Gish, Miller, Myers & Lipman (1990), *J. Mol. Biol.* 215:403-410 (BLAST); Altschul et al. (1997), *Nucleic Acids Research* 25:3389-3402 (PSI-BLAST and gapped BLAST); Karlin & Altschul (1990), *PNAS* 87:2264-2268 (the statistical foundation BLAST’s E-values are built on); Steinegger & Söding (2017), *Nature Biotechnology* 35:1026-1028 (MMseqs2); Korf, Yandell & Bedell, *BLAST* (O’Reilly, 2003), pp. 79-87 (the drop-off extension example, adapted); Wikipedia, [BLAST \(biotechnology\)](#), [PSI-BLAST](#), [Position weight matrix](#), [Pseudocount](#), [Hidden Markov model](#), [Pfam](#), [InterPro](#), [E-values](#) — CC BY-SA 4.0; NCBI, [BLAST Help](#) — public domain (US government work); EMBL-EBI’s [InterPro](#) — CC0. The *HBB* Pfam domain hit and all worked-example numbers below were computed or fetched live while writing this page (2026-09-18) — see the notebook.

### Profile drift

PSI-BLAST is like social media: after a few iterations, everything it finds agrees with it.

### 5.1 Learning goals

By the end of this session you should be able to:

- explain why exact dynamic programming does not scale to searching a whole database;
- describe BLAST’s seed-extend-evaluate heuristic, and how MMseqs2 scales the same ideas;
- interpret the E-value and bit score of a real BLAST result;
- construct a small position-specific scoring matrix (PSSM) by hand, and explain why pseudocounts matter;
- describe PSI-BLAST as BLAST iterated with a PSSM instead of a single sequence, and explain why iteration can drift;

- explain, at a conceptual level, what a profile Hidden Markov Model adds beyond a PSSM.

## 5.2 The scaling problem, and BLAST's heuristic answer

Session 4's dynamic programming finds the *provably optimal* alignment between two sequences, but its cost is proportional to the product of their lengths. For two sequences a few hundred residues long, this is no problem. For searching one query against a database, it is. GenBank and UniProt hold well over a hundred million sequences between them, and running full dynamic programming against every entry would take far too long to be useful. A database search therefore needs a faster method.

**BLAST** (Basic Local Alignment Search Tool) is such a method, and it trades the guarantee of optimality for speed. Instead of filling a DP matrix for every database sequence, it first looks for short exact or near-exact matches. Only the promising regions that these matches point to get the expensive alignment step. Concretely, BLAST runs in three stages:

1. **Seeding.** Break the query into short “words”, list every word that scores well against each of them, and scan the database for occurrences of those words.
2. **Extension.** Grow each seed hit outwards, in both directions, into a full local alignment.
3. **Evaluation.** Ask whether the score of the resulting alignment is better than what pure chance would produce.

Each stage is covered in its own section below. This three-stage picture is the core idea, and it is deliberately simplified. Production BLAST adds refinements such as the *two-hit* rule (only extend when two seeds lie close together on the same diagonal), masking of low-complexity regions, gapped extension, and corrections for unusual amino-acid composition. None of these refinements changes the logic of seed, extend, evaluate.

### 5.2.1 Further reading

- Wikipedia, [BLAST \(biotechnology\)](#)
- NCBI, [BLAST Help](#) — the practical reference for choosing blastp/blastn/blastx and reading real output

## 5.3 Stage 1: seeding — turning the query into fast lookups

The first stage turns the query into a set of short words. For protein sequences, a “word” is usually only 3 residues long, so a query of length 200 has 198 overlapping 3-letter words. BLAST does not search the database only for *exact* matches to these

words, because a biologically real match may substitute a residue or two without losing significance. Instead, it uses a substitution matrix (BLOSUM62, from Session 4) to precompute every 3-letter combination that scores *above a threshold*  $T$  against each query word.

**Worked example.** Scored with the real BLOSUM62 matrix, several other words besides the query word LWG itself clear a typical threshold of  $T = 12$ . The notebook computes them directly from BLOSUM62, and 49 words clear the threshold in total. The top ones are:

Table 5.1: The highest-scoring neighbour words of the query word LWG.

| Neighbour word | Score vs. LWG |
|----------------|---------------|
| LWG            | 21            |
| IWG            | 19            |
| MWG            | 19            |
| VWG            | 18            |
| FWG            | 17            |
| AWG            | 16            |
| CWG            | 16            |
| TWG            | 16            |
| YWG            | 16            |
| KWG            | 15            |

These query words and their high-scoring neighbours go into a **lookup table**, which is built once per query and records, for every 3-letter word, which query positions it would seed. BLAST then scans each database sequence once, from start to end. For every 3-letter word it reads, it asks the table “does this word seed anything in the query, and where?” Because that check is a single table lookup, the scan is fast and linear in the size of the database. Only the few places that produce a seed go on to the expensive extension step.

## 5.4 Stage 2: extension — growing a seed into a real alignment

A seed hit is only a 3-letter coincidence and not yet evidence of anything. BLAST therefore extends it outwards in both directions and recomputes a running score as it goes. The scoring uses the same substitution-matrix logic as Session 4’s dynamic programming, but without filling a full matrix. Instead, extension stops as soon as the running score has fallen too far below its own best point so far.

**The X-drop rule.** BLAST tracks the *running score* as the alignment extends, and separately the *best score seen so far*. If the running score ever falls more than a threshold  $X$  below that best score, the alignment is very unlikely to recover, so extension stops. The reported alignment is then trimmed back to the point where the best score occurred.

**Worked example**, adapted from Korf, Yandell & Bedell (2003). We treat two short strings as if they were sequences, score +1 for a matching character and −1 for a mismatch (ignoring spaces), and use  $X = 5$ :

```
query:    Thequickbrownfoxjumps...
database: Thequietbrownncatpurrs...
```

The running score climbs steadily through Thequ, dips slightly at the ick/iet mismatch, and then recovers and climbs further through brown to a peak of **9** right after ...brown/...brown. After that it falls steadily, as foxjump diverges from catpurr. Once the running score has dropped 5 points below the peak of 9, extension stops. The reported alignment is trimmed back to the peak itself, so it pairs Thequickbrown with Thequietbrown and leaves out the lower-scoring tail beyond it. The notebook reproduces this computation in code.

### 5.4.1 Further watching

[BLAST 2: seed and extend](#) (Rob Edwards, San Diego State University):

Video: [BLAST 2: seed and extend](#), Rob Edwards

## 5.5 Stage 3: is this hit significant? E-values and bit scores

Extending every seed produces a raw alignment score. A raw score alone, however, does not say whether the alignment is *meaningful* or simply what we would expect from two random sequences of that length. To decide, we need to know how scores are distributed by chance. Karlin and Altschul proved in 1990 that the scores from aligning a query against random sequence data follow an **extreme value (Gumbel) distribution**, not a normal distribution, and this result underpins BLAST's statistics.

From that distribution follows the expected number of alignments that score at least  $S$  purely by chance, the **E-value**:

$$E = Kmn e^{-\lambda S}$$

Here  $m$  and  $n$  are the lengths of the query and the database, and  $K$  and  $\lambda$  are constants of the particular scoring system (substitution matrix and gap penalties), which real BLAST output reports as “Karlin-Altschul statistics”. This is the simplified textbook form. Real BLAST replaces  $m$  and  $n$  by slightly smaller *effective* lengths (an alignment cannot start right at the end of a sequence) and adjusts for composition, but the behaviour is the same.

Two consequences of the formula matter. First, a higher score gives an exponentially smaller  $E$ . Second, a bigger database ( $n$ ) inflates  $E$  for the *same* alignment, because

chance has more opportunities to produce a false hit. An E-value is therefore only meaningful together with the database it was computed against, and the same alignment can be “significant” against a small database but not against all of GenBank.

Raw scores and E-values both depend on the scoring system, which makes them awkward to compare across different searches. The **bit score** solves this by normalising away the scoring system:

$$S' = \frac{\lambda S - \ln K}{\ln 2}$$

Bit scores from completely different searches, even ones that use different substitution matrices, can therefore be compared directly. A bit score is not a significance test on its own, though. There is no universal bit-score cutoff, because whether a given score is surprising still depends on the size of the search space it was found in. For significance, read the E-value.

**Reading real output.** A real local BLASTP search of *HBB* (beta-globin, P68871) against its five closest human globin relatives gives the following hits:

Table 5.2: A local BLASTP search of *HBB* against its five closest human globin relatives.

| Hit                  | % identity | E-value                | Bit score |
|----------------------|------------|------------------------|-----------|
| Delta-globin (HBD)   | 93.2%      | $9.3 \times 10^{-105}$ | 284       |
| Epsilon-globin (HBE) | 75.5%      | $2.1 \times 10^{-87}$  | 240       |
| Alpha-globin (HBA)   | 43.4%      | $9.0 \times 10^{-38}$  | 114       |
| Zeta-globin (HBAZ)   | 35.9%      | $3.8 \times 10^{-32}$  | 100       |
| Myoglobin (MYG)      | 16.0%      | 63                     | 10.8      |

The first four are all confident, high-quality hits, with high percent identity, astronomically small E-values and comfortably high bit scores. Myoglobin is the interesting case. It shares the globin fold and a real evolutionary origin with haemoglobin, yet BLAST reports only a weak alignment with an E-value of 63, which taken at face value is no statistical evidence at all. That number, however, only evaluates the alignment BLAST *found*. If we run the same search with the two-hit rule switched off (so that every single seed gets extended; `-window_size 0` in BLAST+), BLAST finds a full-length alignment, 145 residues at 25.5% identity, with an E-value of  $5 \times 10^{-11}$ . With the default settings, no pair of seeds happened to fall close enough together on the right diagonal, so the good alignment was never extended. This is the price of a heuristic. BLAST is fast because it skips most of the search, and occasionally it skips the answer, which an optimal Smith-Waterman search would not have missed.

Database size still matters as well. These E-values are relative to a tiny five-sequence database. Against a database the size of UniProtKB (about  $9 \times 10^{10}$  residues), the same full-length myoglobin alignment would get an E-value of about 0.007, which is significant, but only just. Relatives that have diverged further than myoglobin, such as the bacterial *truncated* haemoglobins, give no detectable pairwise signal at all, even with the optimal alignment (Session 6 shows that they still share the globin fold).

### 5.5.1 Further reading

- Wikipedia, [E-values](#)
- Original source: Karlin & Altschul (1990), *PNAS* 87:2264-2268 — the full derivation behind the formula above

## 5.6 BLAST's ideas at modern database scale: MMseqs2

BLAST was designed in 1990, when the protein databases held tens of thousands of sequences. Today's metagenomic collections hold billions, and at that scale even BLAST becomes too slow. **MMseqs2** (Steinegger & Söding, 2017) is a widely used modern search tool that keeps BLAST's logic but turns one part of it around. In simplified form, it works as follows:

1. **Index the target database by short words** (k-mers, typically 6–7 residues for proteins). The index is a table that maps each k-mer to the database sequences and positions where it occurs, and it is built once and reused.
2. For each k-mer in the query, generate the **similar** k-mers (scored with a substitution matrix, exactly like BLAST's neighbour words) and look them up in the index.
3. Keep only database sequences in which **two matching k-mers fall on the same diagonal**, that is, at the same offset between query and target position, just as in a dot plot from Session 4. Two consistent hits are much stronger evidence of a real alignment than one.
4. Score the surviving candidates quickly with an **ungapped alignment** along that diagonal, and discard the weak ones.
5. Run the expensive **gapped Smith-Waterman** alignment, heavily optimised to use the processor's vector instructions, only on the small set that is left.

The difference from classic BLAST lies in which side gets indexed. BLAST builds a lookup table from the *query* and its neighbour words, and then reads through every database sequence. MMseqs2 instead indexes the *database* once, so each query only has to look up its own k-mers. Steps 3–5 form a cascade of ever more expensive filters, each of which passes on only a small fraction of the candidates. As a result, MMseqs2 is often hundreds of times faster than BLAST at similar sensitivity. No machine learning is involved, because MMseqs2 simply engineers the same seed-extend-evaluate ideas for scale.

Like PSI-BLAST below, MMseqs2 can also iterate, turning its hits into a profile and searching again, and ColabFold uses it to build the alignments that AlphaFold needs (Session 12). Session 6 meets its structural counterpart, Foldseek, from the same group.

## 5.7 From one sequence to many: PSI-BLAST and the PSSM

Myoglobin sits near the edge of what a single sequence can detect, and more distant relatives lie beyond it. The reason is that ordinary BLAST scores everything against one fixed substitution matrix, whatever family the query belongs to, so a faint but real family signal is easily lost against the background of a large database. **PSI-BLAST** (Position-Specific Iterated BLAST) does better, because it builds a *query-specific* scoring matrix and refines it over several rounds:

1. Run an ordinary BLAST search with the query sequence.
2. Take the hits that pass the **inclusion threshold** (an E-value cutoff, by default stricter than the one used for *reporting* hits), align them to the query, and turn that alignment into a **position-specific scoring matrix (PSSM)**, described below. Hits that are displayed but fall short of the inclusion threshold do not enter the profile.
3. Search the database again, this time scoring with the PSSM instead of a generic substitution matrix.
4. Repeat, so that each round's included hits refine the PSSM further, until no new sequences pass the threshold (convergence) or a set number of rounds is reached.

This makes PSI-BLAST more sensitive than a single BLAST pass, particularly for *remote* homologs whose similarity to the original query is too faint for a fixed substitution matrix to catch.

**Iteration is not automatically better: profile drift.** Suppose that one unrelated sequence passes the inclusion threshold. It then enters the alignment and shifts the next PSSM a little towards itself. The next round finds more sequences like the intruder, which shift the profile further. After a few rounds, the “family” can be dominated by proteins that have nothing to do with the original query, often through a shared low-complexity or coiled-coil region rather than real homology. This is a general lesson that returns in the machine-learning half of the course. A model is only as good as the examples it was built from, and a model that chooses its own training data can reinforce its own mistakes. In practice, keep the inclusion threshold strict, look at what entered in each round, and check surprising late-round hits independently.

### 5.7.1 Building a PSSM, step by step

Session 4 ended with alignment columns, their entropy, and sequence logos. A logo *describes* an alignment column, whereas a PSSM turns the same column statistics into numbers that can *score* a new sequence. The recipe below is a teaching model. The real PSI-BLAST computation also weights the sequences and uses more sophisticated pseudocounts, as noted along the way, but every step serves the same purpose in the real thing.

**1. Count.** From a multiple sequence alignment, count how often each residue appears at each position. Take this small toy alignment of 5 sequences:

|            |   |   |   |   |   |
|------------|---|---|---|---|---|
| Position:  | 1 | 2 | 3 | 4 | 5 |
| Protein 1: | A | G | L | S | P |
| Protein 2: | A | G | L | T | P |
| Protein 3: | R | G | I | S | P |
| Protein 4: | A | A | L | S | Q |
| Protein 5: | A | A | L | S | P |

At position 1, 4 of the 5 sequences have A and 1 has R. At position 5, 4 have P and 1 has Q, and so on.

Real alignments are rarely this balanced. If a search returns fifty nearly identical mammalian sequences and three distant ones, raw counts would make the profile describe mammals rather than the family. Real tools therefore **reduce redundancy or weight sequences**, removing or down-weighting near-duplicates so that an over-represented subfamily does not dominate the counts. The exact recipe varies between programs, but the principle is what matters.

**2. Add pseudocounts.** Turning raw counts straight into frequencies causes two problems. First, a residue that never appears at a position gets a frequency of exactly zero, and its log-odds score becomes  $-\infty$ . Second, with only 5 sequences, the absence of a residue might be bad luck rather than a real biological signal. **Pseudocounts**, small constants added to every count before the frequencies are computed, solve both problems. In our teaching model we add +1 uniformly to the count of every residue type at every position. Conceptually, a pseudocount is a **prior**, that is, what we assume about a column before looking at the alignment (here, “every residue is possible”). With only five sequences the prior has a large influence, while with hundreds of sequences the observed counts swamp it. Real tools use smarter priors, for example by giving larger pseudocounts to residues that substitute easily for the observed ones (according to a substitution matrix).

**3. Compare to background.** Compute each residue’s **background frequency**  $q_j$ , that is, how often it occurs in proteins generally (e.g. alanine  $\approx 7.4\%$ , tryptophan  $\approx 1.3\%$ ), independently of this alignment.

**4. Compute log-odds scores.** For each position  $i$  and residue  $j$ ,

$$S_{i,j} = \log_2 \frac{f_{i,j}}{q_j}$$

Here  $f_{i,j}$  is the (pseudocount-adjusted) observed frequency. With base 2, a score of +1 means “twice as likely here as by chance”, +3 means “8× as likely”, and so on. Arranged into a matrix, with one row per alignment position and one column per residue, the scores  $S_{i,j}$  are the PSSM. (Some tools store the matrix the other way round, but the orientation is only a storage convention.) Scoring a candidate sequence against the PSSM means looking up one value per position and **summing** them. This is the weighted-sum, or dot-product, idea from Session 1, generalised to sequence positions.

The notebook builds this PSSM from the alignment above and confirms that the highest-scoring sequence the matrix can produce is AGLSP, the alignment's own consensus, as it should be.

**Why pseudocounts really matter.** With only five sequences, every frequency is a small-sample estimate, and the raw log-odds score takes it at face value. Take a position where 4 of 5 sequences show alanine (background 7.4%) and 1 shows the much rarer tryptophan (1.3%). The raw scores are  $\log_2(0.8/0.074) \approx 3.43$  for alanine and  $\log_2(0.2/0.013) \approx 3.94$  for tryptophan, so a single observation outscores four. Yet one tryptophan in five sequences could easily be chance, and the score treats it as a reliable frequency of 20%. With +1 pseudocounts the frequencies become 5/25 and 2/25, and the scores fall to 1.43 and 2.62. The pseudocounts make every score more cautious, but they do not change the ranking: tryptophan still wins. A pseudocount cannot tell a lucky single observation from a real one; only more sequences can.

**Gaps.** A PSSM contains only substitution scores, one row per position. To align a new sequence to it, PSI-BLAST therefore uses ordinary gapped alignment, the same dynamic programming as in Session 4, with the PSSM row for position  $i$  taking the place of the substitution-matrix row for query residue  $i$ . The gap-open and gap-extension penalties are the ordinary ones, the same at every position.

**A PSSM is already a linear model.** For an ungapped sequence  $x = x_1x_2 \dots x_L$ , the PSSM score is

$$S(x) = \sum_{i=1}^L w[i, x_i],$$

where  $w[i, x_i]$  is the matrix entry for residue  $x_i$  at position  $i$ . The score is thus one weight looked up per position and added up. Session 7 keeps this weighted-sum idea but *learns* the weights from labelled examples instead of deriving them from alignment counts.

### 5.7.2 Further reading

- Wikipedia, [PSI-BLAST](#), [Position weight matrix](#), and [Pseudocount](#)

## 5.8 Profile Hidden Markov Models: what a PSSM still can't represent

A PSSM gives position-specific *substitution* scores, but the alignment algorithm handles gaps separately, with the same penalties everywhere. Real families are not like that. A loop between two helices may tolerate insertions of any length, while the core of the domain almost never has one. A **profile Hidden Markov Model (profile HMM)** therefore integrates position-specific match, insertion and deletion behaviour into one

probabilistic model of the family. It uses three kinds of states, arranged in a repeating unit with one unit per alignment position:

- **Match (M)** states represent a conserved position. Each has its own per-residue emission probabilities and plays the same role as one column of a PSSM.
- **Insert (I)** states represent extra residues between conserved positions. The model can emit a residue from them and then loop back to allow more than one.
- **Delete (D)** states represent a skipped conserved position and emit nothing at all.

Special **Begin** and **End** states bookend the model. Transitions connect the M/I/D states of consecutive positions (e.g.  $M_i \rightarrow M_{i+1}$ ,  $M_i \rightarrow I_i$ ,  $M_i \rightarrow D_{i+1}$ ), and each transition has its own learned probability.

**Why “hidden”.** For an observed residue in a real sequence, the model never tells you directly whether it came from a Match state or an Insert state. That has to be *inferred* from the path through the states that best explains the whole sequence, using the emission and transition probabilities together. The states generate what you observe, but they themselves stay hidden.

This is the extra structure that PSSMs lack, and it is why profile HMMs, not PSSMs, are the standard tool behind large-scale protein family databases such as **Pfam**. A profile HMM trained on a protein family can recognise a new sequence as a member even when it has a genuinely different number of residues in a variable region. The model itself has learned *where* insertions and deletions are common in that family, which a PSSM with uniform gap penalties can only approximate.

#### What neither a PSSM nor a profile HMM can represent

Both models score a sequence position by position. The score is a sum over columns, so each position contributes independently of the others. Real proteins do not always work that way. When mutations interact, so that the effect of one depends on the residue at another position (**epistasis**), no profile can capture it. Session 9 shows a protein where this matters (all 160,000 variants at four positions of GB1), and Session 12 shows how pairwise models fitted to an alignment (direct coupling analysis) add those interactions.

### 5.8.1 Further watching

[A friendly introduction to Bayes Theorem and Hidden Markov Models](#) (Luis Serrano):

Video: [A friendly introduction to Bayes Theorem and Hidden Markov Models](#), Luis Serrano

### 5.8.2 Further reading

- Wikipedia, [Hidden Markov model](#), [Pfam](#), and [InterPro](#)

## 5.9 Worked example: a real profile-HMM hit on *HBB*

The same idea works on a real sequence at production scale. To tie this back to the *HBB* (beta-globin) locus from Session 3, we search Pfam's library of profile HMMs against the real UniProt entry P68871 (*HBB\_HUMAN*), fetched live via EBI's InterPro API. The search finds a match to **PF00042 ("Globin")**, spanning residues 27-142 of the 147-residue protein, with an E-value of around  $1.3 \times 10^{-29}$ . In a database of this size, essentially no match that good is expected by chance. The profile HMM thus does on a real sequence what the toy PSSM above did on paper: it scores how well a sequence matches a family, position by position.

## 5.10 The course notebook

`notebooks/session05-blast-and-profiles-in-code.ipynb` reproduces every worked example on this page in code. It traces the X-drop extension, builds the PSSM from the toy alignment (with and without pseudocounts, including the rare-residue-dominance check), and fetches *HBB*'s real Pfam domain hit live from the InterPro API. Open it in [Google Colab](#) via the badge at the top of the notebook page to edit and run it yourself.

## 5.11 Check your understanding

The questions below give immediate feedback and cover the sections above. They are in the same style as the course's live in-class peer-instruction quizzes, and the full question bank is in `quizzes/session05.md` in the repository.

*The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)*

## 5.12 The lab

**The full lab** covers BLAST and profile methods as one continuous exercise. (Session 4 now has its own short lab for alignment practice, so this one no longer needs to cover alignment as well.) You first identify a real, unknown gene from a raw sequence through a real NCBI BLAST search. It turns out to be a sixth real human globin-family gene, continuing this course's running thread. Three code exercises follow: a real Clustal Omega alignment and PSSM on five globin-family proteins, a real iterated profile search (jackhmmer, the profile-HMM counterpart of PSI-BLAST) that compares its first round with its third against every sequence in the PDB, and a live Pfam domain search for HIV-1's Gag-Pol polyprotein via the same InterPro API that this page's *HBB* worked example uses.

Every search in `notebooks/session05-lab.ipynb` runs on a server at EMBL-EBI, reached through its **web API**: the notebook sends the sequences, the server runs the program, and the notebook reads the result back. There is nothing to install, so the

lab runs on Colab, in the browser or from VS Code. The same pattern (submit a job, wait for it, fetch the result) works for most large bioinformatics services, and it is how you will usually run big searches from your own code.

### 5.12.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## 5.13 What's next

A remote homolog can become impossible to recognise from its sequence while its fold stays conserved. Session 6 therefore turns to structure: the four levels of protein structure, explored hands-on in PyMOL, structure alignment, and structural search with Foldseek. Session 7 then returns to the weighted sum  $S(x) = \sum_i w[i, x_i]$  that a PSSM computes. It formalises the sum as a **perceptron** and shows that its weights can be *learned* from labelled examples by gradient descent, instead of being derived from alignment statistics as this session's PSSM was.

## 6 Session 6: Protein Structure

### **i** Note

**Sources for this page:** Petras Kundrotas's KB8029 lecture "Protein Structure" (Feb 2026) — this page follows its structure for the levels- of-structure and structure-prediction material, adapted and substantially shortened, not copied verbatim; Wikipedia, [Protein structure](#), [Alpha helix](#), [Beta sheet](#), [Anfinsen's dogma](#), [Homology modeling](#), [Threading \(protein sequence\)](#), [Root-mean-square deviation of atomic positions](#), [Template modeling score](#), [CASP](#) — CC BY-SA 4.0; RCSB [PDB-101](#) — CC BY 4.0; Baldwin & Chothia (1979), "Haemoglobin: the structural changes related to ligand binding and its allosteric mechanism," *Journal of Molecular Biology* 129:175-220 ([PubMed](#)). Structural-classification and structural-alignment sources: Murzin, Brenner, Hubbard & Chothia (1995), "SCOP: a structural classification of proteins database for the investigation of sequences and structures," *J. Mol. Biol.* 247:536-540 ([PubMed](#)); Orengo, Michie, Jones, Jones, Swindells & Thornton (1997), "CATH — a hierarchic classification of protein domain structures," *Structure* 5:1093-1108 ([PubMed](#)); Cheng, Schaeffer, Liao et al. (2014), "ECOD: an evolutionary classification of protein domains," *PLoS Computational Biology* 10:e1003926 ([PubMed](#)); Lau, Bordin, Kandathil et al. (2024), "Exploring structural diversity across the protein universe with The Encyclopedia of Domains," *Science* 386:eadq4946 ([PubMed](#)); Holm & Sander (1993), "Protein structure comparison by alignment of distance matrices" (DALI), *J. Mol. Biol.* 233:123-138 ([PubMed](#)); Zhang & Skolnick (2005), "TM-align: a protein structure alignment algorithm based on the TM-score," *Nucleic Acids Research* 33:2302-2309 ([PubMed](#)); van Kempen, Kim, Tumescheit, Mirdita, Lee, Gilchrist, Söding & Steinegger (2024), "Fast and accurate protein structure search with Foldseek," *Nature Biotechnology* 42:243-246 ([PubMed](#)) — CC BY 4.0 (Foldseek paper). The two real crystal structures used below and in the notebook, PDB entries [2HHB](#) (deoxy hemoglobin) and [1HHO](#) (oxy hemoglobin), were fetched live from the PDB archive — structure coordinate data is [CC0 1.0](#) (public domain), as is a third real structure used below, [1EMA](#) (green fluorescent protein), as are [3RGK](#) (human myoglobin, a K45R variant), [1OJ6](#) (human neuroglobin) and [1IDR](#) (*M. tuberculosis* truncated hemoglobin N). All secondary-structure counts, distances, distance maps, alignments, TM-scores and rendered figures below were computed fresh for this page and its notebook using [Biopython](#), a locally-run [PyMOL](#) (open-source), TM-align (through [tmttools](#)), and a locally-run [Foldseek](#) (open-source), not taken from the lecture's own numbers or images; CATH and Pfam classifications were fetched live from the PDB and InterPro APIs.

Tense and relaxed

Haemoglobin walks into a bar. The bartender says: “You look tense.”

Haemoglobin: “Give me some oxygen and I’ll relax.”

## 6.1 Learning goals

By the end of this session you should be able to:

- describe the four levels of protein structure, using haemoglobin, and name the computational task that belongs to each;
- represent a structure as coordinates, as a distance map and as a contact map, and explain what each makes easy to see;
- explain the difference between a sequence alignment and a structure alignment, and why structure is often more conserved than sequence;
- interpret RMSD and TM-score, and explain at a conceptual level how DALI and Foldseek compare structures;
- explain why a single PDB entry is not necessarily “the” structure of a protein;
- state secondary-structure prediction as a supervised learning problem: what goes in, what comes out.

The question for this session is: **what information about a protein is contained in its sequence, what becomes visible only in 3D, and how can computers connect the two?** Sessions 3–5 worked with sequences, retrieving them, aligning them and searching for families. In Session 3 you also found structures, but only as *database records* (a PDB entry, an AlphaFold model). In this session the structure itself becomes the data. Its coordinates can be turned into distances, secondary structure, domains and interfaces, and compared with other structures. The course’s globin thread continues as well. In Session 3 you looked up *HBB*’s identifiers, in Session 4 you aligned globin sequences and in Session 5 you searched for the globin family, and now you compare globin structures.

## 6.2 Sequence becomes structure

### 6.2.1 Four levels, four computational tasks

A protein starts as a **polypeptide chain**, a linear polymer of amino acids, which folds into a 3D shape. Structural biology describes the result at four levels, and in this course each level comes with its own computational problem:

Table 6.1: The four levels of protein structure and where they appear in the course.

| Level             | What it is                                                    | Computational task               | Where in the course      |
|-------------------|---------------------------------------------------------------|----------------------------------|--------------------------|
| <b>Primary</b>    | the amino-acid sequence                                       | alignment, BLAST, profiles       | Sessions 3-5             |
| <b>Secondary</b>  | local backbone shapes: $\alpha$ -helix, $\beta$ -strand, coil | prediction from sequence         | Sessions 7-11            |
| <b>Tertiary</b>   | the 3D fold of one chain                                      | structure comparison, prediction | this session; Session 12 |
| <b>Quaternary</b> | several chains packed together                                | interfaces, complexes            | this session; Session 12 |

- An  **$\alpha$ -helix** is a coil held together by hydrogen bonds between the backbone N-H and C=O groups of residues four apart. A  **$\beta$ -sheet** is built from extended  **$\beta$ -strands** that are hydrogen-bonded edge to edge. Everything else is called **coil**. Programs such as DSSP assign these labels from the coordinates of a solved structure.
- The tertiary fold is held together mainly by the **hydrophobic effect** (non-polar side chains bury themselves away from water), together with hydrogen bonds, electrostatic interactions and van der Waals contacts.
- Many proteins work as a single chain, while others assemble into complexes of two to hundreds of chains.

### 6.2.2 Haemoglobin: all four levels in one protein

Haemoglobin is a **heterotetramer** of two  $\alpha$ -globin chains (141 residues) and two  $\beta$ -globin chains (146 residues;  $\beta$ -globin is *HBB*, this course's running example since Session 3). Each chain folds on its own and carries an oxygen-binding haem group. Using PyMOL's secondary-structure algorithm (dss), the notebook computes that one  $\beta$  chain is **118 of 146 residues (80.8%)  $\alpha$ -helix and 0%  $\beta$ -strand**. The PDB entry's own annotation, which Session 7 uses, gives 117 helical residues, because the two methods differ slightly in where exactly a helix ends.

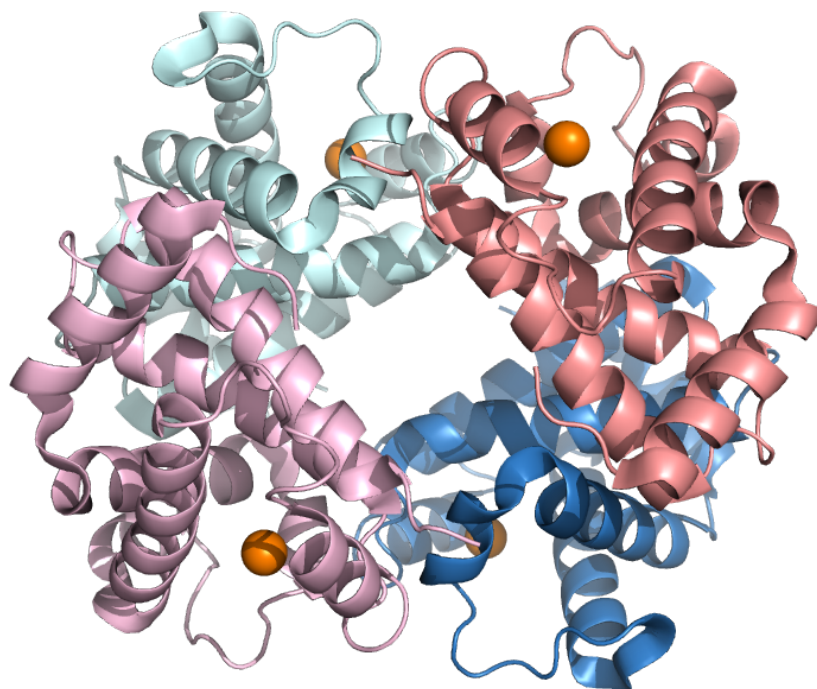


Figure 6.1: PDB entry 2HHB, human deoxyhaemoglobin: two  $\alpha$  chains (blue/cyan), two  $\beta$  chains (salmon/pink), four haem iron atoms (orange spheres). Rendered from the real structure with PyMOL.

### 6.2.3 Does the sequence determine the structure?

Whether a computer can predict a structure at all depends on how much of it the sequence determines. In the 1960s Christian Anfinsen showed that ribonuclease, unfolded in the test tube, refolds into its active shape on its own once the denaturant is removed. For many small globular proteins, then, the sequence contains enough information to specify the native fold under the right conditions, and this principle is what makes structure prediction possible. It is not a law for every protein, however. Some proteins need chaperones to fold efficiently, many need a ligand, cofactor or binding partner, some switch between alternative states (haemoglobin itself, below), and **intrinsically disordered** regions have no single fold.

Two kinds of interaction build a fold. **Local** interactions, between residues close in sequence, form secondary structure. The hydrogen bonds of an  $\alpha$ -helix, for example, connect residues  $i$  and  $i + 4$ . **Long-range** interactions, between residues far apart in sequence but close in space, pack the helices and strands into a tertiary structure. Local structure can therefore be partly guessed from a short window of sequence (Session 7), whereas long-range structure cannot, which is why tertiary-structure prediction is so much harder.

### 6.2.4 A structure as a matrix: distance and contact maps

A structure file is a list of atomic coordinates, but coordinates are not always the most useful representation. An alternative is the distance between every pair of residues, which for a chain of  $L$  residues gives an  $L \times L$  **distance map**. Thresholding the distance map gives a **contact map**, with 1 where two residues are close (here, C $\beta$  atoms within 8 Å) and 0 elsewhere. Unlike coordinates, both maps stay the same however the protein is rotated or moved, so two structures can be compared without first superimposing them.

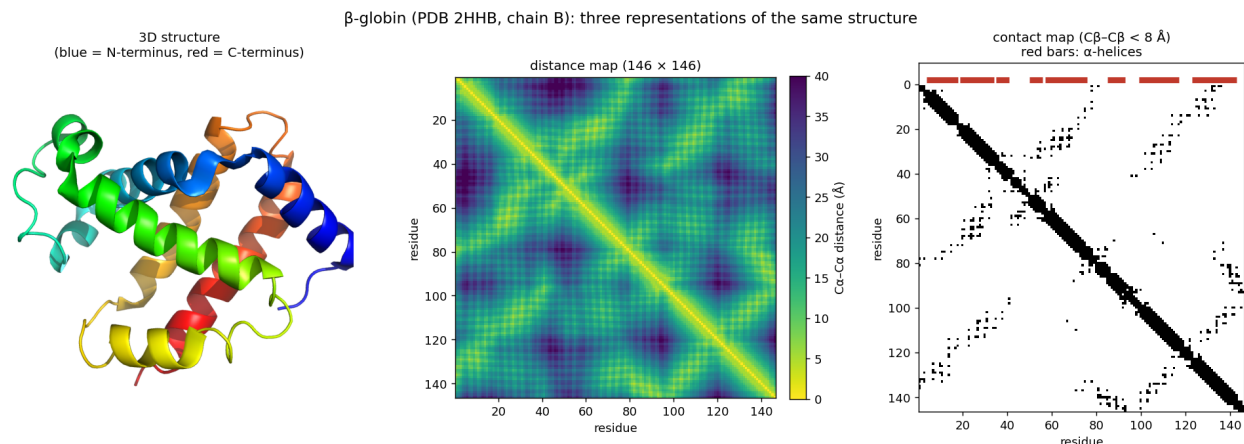


Figure 6.2: Three representations of β-globin (2HHB, chain B). Left: the 3D structure, coloured from the N-terminus (blue) to the C-terminus (red). Middle: the distance map. Right: the contact map, with the eight α-helices of the PDB file as red bars. The band along the diagonal is each helix in contact with itself (local interactions); the off-diagonal stripes are helices packed against each other (long-range interactions).

This representation returns twice later in the course. First, an  $L \times L$  map looks like an image, which is why image-processing networks (CNNs, Session 10) turned out to be good at predicting contacts. Second, coevolution, contact prediction and AlphaFold’s “pair representation” (Session 12) all work on residue-pair information like this.

#### Representations

The same protein can be represented as a sequence, a profile (Session 5), a set of coordinates, a distance or contact map, or a learned embedding (Session 11). The choice of representation determines which patterns are easy to see and, later in the course, which patterns a model can easily learn.

### 6.2.5 Looking at structures: PyMOL

To look at a structure yourself, you need a molecular viewer. [PyMOL](#) is, with ChimeraX, one of the two most widely used, and its open-source version is free. It loads a structure

directly from the PDB and shows it as **cartoon** (secondary structure as ribbons and arrows, as in the figures above), as **sticks** (every bond, useful for a side chain or ligand) or as **spheres** (every atom, useful for packing). It also measures distances and angles directly from the coordinates. This session's lab uses PyMOL hands-on, but you do not need to install the program: open-source PyMOL is also a Python library, which the lab notebook installs on Colab. You type the same commands as in PyMOL's own command line, and the notebook draws the picture. The course notebook drives PyMOL the same way, so that its figures and measurements are reproducible.

## 6.2.6 Further reading

- Wikipedia, [Protein structure](#) and [Anfinsen's dogma](#)
- RCSB [PDB-101](#) — short, illustrated explainers on structure and specific proteins, aimed at newcomers

## 6.3 Comparing proteins in 3D

### 6.3.1 Sequence alignment versus structure alignment

Session 4 defined an alignment as a hypothesis about which residues correspond. A **sequence alignment** decides this from the sequences alone, using substitution scores. A **structure alignment** decides it from the 3D coordinates, by asking which residues occupy the same position in the two folds. The notebook compares  $\beta$ -globin both ways with five proteins, ranging from a close paralog to an unrelated fold:

Table 6.2: Five proteins compared with  $\beta$ -globin, by sequence and by structure.

| Compared with $\beta$ -globin                            | Sequence identity* | Sequence-only score (shuffled max) | Residue pairs agreeing | TM-score |
|----------------------------------------------------------|--------------------|------------------------------------|------------------------|----------|
| $\alpha$ -globin (human, 2HHB)                           | 43%                | 288 (43)                           | 95%                    | 0.88     |
| myoglobin (human, 3RGK)                                  | 25%                | 111 (46)                           | 97%                    | 0.91     |
| neuroglobin (human, 1OJ6)                                | 22%                | 89 (41)                            | 66%                    | 0.82     |
| truncated haemoglobin N ( <i>M. tuberculosis</i> , 1IDR) | 16%                | 28 (49)                            | 0%                     | 0.57     |
| GFP (jellyfish, 1EMA)                                    | 6%                 | 25 (50)                            | —                      | 0.33     |

\*Identity over the structurally aligned residues. “Sequence-only score” is the best Smith-Waterman alignment with BLOSUM62, and the number in brackets is the highest score among 200 alignments against shuffled versions of the same sequence. “Residue pairs agreeing” is the fraction of structurally aligned pairs that the sequence alignment also pairs up. TM-scores are normalised by the length of  $\beta$ -globin.

The table is best read from top to bottom. For  $\alpha$ -globin and myoglobin, the sequence alignment has a strong signal and pairs up almost exactly the same residues as the structure alignment. (Session 5 showed that BLAST, with default settings, nevertheless missed myoglobin, because a heuristic search can skip an alignment that an exhaustive one finds.) For neuroglobin the sequence signal is still clear, but a third of the pairs are wrong, because the gaps are in the wrong places. For the bacterial truncated haemoglobin, the sequence alignment does no better than shuffled sequences and pairs up none of the right residues. The structure alignment nevertheless finds a shared globin fold over more than 100 residues. This is the central lesson: **structure is more conserved than sequence**. When the sequence evidence fades, structure can still reveal a relationship.

### 6.3.2 Measuring structural similarity: RMSD and TM-score

To compare structures, we need a number for how similar they are. **RMSD** (root-mean-square deviation) is the most direct measure. It is the root-mean-square distance between corresponding atoms after the two structures have been optimally superimposed. RMSD is simple, but it grows with protein length, and one badly placed loop can inflate it even when the rest of the fold matches. RMSD also depends on *which* atoms are compared. A superposition can be restricted to the residues that fit well, and a very low RMSD over a short fragment says nothing about the rest of the proteins. An RMSD should therefore always be given together with the number of aligned residues, for example “1.5 Å over 140 residues”. The **TM-score** avoids these problems. It weights each residue pair by how close it is, so that small deviations count fully and large ones hardly at all, and it normalises by protein length. It runs from 0 to 1 (identical). Unrelated proteins typically score around 0.2–0.3, and scores above about 0.5 usually mean the same overall fold. This is a rule of thumb, not a law. The truncated haemoglobin above (0.57) and GFP (0.33) are clear cases, but scores in between need other evidence.

### 6.3.3 DALI: comparing distance maps

If two proteins have the same fold, their distance maps should look alike, even when their sequences do not. **DALI** (Holm & Sander, 1993) uses this idea to compare structures. Rather than superimposing coordinates, it looks for residue correspondences that make the patterns of intramolecular distances agree.

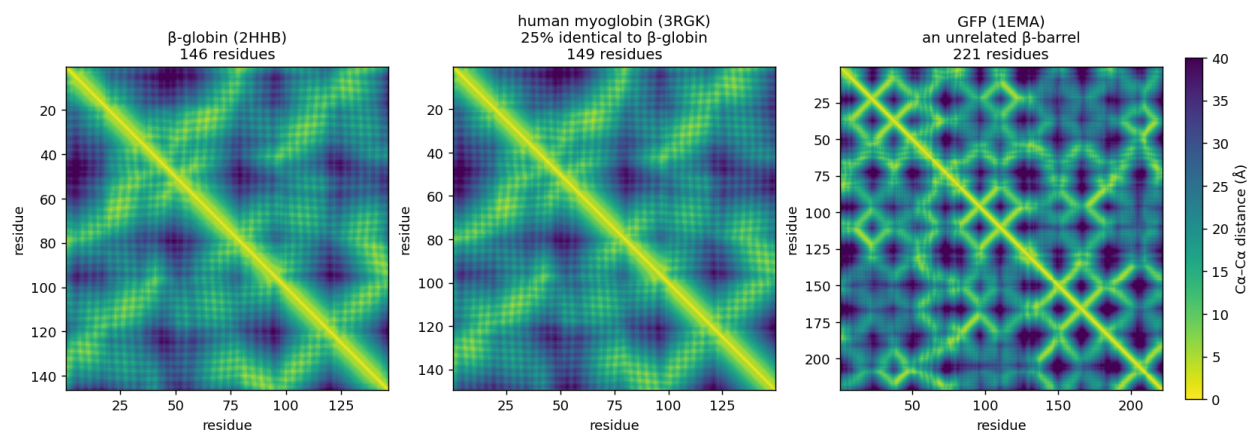


Figure 6.3: Distance maps of  $\beta$ -globin, human myoglobin and GFP. The first two are almost indistinguishable although only a quarter of their residues are identical; GFP’s  $\beta$ -barrel gives a completely different pattern.

The analogy with sequence alignment is direct:

- Sequence alignment compares residues along two sequences.
- DALI compares patterns of residue-residue geometry.

DALI and **TM-align** (Zhang & Skolnick, 2005), which finds the superposition that maximises the TM-score, are accurate and still used for careful pairwise comparisons. They are, however, far too slow to search hundreds of millions of structures.

#### 6.3.4 Foldseek: structure search at sequence-search speed

Session 5 ended with MMseqs2, which searches billions of sequences. **Foldseek** (van Kempen, Steinegger et al., 2024), from the same group, does the same for structures:

Table 6.3: Searching by sequence and by structure.

|                          | Search tool    | Searches with           |
|--------------------------|----------------|-------------------------|
| Sequence (Session 5)     | BLAST, MMseqs2 | the amino-acid sequence |
| Structure (this session) | Foldseek       | a “structural sequence” |

The key idea of Foldseek is a change of representation. For each residue, Foldseek looks at the local 3D environment (the geometry between that residue and its closest neighbour in space) and assigns one of 20 discrete structural states, the **3Di alphabet**. A structure thus becomes a string of 3Di letters, one per residue. Two structures can then be compared with fast sequence-alignment machinery (the same prefilter-and-align approach as MMseqs2) instead of expensive geometric superposition:

### 3D coordinates

- local geometry around each residue
- one 3Di state per residue
- a structural "sequence"
- fast alignment and database search

This makes it possible to search the hundreds of millions of structures in the AlphaFold Database and the PDB in seconds to minutes. Foldseek is tens of thousands of times faster than TM-align at comparable sensitivity.

#### Machine-learning preview: a learned representation

Nobody designed the 20 states of Foldseek's 3Di alphabet by hand. Instead, a machine-learning model (a variational autoencoder) was trained on pairs of structurally aligned residues to find a compact set of recurring 3D environments that are conserved between related structures. You do not need to know how such a model works. The point is that a *useful representation can itself be learned from data*. In Session 5, humans chose both the representation (amino acids) and the scoring scheme (BLOSUM62), whereas in Foldseek, machine learning helped choose the representation. Sessions 7-12 build up the machinery behind models that learn increasingly powerful representations.

**A worked comparison.** We ran Foldseek locally on three structures: the two haemoglobin states used on this page (2HHB, deoxy; 1HH0, oxy) and green fluorescent protein (1EMA). Matching  $\alpha$ -to- $\alpha$  or  $\beta$ -to- $\beta$  chains between 2HHB and 1HH0, which are the *same* protein in two states, gives a TM-score of **0.98** and an RMSD of **0.7 Å**, about as similar as two independently solved structures get.  $\alpha$  against  $\beta$  globin (43% identical) still gives a TM-score of **0.90-0.91** (RMSD 1.7-1.8 Å). Searching 2HHB against GFP, in contrast, returns **no hits**, even with a loosened significance threshold.

### 6.3.5 Further reading

- Wikipedia, [Root-mean-square deviation of atomic positions](#) and [Template modeling score](#)
- Foldseek, [GitHub](#) and [web server](#) — try it on any PDB entry or AlphaFold model, no installation needed
- DALI server, [ekhidna2.biocenter.helsinki.fi/dali](http://ekhidna2.biocenter.helsinki.fi/dali)

## 6.4 Domains, complexes and conformational states

### 6.4.1 Structural domains

A **domain** is a compact, semi-independent unit of a structure, which is often able to fold on its own and is reused in many different proteins. Sessions 3 and 5 met domains from the *sequence* side, as Pfam families built from profile HMMs. Structural

classifications instead group domains by their 3D shape, which, as the table above showed, can reveal relationships that sequence misses.

**CATH** (Orengo et al., 1997) classifies domains in four levels: **Class** (mainly  $\alpha$ , mainly  $\beta$ ,  $\alpha/\beta$ , few secondary structures), **Architecture** (the overall arrangement of secondary-structure elements), **Topology** (how they are connected, that is, the fold), and **Homologous superfamily** (domains with evidence of common ancestry).  $\beta$ -globin's domain is 1.10.490.10: Mainly Alpha  $\rightarrow$  Orthogonal Bundle  $\rightarrow$  Globin-like  $\rightarrow$  Globins. So are myoglobin, neuroglobin and the *M. tuberculosis* truncated haemoglobin from the table above. Pfam, which works from sequence, needs a separate family for the truncated haemoglobin (PF01152, "Bacterial-like globin", not PF00042, "Globin") and connects the two only at the level of a *clan*. GFP, in contrast, is in CATH class 2 (Mainly Beta), Beta Barrel.

Three other classifications exist, each with a different philosophy: **SCOP** (Murzin et al., 1995; expert manual curation), **ECOD** (Cheng et al., 2014; organised by evolutionary relationship), and **TED** (Lau, Bordin et al., 2024; domains parsed automatically from the entire AlphaFold Database, hundreds of millions of predicted structures). They do not always agree on where one domain ends or on how distant superfamilies group together. Knowing that several classifications exist, and why, therefore matters more than picking one.

#### 6.4.2 Quaternary structure, interfaces and ligands

Beyond single chains, a complex is held together by **interfaces**, patches of surface where two chains touch. Interfaces are usually more hydrophobic and more conserved than the rest of the surface. Many proteins also bind **ligands**, small molecules or ions such as haemoglobin's haem groups. Interfaces and ligand-binding sites are where much of a protein's function happens, and predicting them is one of the open problems of Session 12.

#### 6.4.3 One protein, more than one structure: haemoglobin T and R

A PDB entry is one snapshot of one state. Haemoglobin has two classic states, deoxy (**T**, tense; 2HHB) and oxy (**R**, relaxed; 1HHO). Each chain hardly changes between them (TM-score 0.98), but the two  $\alpha/\beta$  pairs rotate relative to each other by about 13–15° (Baldwin & Chothia, 1979; the third discussion problem measures it). The notebook measures one consequence: the distance between the iron atoms of the two  $\beta$ -chain haems is about 5 Å shorter in the oxy state. This quaternary change is how haemoglobin binds oxygen cooperatively, since binding at one subunit makes binding at the others easier.

It follows that **a PDB structure is not necessarily "the" structure of a protein**. Proteins can occupy different conformational states depending on ligands, partners and conditions. This matters whenever a predicted model is compared with "the" experimental structure, and it is one of the limitations that Session 12 returns to, because AlphaFold usually predicts one state.

#### 6.4.4 Further reading

- CATH, [cathdb.info](http://cathdb.info); SCOP, [ebi.ac.uk/pdbe/scop](http://ebi.ac.uk/pdbe/scop); ECOD, [prodata.swmed.edu/ecod](http://prodata.swmed.edu/ecod); TED, [ted.cathdb.info](http://ted.cathdb.info)
- RCSB PDB-101, [Hemoglobin](#) — the T and R states, illustrated

### 6.5 From structure back to sequence prediction

#### 6.5.1 Secondary structure as a prediction target

Given only the sequence, could a computer predict which residues become helix, strand or coil? Stated as a learning problem, the task is:

input: the sequence (or its profile), one window per residue  
target: H (helix), E (strand) or C (coil) for every residue

The training labels come from solved structures, assigned by DSSP as in the notebook. What information could help the prediction? The residue itself is one source, because some amino acids favour helices, others strands, and proline and glycine break both. Its neighbours in the sequence are another. A third, from Session 5, is how that position varies across homologous sequences, that is, a profile. Sessions 7–11 develop the machinery for learning such mappings, and the course returns to this problem in Sessions 7, 9 and 10 and in the capstone project.

#### 6.5.2 Predicting the whole fold

Predicting the full 3D structure has been one of biology's hardest problems for decades. Since 1994 the **CASP** competition (Critical Assessment of Structure Prediction) has tested methods blindly, on structures that are solved but not yet published. The classical strategies each build on something you have already met:

- **Physics** (*de novo* or *ab initio* modelling) searches for the lowest-energy arrangement of the chain, using physical interactions. It is rigorous in principle, but so expensive that it long worked only for small proteins. **Rosetta** (Baker lab) assembles structures from short fragments of known structures, and it grew into a tool for protein *design*.
- **Comparative (homology) modelling** finds a related protein of known structure (Sessions 4–5) and builds the model on it as a template. It is reliable when a template exists, but useless for a new fold. The table above shows why it works: related sequences share their structure.
- **Contact prediction** uses the fact that residues that touch tend to mutate together. Large alignments (Session 5) therefore reveal which pairs are in contact and fill in a contact map like the one above.

- **Deep learning** arrived with AlphaFold2 (2020), which combines learned representations, evolutionary information and geometry, and was the breakthrough in CASP14. David Baker, Demis Hassabis and John Jumper shared the 2024 Nobel Prize in Chemistry for protein design and structure prediction. Session 12 covers AlphaFold, once Sessions 7–11 have built up the neural-network machinery.

In terms of representations, the course's path from here is:

sequence

- profile / evolutionary information (Session 5)
- learned representation (Sessions 7–11)
- local structure, contacts, distances (Sessions 7, 10, 12)
- 3D structure (Session 12)

### 6.5.3 Further reading

- Wikipedia, [Homology modeling](#) and [CASP](#)
- Rosetta Commons, [rosettacommons.org](https://rosettacommons.org)

## 6.6 The course notebook

`notebooks/session06-protein-structure-in-code.ipynb` fetches the two haemoglobin structures (deoxy 2HHB, oxy 1HH0) and makes each level of structure concrete. It reads the chains and residues from the file, computes the secondary structure with PyMOL (and compares it with the PDB annotation that Session 7 uses), and renders an image of the tetramer. It also draws  $\beta$ -globin's distance and contact maps, builds the sequence-versus-structure alignment table above, and measures the iron-to-iron distance that changes by about 5 Å between the two states. Open it in [Google Colab](#) via the badge at the top of the notebook page to edit and run it yourself.

## 6.7 Check your understanding

The questions below give immediate feedback and cover the sections above. They are in the same style as the course's live in-class peer-instruction quizzes, and the full question bank is in `quizzes/session06.md` in the repository.

*The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)*

## 6.8 The lab

The whole lab runs in [notebooks/session06-lab.ipynb](#) on Colab, in the browser or from VS Code; nothing is installed on your own computer. **It starts with PyMOL**, run as a Python library inside the notebook, on three structures beyond haemoglobin. An HIV envelope glycoprotein trimer with an antibody (7LXM) is used for representations, chains and structural alignment (cealign). A chitinase with bound metal ions and an inhibitor (8C6Z) is used for ligands and charged residues. Human neuroglobin (10J6) is used for the central comparison, which you run with TM-align:

|                                                     |                    |
|-----------------------------------------------------|--------------------|
| same protein, experimental vs. AlphaFold prediction | → nearly identical |
| a different globin (hemoglobin)                     | → same fold        |
| GFP                                                 | → unrelated fold   |

Here AlphaFold DB simply supplies a structure to inspect, and Session 12 asks how that prediction is produced and how to decide which parts of it to trust. You then search the whole PDB with neuroglobin through the Foldseek web server's API, and the lab ends with a structural domain lookup in CATH. An optional tip shows how to view structures directly in VS Code with the Protein Viewer extension.

### 6.8.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## 6.9 What's next

This session gave you structural properties that are worth predicting from sequence. Session 7 starts building the machinery to predict them. It returns to the weighted sum that a PSSM computes,  $S(x) = \sum_i w[i, x_i]$ , and formalises it as a **perceptron** whose weights are *learned* from labelled examples. It then applies the perceptron to the secondary structure you met in this session, through Rost & Sander's PHD method, and shows why the helix you saw in PyMOL can be hard to predict from a window of residues.

## 7 Session 7: Linear Models, Gradient Descent, the XOR Problem & Secondary Structure Prediction

### **i** Note

**Sources for this page:** [d2l.ai](https://d2l.ai), *Dive into Deep Learning*, [Linear Neural Networks for Regression](#) — CC BY-SA 4.0; Wikipedia, [Perceptron](#), [Perceptrons \(book\)](#), [Gradient descent](#), [Linear separability](#), [AI winter](#), [Backpropagation](#), [Protein secondary structure](#), [Alpha helix](#), [Beta sheet](#), and [Protein structure prediction](#) — CC BY-SA 4.0; Bulyk, Johnson & Church (2002), *Nucleic Acids Research* 30(5):1255-1261, [PubMed](#); Chou & Fasman (1974), *Biochemistry* 13(2):222-245, [PubMed](#); Garnier, Osguthorpe & Robson (1978), *Journal of Molecular Biology* 120(1):97-120, [PubMed](#); Rost & Sander (1993), “Prediction of protein secondary structure at better than 70% accuracy,” *Journal of Molecular Biology* 232:584-599. The brain/perceptron figure is Rosenblatt, F., “The Design of an Intelligent Automaton,” *Research Reviews*, Office of Naval Research, October 1958, pp. 5-13 — public domain (US federal government work), via [Wikimedia Commons](#). The real secondary-structure counts for *HBB* below were fetched live from EBI’s [PDB API](#) — CC BY 4.0 — see the notebook. The by-hand AND/OR/NAND perceptron weights, the closed-form vs. iterative linear-regression comparison, and the “solve XOR by engineering a feature” example below are mined for ideas from Magnus Ekman’s *Learning Deep Learning* (slide decks Chapter-1, Chapter-2, Appendix-A in slides/) — a copyrighted, not openly-licensed textbook (the very one this course rebuild is replacing), credited here as an internal source the way Petras Kundrotas’s own lecture decks are on other days, not linked as further reading. Every number below was independently recomputed and verified in code, not transcribed. Text below is written from scratch, not copied, following these ideas.

### A fast learner

**Interviewer:** What’s your biggest strength?

**Me:** I’m a fast learner

**Interviewer:** What’s  $11 * 11$

**Me:** 65

**Interviewer:** Not even close. It’s 121

**Me:** It’s 121

Session 6 gave us structural properties that we would like to predict from sequence,

with secondary structure first among them. This session starts building the mathematical machinery for doing that.

## 7.1 Learning goals

By the end of this session you should be able to:

- explain that a profile score and a perceptron share the same mathematical core, a weighted sum (dot product), and how they differ;
- explain how a model's weights can be *learned* from labelled examples by gradient descent, rather than derived from alignment statistics;
- explain, with a real biological example, why some patterns cannot be captured by any linear model, and why a hidden layer only helps if its activation function is nonlinear;
- describe how Rost & Sander's PHD method (1993) combined profile input from multiple alignments with a neural network to predict secondary structure better than earlier methods;
- explain why a more powerful model cannot compensate for an input representation that lacks the relevant information.

## 7.2 Two pictures to keep in mind

This session uses many names: profile score (PSSM), perceptron, linear and logistic regression, sigmoid, tanh, hidden layer, multilayer perceptron and PHD. Behind them there are only two pictures, and together these two pictures explain the whole idea of the lecture.

### One weighted sum: one straight boundary



PSSM score (no activation) · perceptron (step) · logistic regression (sigmoid) → AND, OR, NAND; not XOR

### Add a hidden layer: several boundaries, combined



multilayer perceptron (MLP), hidden units with tanh or ReLU, e.g. PHD → XOR

*Without the nonlinear activation, the two weighted sums collapse into one, and XOR is impossible again.*

Figure 7.1: Top: one weighted sum followed by an activation. A PSSM score, a perceptron and logistic regression are all this picture, and all of them draw one straight boundary. Bottom: a hidden layer of weighted sums with a nonlinear activation, followed by one more weighted sum. This is a multilayer perceptron (MLP), such as PHD, and it can combine several boundaries.

Keep them in mind as you read. The goal of the session is to understand what changes between the two pictures, and why that change makes XOR possible.

## 7.3 A profile score and a perceptron: the same weighted sum

**You actually already know the central computation of a neural network.** Session 5 introduced the profile score, which checks how well a sequence matches a known family. It multiplies each position's observed value by a position-specific weight and adds up the results. In Session 1's terms, this is a dot product, something you learned on the first day. A **perceptron** has exactly the same mathematical core, because it also computes a weighted sum  $z$ :

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

The extra term  $b$ , a **bias**, shifts the whole sum by a constant amount, so that the model's decision threshold can sit somewhere other than zero. The PSSM and the perceptron share this weighted sum, but they differ in what they do with it. A PSSM reports the sum itself as a *score*. A perceptron instead passes the sum through an **activation function**, originally a threshold (output 1 if  $z > 0$ , else 0), to make a *classification*. Keeping the two terms apart helps later on. "Profile score" means a linear score, and "perceptron" means a (thresholded) classifier that adds an activation function to the same weighted sum.

**Where the name comes from.** Frank Rosenblatt introduced the perceptron in 1958, first as a simulation on an IBM 704 computer, and the Mark I Perceptron machine followed in 1960. He modelled it on a simplified neuron. Each synapse weights its input, the cell sums the weighted inputs, and the cell fires once the sum crosses a threshold. The weights  $w_i$  therefore stand for connection strengths, and  $b$  for how easily the cell fires. His own diagram set the circuit of the brain next to the perceptron built to mirror it:

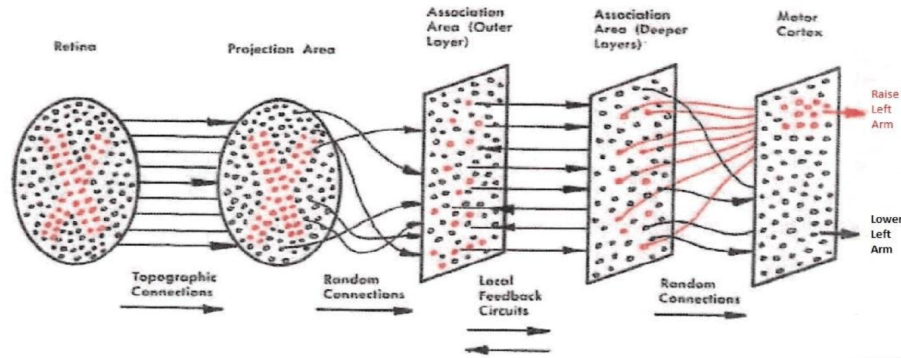


**FIG. 1 — Organization of a biological brain.** (Red areas indicate active cells, responding to the letter X.)

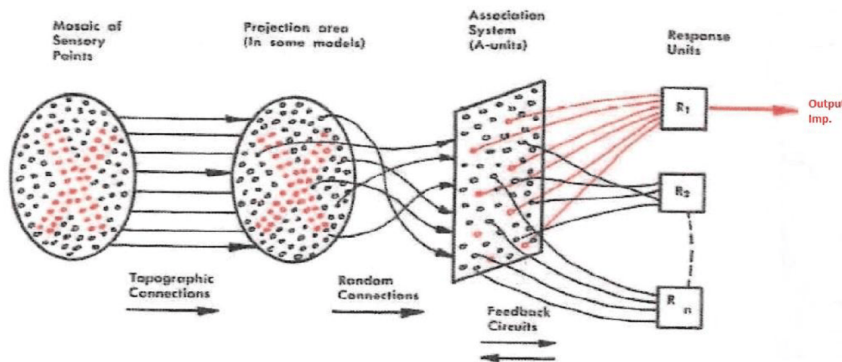


**FIG. 2 — Organization of a perceptron.**

Figure 7.2: Rosenblatt's own 1958 diagram: a biological brain's circuit (top) next to the perceptron he built to mirror it (bottom) — retina/sensory input, an association layer, and motor/response output on both sides. Public domain (US federal government work) — Rosenblatt, F., "The Design of an Intelligent Automaton," *Research Reviews*, Office of Naval Research, October 1958, pp. 5-13.

Geometrically,  $z = 0$  is a line in two dimensions (a plane, or hyperplane, in higher dimensions), and this line is the model's **decision boundary**. On one side of it the output is 1, and on the other side it is 0.

Solving  $w_1x_1 + w_2x_2 + b = 0$  for  $x_2$  gives  $x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$ . This is the familiar  $y = mx + c$  form of a line, with the bias  $b$  playing the role of the intercept  $c$  (scaled by  $-1/w_2$ ). Changing  $b$  therefore slides the whole line sideways without changing its

slope. The intercept  $b_0$  plays the same role in ordinary linear regression, as the next section shows.

### 7.3.1 Further reading

- Wikipedia, [Perceptron](#) — other activation functions and the original biological motivation, beyond the general-purpose version above

### 7.3.2 Further watching

*But what is a neural network?* (3Blue1Brown, Deep Learning ch. 1), animating what a weighted sum + activation is doing:

Video: [But what is a neural network?](#), 3Blue1Brown

## 7.4 Learning the weights from examples

So far the weights have simply been given. A profile's weights come from counting: we look at a multiple alignment, count how often each residue appears at each position, and turn those counts into log-odds scores. This works well when we already have a trustworthy alignment to count from. As Session 5 showed, small alignments need some extra tricks (pseudocounts), but that is still counting, only with some modifications.

The alternative to **counting**, and the one the rest of this course builds on, is to **learn** the weights directly from labelled examples with an optimisation method, normally gradient descent as described in Session 1. We often start with arbitrary (random) weights and measure how wrong the model's current predictions are with a *loss function*. We then compute the gradient of that loss with respect to each weight, and nudge every weight a small step in the direction that reduces the loss. This is repeated many times. No alignment and no hand-counting are needed, only examples and many small corrections. This is the foundation of all machine learning: we do not tell the computer what to do, but let it **learn** what to do.

In this session's lab, the code works directly with raw tensors and automatic differentiation (Session 2). It deliberately avoids the higher-level `nn.Module` and training-loop tools that Session 8 introduces, because the point here is to see every step of the mechanics rather than to use a convenient shortcut around them.

**Worked example.** Take a perceptron with one input and no bias yet ( $z = wx$ ), an initial weight  $w_0 = 0$ , a training example  $(x, y) = (2, 6)$ , a squared-error loss  $L(w) = (wx - y)^2$ , and a learning rate  $\eta = 0.05$ . Session 1's chain rule gives

$$\frac{dL}{dw} = 2(wx - y) \cdot x$$

The table shows the first iterations of gradient descent,  $w_{n+1} = w_n - \eta dL/dw$ :

Table 7.1: The first iterations of gradient descent with learning rate  $\eta = 0.05$ .

| $n$ | $w_n$ | prediction $w_n x$ | $dL/dw$                  | $w_{n+1} = w_n - 0.05 dL/dw$        |
|-----|-------|--------------------|--------------------------|-------------------------------------|
| 0   | 0.00  | 0.00               | $2(0 - 6)(2) = -24.0$    | $0.00 - 0.05 \times (-24.0) = 1.20$ |
| 1   | 1.20  | 2.40               | $2(2.4 - 6)(2) = -14.4$  | $1.20 - 0.05 \times (-14.4) = 1.92$ |
| 2   | 1.92  | 3.84               | $2(3.84 - 6)(2) = -8.64$ | $1.92 - 0.05 \times (-8.64) = 2.35$ |

The weight converges towards  $w = 3$ , where the prediction  $wx = 3 \times 2 = 6$  would exactly match the target. Just as in Session 1's one-variable example, the size of each correction ( $dL/dw$ ) shrinks by itself as the prediction gets closer, so  $\eta$  needs no manual adjustment. This holds only as long as the learning rate is not too large. With an excessive learning rate, gradient descent overshoots the minimum and can oscillate or diverge (Session 1's second discussion problem).

**Deriving vs. learning, on the simplest possible model.** Ordinary linear regression, which fits a line  $\hat{y} = b_0 + b_1 x$  to data, has a direct numerical solution. It therefore makes a useful check, because gradient descent should reach the same optimum iteratively. Take five noisy points roughly following  $y \approx 2x$ , with  $x = (1, 2, 3, 4, 5)$  and  $y = (2.1, 3.9, 6.2, 7.8, 10.1)$ . A least-squares solver gives  $b_0 = 0.05$  and  $b_1 = 1.99$  in one step, and gradient descent (learning rate  $\eta = 0.01$ , thousands of small steps) converges to the same  $b_0 = 0.050$  and  $b_1 = 1.990$ .

This mirrors the contrast between Session 5's PSSM, whose weights are *derived* directly from alignment counts, and this session's gradient descent, whose weights are *learned* iteratively. Gradient descent is what the rest of the course uses, because most models, including a perceptron with its classification loss, have no direct solution.

#### **i** Optional: the formula

The least-squares solution can be written with the **normal equations**. We stack the inputs into a matrix  $X$ , with a column of 1s for  $b_0$  next to the  $x$  values. Then

$$\hat{\beta} = (X^\top X)^{-1} X^\top \mathbf{y}$$

This formula is a way to *write* the solution, not a way to compute it, because numerical libraries (`np.linalg.lstsq`, `scikit-learn`) use more stable methods than inverting  $X^\top X$ . Moreover, when the columns of  $X$  are exactly linearly dependent,  $X^\top X$  cannot be inverted at all. This session's lab comes close to that case. The 20 amino-acid fractions of a protein sum to 1 (for 54 of its 55 proteins, since one contains unknown residues), so together with the intercept column they are almost exactly linearly dependent. A least-squares solver still finds a best fit, but the individual weights are poorly determined.

### 7.4.1 Further reading

- Wikipedia, [Gradient descent](#) — variants (stochastic gradient descent, momentum, Adam) are introduced in Session 9, where we train real networks

### 7.4.2 Further watching

[Gradient descent, how neural networks learn](#) (3Blue1Brown, Deep Learning ch. 2):

Video: [Gradient descent, how neural networks learn, 3Blue1Brown](#)

## 7.5 What one weighted sum can represent

Before asking what a perceptron cannot do, it helps to see what it can do. A single perceptron with a step activation (output 1 if  $z > 0$ , else 0) can implement basic logic gates directly. No training is required, only weights chosen by hand:

Table 7.2: Perceptron weights, chosen by hand, that implement three logic gates.

| Gate | $w_1$ | $w_2$ | $b$  |
|------|-------|-------|------|
| AND  | 1     | 1     | -1.5 |
| OR   | 1     | 1     | -0.5 |
| NAND | -1    | -1    | 1.5  |

Take AND as an example. At  $(1, 1)$ ,  $z = 1 + 1 - 1.5 = 0.5 > 0 \rightarrow 1$ , and at every other corner  $z \leq -0.5 \rightarrow 0$ , which is exactly the truth table of AND. (NAND is AND with both weights and the bias negated, which flips every output.) In the notebook, all three gates check out exactly against their truth tables.

Each gate is one straight decision boundary through the square of four inputs, with the “1” corners on one side and the “0” corners on the other. For AND, only  $(1, 1)$  lies on the positive side of the line  $x_1 + x_2 = 1.5$ . The second discussion problem asks you to find weights for OR yourself.

This is easiest to see if we draw the four inputs as dots on a plane and the decision boundary  $z = 0$  as a line between them. The step activation then simply asks on which side of the line a dot lies:

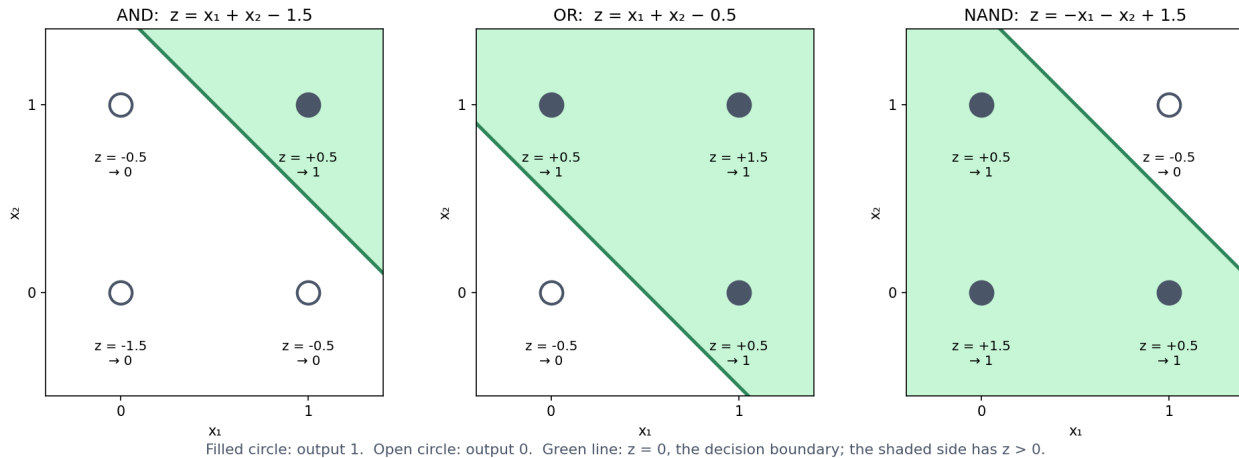


Figure 7.3: The three gates of the table, one panel each. Filled circles are inputs where the perceptron outputs 1, open circles where it outputs 0; the value of  $z$  is written under each corner. The green line is the decision boundary  $z = 0$ ; on the shaded side  $z > 0$ . For AND the line cuts off one corner, for OR it cuts off the opposite corner, and NAND is AND with the sides swapped.

In the course notebook

Section 3 of the course notebook ([open in Colab](#)) computes  $z$  and the output for all four corners of all three gates with exactly these weights, and draws this figure.

## 7.6 What one weighted sum can never represent: XOR

A model built purely from weighted sums has a real limitation, because there is a whole class of patterns it can never represent, however its weights are chosen. The classic illustration is the **XOR problem**. XOR is a function of two binary inputs that outputs 1 when its inputs *differ* and 0 when they are the *same*.

Table 7.3: The XOR function.

| $x_1$ | $x_2$ | XOR |
|-------|-------|-----|
| 0     | 0     | 0   |
| 0     | 1     | 1   |
| 1     | 0     | 1   |
| 1     | 1     | 0   |

**Before reading on, try it:** find weights  $w_1, w_2, b$  that reproduce this table with one perceptron. You will not succeed. The two “1” outputs sit on opposite corners of the input square, and no straight line can put both of them on one side while leaving both “0” outputs on the other. Section 4 of the notebook tries 226,981 combinations of  $w_1$ ,

$w_2$  and  $b$  (every value from  $-3$  to  $3$  in steps of  $0.1$ ). About 5,000 of them reproduce AND, 9,500 OR and 4,500 NAND, but not one reproduces XOR.

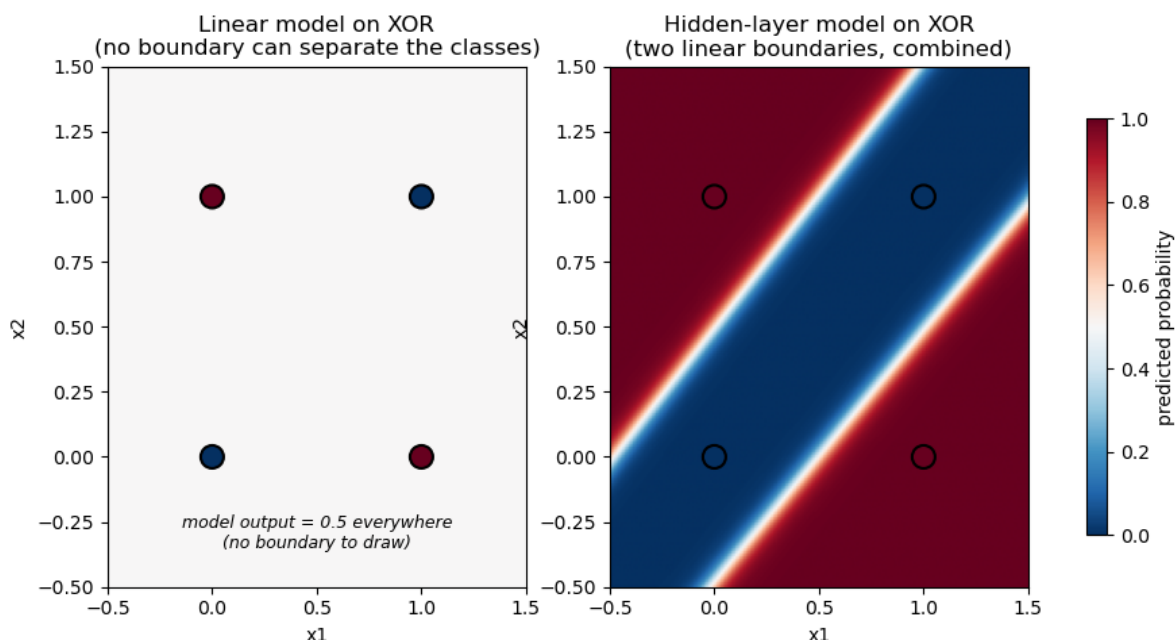


Figure 7.4: A linear model trained on XOR (left) settles on predicting 0.5 everywhere. Every straight-line boundary it could draw leaves at least one point on the wrong side, so gradient descent settles for the least-wrong compromise. A hidden-layer model on the same data (right) combines two linear boundaries into a non-linear one that separates all four points correctly. Both panels show the notebook’s own trained models, not illustrative sketches.

### **i** Optional: the algebraic proof

The geometric argument can be turned into a short proof. A perceptron with a sign-style activation outputs 1 when  $z = w_1x_1 + w_2x_2 + b > 0$  and 0 otherwise. Reproducing the XOR table therefore requires all four of the following inequalities:

$$b \leq 0, \quad w_1 + b > 0, \quad w_2 + b > 0, \quad w_1 + w_2 + b \leq 0$$

The first and last inequalities come from the two “0” rows, and the middle two from the “1” rows. Adding the second and third inequalities gives  $w_1 + w_2 + 2b > 0$ , that is,  $w_1 + w_2 + b > -b \geq 0$  (using  $b \leq 0$  from the first inequality). The fourth inequality, however, says directly that  $w_1 + w_2 + b \leq 0$ . No values of  $w_1$ ,  $w_2$  and  $b$  can satisfy both at once, so no such weights exist. This is the same fact that the geometric picture showed, derived without drawing anything.

### 7.6.1 Further reading

- Wikipedia, [Linear separability](#) — generalises the geometric argument above beyond the two-input case shown here

## 7.7 Why a hidden layer helps

One perceptron draws one line, but XOR needs two. Combining perceptrons therefore goes further than any single one can. If **OR** and **NAND** are computed separately from the same two inputs and then fed as the two inputs to an **AND** perceptron, the result reproduces **XOR** exactly:

Table 7.4: XOR built from OR, NAND and AND.

| $x_1$ | $x_2$ | OR | NAND | AND(OR, NAND) | XOR |
|-------|-------|----|------|---------------|-----|
| 0     | 0     | 0  | 1    | 0             | 0   |
| 0     | 1     | 1  | 1    | 1             | 1   |
| 1     | 0     | 1  | 1    | 1             | 1   |
| 1     | 1     | 1  | 0    | 0             | 0   |

Drawn as a network, this is a **two-layer perceptron**, with two inputs, a hidden layer of two units and one output unit. Every arrow carries a weight, and every unit adds its bias and applies the step activation:

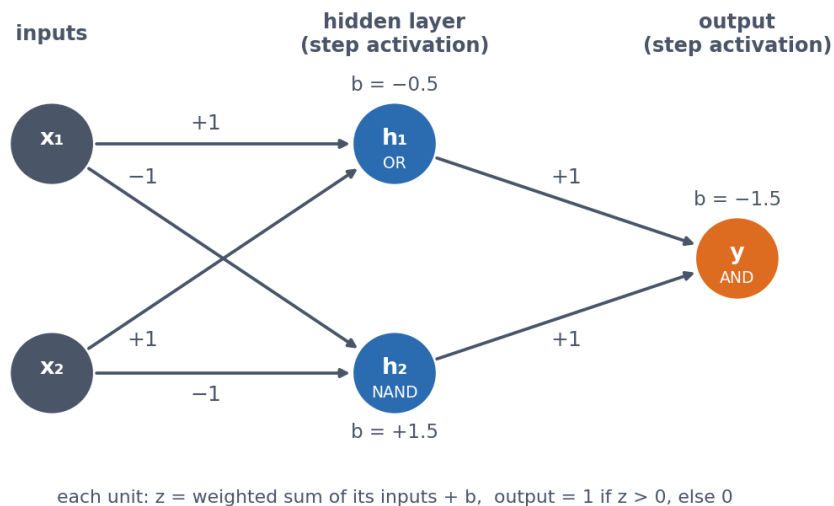


Figure 7.5: The hand-built two-layer perceptron for XOR. The hidden unit  $h_1$  computes OR (weights  $+1, +1$ , bias  $-0.5$ ),  $h_2$  computes NAND (weights  $-1, -1$ , bias  $+1.5$ ), and the output unit  $y$  computes AND of the two hidden outputs (weights  $+1, +1$ , bias  $-1.5$ ).

**Step by step.** For each input, we first compute the two hidden units, and then the output unit from their outputs:

Table 7.5: The two-layer perceptron for XOR, computed step by step.

| $x_1$ | $x_2$ | $z_{h_1} =$<br>$x_1 +$<br>$x_2 - 0.5$ | $h_1$ | $z_{h_2} =$<br>$-x_1 -$<br>$x_2 + 1.5$ | $h_2$ | $z_y =$<br>$h_1 +$<br>$h_2 - 1.5$ | $y$      |
|-------|-------|---------------------------------------|-------|----------------------------------------|-------|-----------------------------------|----------|
| 0     | 0     | -0.5                                  | 0     | +1.5                                   | 1     | -0.5                              | <b>0</b> |
| 0     | 1     | +0.5                                  | 1     | +0.5                                   | 1     | +0.5                              | <b>1</b> |
| 1     | 0     | +0.5                                  | 1     | +0.5                                   | 1     | +0.5                              | <b>1</b> |
| 1     | 1     | +1.5                                  | 1     | -0.5                                   | 0     | -0.5                              | <b>0</b> |

The last column is XOR. Take  $(1, 1)$  as an example. Both inputs are on, so the OR unit fires ( $z = 1.5$ ), but the NAND unit does not ( $z = -0.5$ ). The output unit then sees  $h_1 + h_2 = 1$ , and since  $1 - 1.5 < 0$ ,  $y = 0$ . Only the two mixed inputs switch on *both* hidden units, and only then does the AND unit fire.

The same calculation can be written in matrix form for all four inputs at once, with the inputs as the rows of  $X$  and one column of  $W_1$  per hidden unit:

$$H = \text{step}(XW_1 + \mathbf{b}_1), \quad W_1 = \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}, \quad \mathbf{b}_1 = (-0.5, 1.5)$$

$$\mathbf{y} = \text{step}(HW_2 + b_2), \quad W_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b_2 = -1.5$$

Section 5 of the notebook does exactly this in PyTorch and prints every intermediate value, and the result is the table above.

The two perceptrons computing OR and NAND form a **hidden layer**, and the third perceptron, which combines their outputs, gives the model's final output. The notebook's hidden-layer network has the same shape (two hidden units feeding one output unit), but it finds its weights by gradient descent rather than by hand. Seeing both, an exact hand-built solution here and a learned approximate one later, makes clear that gradient descent is not discovering some exotic new trick. It finds weights for a structure that could, in principle, be written down directly.

### 7.7.1 The activation must be nonlinear

The construction above works only because each unit applies a **nonlinear** activation (here a step) before its output is passed on. Without one, stacking layers gains nothing. To see why, we write the hidden layer as  $\mathbf{h} = W_1\mathbf{x} + \mathbf{b}_1$  and the output as  $z = W_2\mathbf{h} + b_2$ , with no activation. Then

$$z = W_2(W_1\mathbf{x} + \mathbf{b}_1) + b_2 = (W_2W_1)\mathbf{x} + (W_2\mathbf{b}_1 + b_2)$$

which is again a single weighted sum of the inputs, with one matrix  $W_2W_1$  and one bias. Any number of linear layers therefore collapses into one linear layer, and one linear layer cannot solve XOR. A nonlinear activation between the layers (a step, sigmoid, tanh or ReLU) is what lets the hidden units bend the decision boundary. The notebook shows this directly. The same two-hidden-unit network, trained on XOR with the activation removed, gets stuck at 0.5 for every input, just like the plain linear model.

**Keeping track of shapes.** The table follows the shapes in the notebook’s network for XOR, with the four input rows stacked into one matrix (Session 1’s shape rules):

Table 7.6: The shapes of the matrices in the notebook’s network for XOR.

|                                                  | Shape    |                                           |
|--------------------------------------------------|----------|-------------------------------------------|
| inputs $X$                                       | $(4, 2)$ | four examples, two inputs each            |
| weights $W_1$                                    | $(2, 2)$ | two inputs $\rightarrow$ two hidden units |
| hidden layer<br>$h = \tanh(XW_1 + \mathbf{b}_1)$ | $(4, 2)$ | two hidden values per example             |
| weights $W_2$                                    | $(2, 1)$ | two hidden units $\rightarrow$ one output |
| output $y = \sigma(hW_2 + b_2)$                  | $(4, 1)$ | one prediction per example                |

If you can follow these shapes, the training loop in the notebook is simply this table written in code.

#### A short history

Minsky and Papert’s book *Perceptrons* (1969) proved rigorously that single-layer perceptrons cannot solve problems like XOR. It is widely credited with contributing to a drop in neural-network research in the 1970s, the first “AI winter”. Multi-layer networks were known to be more powerful, but training them became practical only with **backpropagation** (popularised by Rumelhart, Hinton & Williams in 1986), which applies Session 1’s chain rule through the hidden layers. Session 9 covers it.

## 7.8 Biological analogues

### 7.8.1 Interactions between positions: Bulyk et al.

XOR is not just a toy puzzle about logic gates, because the same structure shows up in real sequence motifs. Bulyk, Johnson & Church (2002) used protein-binding microarrays to measure how the binding affinity of a zinc-finger transcription factor depended on the nucleotides at different positions of its binding site. They found that the positions do **not** contribute independently, since the effect of the nucleotide at one

position depends on what is at another position. This directly violates the assumption behind a profile score, namely that the contribution of each position can simply be added to the others. They measured the zinc-finger protein Zif268 (mouse Egr1) and mutants of it binding to all 64 possible central triplets of its site, and concluded that neither a consensus sequence nor a position weight matrix describes the protein's true specificity.

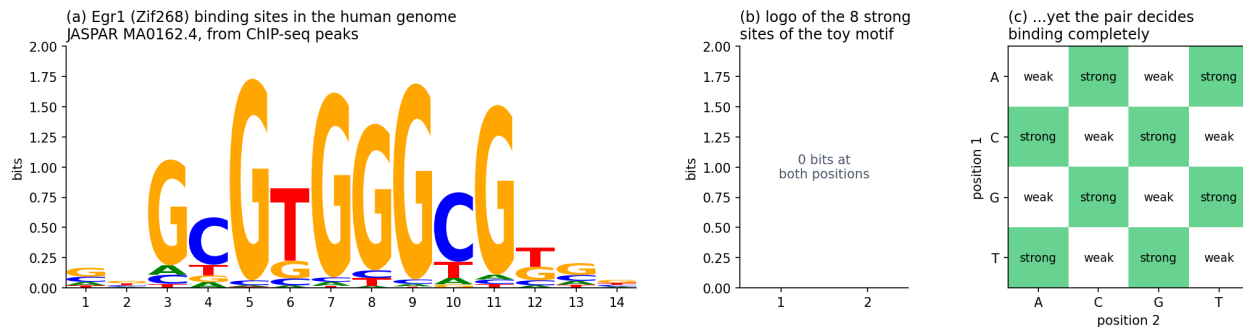


Figure 7.6: (a) What a binding-site description normally looks like: the sequence logo of Egr1 (Zif268) sites in the human genome, from ChIP-seq peaks (JASPAR MA0162.4, CC BY 4.0), with the familiar GCG TGG GCG core. A logo, like a profile, describes each position on its own. (b) The logo of the strong sites of the XOR-shaped toy motif below is completely flat: each position, taken alone, carries 0 bits. (c) Yet the *pair* of positions decides binding completely. A logo or a profile cannot show this; a hidden layer can learn it. Drawn by tools/builders/make\_session07\_figures.py.

To see why this kind of dependency is an XOR-shaped problem, consider a deliberately simplified, illustrative version. It is not the actual data from the paper, which involve more positions and continuous binding affinities, but it has the same qualitative shape. Suppose a two-position motif binds strongly only when the two positions belong to **different** purine/pyrimidine classes (one R, one Y), and weakly when they belong to the **same** class:

Table 7.7: Binding strength depending on the purine/pyrimidine classes of two positions.

| Position 1 | Position 2 | Binding |
|------------|------------|---------|
| Y          | Y          | weak    |
| Y          | R          | strong  |
| R          | Y          | strong  |
| R          | R          | weak    |

If we relabel “weak” as 0 and “strong” as 1, this is the XOR table above. Neither position predicts binding on its own, and only their combination does. This is exactly what no profile score, and no single perceptron, can represent. A hidden layer with a nonlinear activation (Section 7.7), in contrast, can represent such an interaction.

**A different fix: change the input.** A hidden layer is not the only way to solve XOR. For 0/1-valued  $x_1, x_2$ , it happens that

$$\text{XOR}(x_1, x_2) = x_1 + x_2 - 2x_1x_2$$

holds exactly (check all four rows:  $(0, 0) \rightarrow 0$ ,  $(0, 1) \rightarrow 1$ ,  $(1, 0) \rightarrow 1$ ,  $(1, 1) \rightarrow 1 + 1 - 2 = 0$ ). The right-hand side is *linear* in three inputs:  $x_1$ ,  $x_2$ , and the hand-engineered feature  $x_1x_2$ . Adding this one extra, human-chosen feature therefore turns an unsolvable problem into an exactly solvable linear regression, with weights  $(1, 1, -2)$  and bias 0. This is a real, older alternative to a hidden layer, and it works. It only works, however, if someone has noticed in advance that  $x_1x_2$  is the interaction that matters. For Bulyk et al.'s real binding-site data, or for the secondary-structure problem below, nobody knows the useful interaction terms in advance. The advantage of a hidden layer is that it *learns* useful nonlinear features like this from data, instead of requiring someone to engineer them. The engineered feature also makes a point of its own. Changing the *input representation*, not the model, turned an impossible problem into an easy one (Section 7.10).

### 7.8.2 Secondary structure prediction: PHD

Bulyk et al.'s zinc-finger microarrays make the XOR-shaped limitation concrete on a small, hand-built example. Secondary structure prediction, a much larger real problem, illustrates the same general limitation of additive, position-by-position scores. It also shows how better input and a more flexible model moved past that limitation. (XOR itself is not a historical explanation of the plateau described below, but a clean example of the kind of limitation involved.)

**What secondary structure is.** A protein's *secondary structure* is the local, repeating shape into which its backbone folds, short of the full 3D fold (its *tertiary* structure), as Session 6 showed in PyMOL. Two patterns dominate. The **alpha helix** is a right-handed coil held together by hydrogen bonds between backbone atoms four residues apart. The **beta sheet** is made of extended strands lying alongside each other and hydrogen-bonded edge to edge. Everything else is usually called **coil** (or loop/turn). Predicting secondary structure therefore means labelling every residue in a sequence with one of these three classes. This is a per-residue classification problem rather than a single yes/no like XOR, but it is built from the same kind of position-by-position scoring that this course has used throughout.

**Before neural networks.** The first methods, Chou & Fasman (1974) and GOR (1978), predate neural networks by decades. Both build each residue's score mostly by adding up contributions from single residues, and they reached roughly 50-65% three-state accuracy. The optional box below explains how they work and is useful background for the capstone project. The rest of this section explains why adding up residues from a single sequence is not enough.

## i Signal peptides: a weight matrix, then a network

Session 4 described the sequence motif of signal peptides: a positively charged n-region, a hydrophobic h-region, and small residues at positions  $-3$  and  $-1$  of the cleavage site. The history of predicting them runs through this session's two pictures (Nielsen, Tsirigos, Brunak & von Heijne 2019, [Protein J. 38:200](#), told by four of the people who made it).

**Picture one: a weight matrix.** Von Heijne's 1986 method ([Nucleic Acids Res. 14:4683](#)) is a position weight matrix with a log-odds weight for every amino acid at positions  $-13$  to  $+2$  around the cleavage site. It slides along the first 40 residues of a protein, adds up one weight per position, and puts the cleavage site where the sum is highest. The highest sum also served to tell signal peptides from other sequences. This is one weighted sum, the top picture, and like every profile score it treats each position independently.

**Picture two: networks with a hidden layer.** SignalP 1.0 (Nielsen et al. 1997, [Protein Eng. 10:1](#)) replaced the matrix with feed-forward networks on sliding windows, as PHD does for secondary structure (above). One network scores whether each position lies inside a signal peptide (the S-score), a second recognises the cleavage site itself (the C-score), and the two are combined to place the cut. With a hidden layer, a network can learn combinations, for example that a good cleavage site needs a hydrophobic stretch upstream of it. A sum of independent position scores cannot represent such a combination.

The same history also holds a warning. A network can learn combinations that are not real. The first networks for signal peptides were trained on a few dozen sequences, and one of them was built to classify every training protein correctly. It then did poorly on new proteins. Session 8 is about this problem, overfitting.

## 7.9 Optional: Chou-Fasman, GOR and the accuracy curve

*This is useful background for the [capstone project](#), where you build your own secondary-structure predictor. Chou-Fasman and GOR are simple baselines to compare it with.*

The first real attempts predate neural networks by decades. **Chou & Fasman (1974)** derived empirical propensities for helix, sheet and turn from the (then small) set of known structures. Alanine, glutamate, leucine and methionine, for example, are strong helix-formers, while proline and glycine are helix-breakers. The method combined these propensities with rules. It first finds short *nucleation* regions of high-propensity residues and then *extends* them along the chain until breakers stop them. It is therefore a propensity- and rule-based method, not a simple one-residue-at-a-time classifier, and it reached roughly **50-60%** three-state accuracy. **Garnier, Osguthorpe & Robson (1978)**, the **GOR** method, scored each residue with an information-theoretic sum of contributions from a whole *window* of neighbours. It reached roughly **65%** and was better at helices than at sheets. Neither method is literally a perceptron, but both build their score mostly by adding up per-residue contributions, and this is where the limitation lies.

GOR in particular is a linear model in the sense of Sections [7.3-7.5](#). To label a

residue, it looks at a window of 17 residues centred on it. For each offset in the window, it looks up in a table how much the amino acid at that offset favours helix, strand or coil, and it adds up these values. Each state thus gets a score, and the largest score wins. This amounts to three position-specific scoring matrices, one per state, read off the same window. GOR is therefore three weighted sums, with one straight boundary between each pair of states.

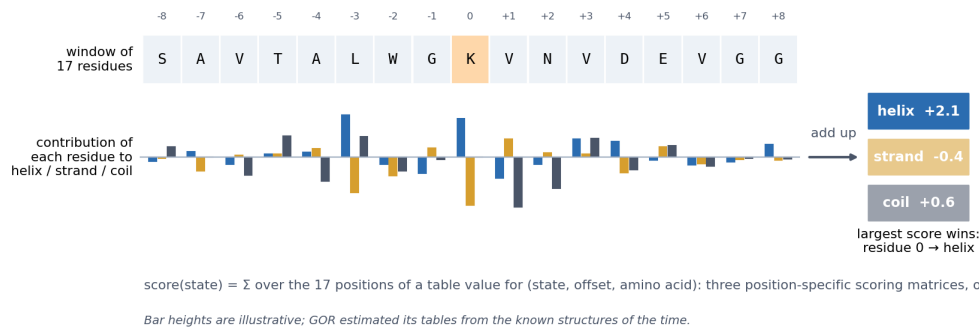


Figure 7.7: GOR as a linear model. For the middle residue of a 17-residue window (here from HBB), every residue in the window contributes a value to the helix, strand and coil scores, depending on its amino acid and its offset from the middle. The contributions are added, and the state with the largest total is predicted. The bar heights are illustrative; GOR estimated its tables from the protein structures known at the time (originally for four states, with turns separate).

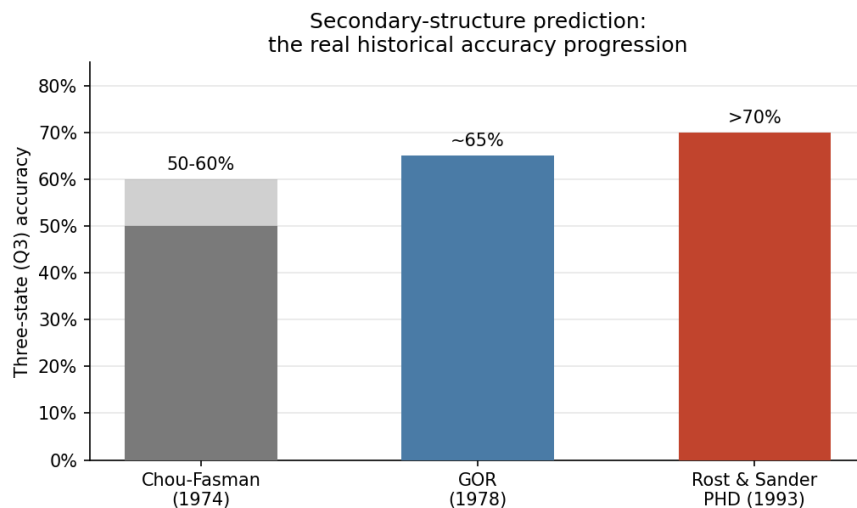


Figure 7.8: The real historical accuracy progression: Chou-Fasman’s propensities and nucleation/extension rules, GOR’s neighbour-window information-theoretic score, and Rost & Sander’s profile-fed neural network. Figures are the broadly-cited historical consensus (Wikipedia, “Protein structure prediction”) — the three original papers used different, non-identical benchmark sets, not one shared modern test set.

**Why a single sequence is not enough.** A window of neighbours is still only local information, and local sequence does not determine local structure. Kabsch & Sander (1984) searched the 62 protein structures then known and found identical five-residue stretches that were part of a helix in one protein and of a strand in another. With today’s PDB, the effect extends to stretches of ten residues. These **chameleon sequences** (Li et al. 2015, ChSeq database) adopt different secondary structures depending on the protein around them.

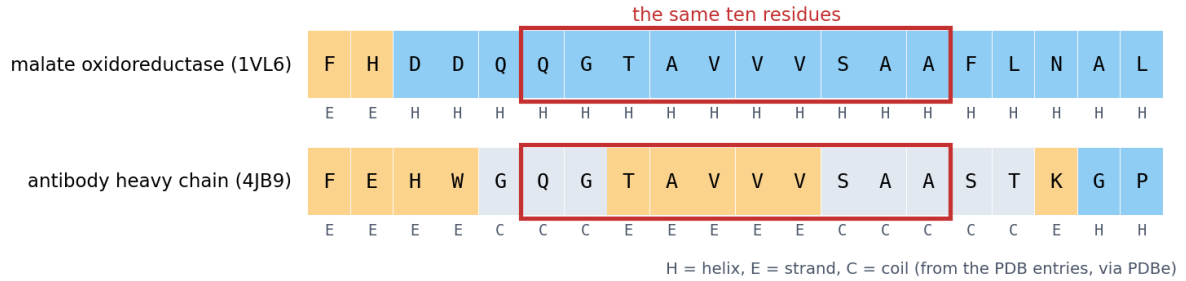


Figure 7.9: The same ten residues, QGTAVVVSAA, in two unrelated proteins: part of a long helix in a malate oxidoreductase from *Thermotoga maritima* (PDB 1VL6) and a strand followed by coil in a human antibody heavy chain (PDB 4JB9). Positions are counted along each chain’s sequence (1VL6: 164-183, 4JB9: 118-137); in the PDB files’ own numbering these are residues 152-171 of 1VL6 and, in antibody numbering, 100F-119 of 4JB9. The secondary structure of each residue is taken from the PDB entries (via PDBe). No method that sees only this window can label both correctly. Example from Li et al. (2015).

Even a better model did not overcome this. Qian & Sejnowski (1988, *J Mol Biol* 202:865) trained a network with a hidden layer on windows of single sequences and reached **64.3%**. They concluded that “no method based solely on local information in the protein sequence is likely to produce significantly better results for non-homologous proteins”. The limit was the information in the input, not only the model.

**Turning a sequence into numbers: one-hot encoding.** Before a network can read a sequence, the sequence has to be turned into numbers, because a network computes weighted sums. The obvious idea, numbering the amino acids  $A = 1$ ,  $C = 2$ ,  $D = 3$ , ..., does not work. It tells the model that  $C$  lies between  $A$  and  $D$ , and that  $Y$  (20) is twenty times “more” than  $A$ , which means nothing biologically. Instead, each residue becomes **20 numbers**, all 0 except for a single 1 in the column of that amino acid:

$$A \rightarrow (1, 0, 0, \dots, 0), \quad C \rightarrow (0, 1, 0, \dots, 0), \quad \dots, \quad Y \rightarrow (0, 0, \dots, 0, 1)$$

Every amino acid is then equally different from every other, and the network learns its own weight for each amino acid at each position. A window of 13 residues becomes  $13 \times 20 = 260$  input numbers. This is how Qian & Sejnowski’s network read single sequences, and it is also the encoding of the localisation data in Session 8 (150 residues, 3000 numbers). For DNA, the same idea gives 4 numbers per base.

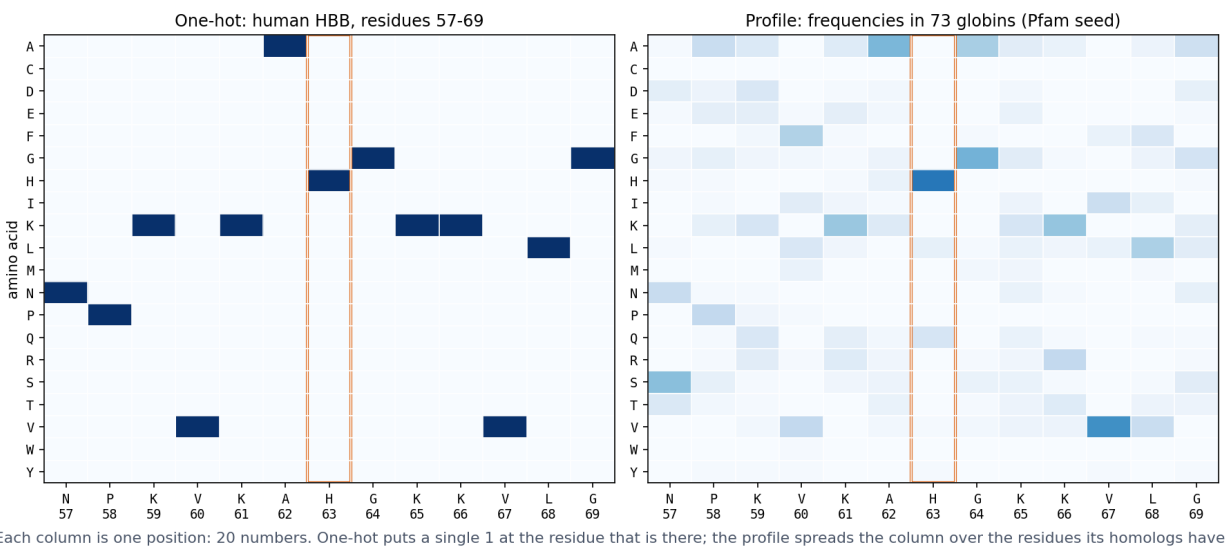


Figure 7.10: The same 13-residue window around His63 in two encodings. Left: one-hot encoding of human HBB, one 1 per position. Right: the profile of the window, the residue frequencies at the aligned positions of the 73 globins in the Pfam seed alignment (Session 4). His63 (orange) is His in most globins; other positions vary much more. A profile column is the average of the one-hot vectors of the aligned sequences. Drawn by tools/builders/make\_session07\_figures.py.

One-hot encoding is not the only choice. A residue can also be described by a few **physicochemical properties** (hydrophobicity, charge, size), which is compact and lets similar residues look similar. It can be described by its row of a **substitution matrix** such as BLOSUM62 (Session 4), or, as PHD does below, by a **profile** of its homologs. Later in the course, the network learns the representation itself. An **embedding layer** (nn.Embedding in PyTorch, used by the recurrent networks of Session 11) gives each amino acid a short vector of trainable numbers. This is exactly a linear layer applied to the one-hot vector, since multiplying a one-hot vector by a weight matrix simply picks out one row. Protein language models (Session 11) go one step further and give each residue a vector that also depends on its context in the sequence.

**What changed with PHD: the use of evolutionary information.** The key innovation of Rost & Sander's **PHD** (1993) was its input. In their words, “a new key aspect is the use of evolutionary information in the form of multiple sequence alignments that are used as input in place of single sequences.” For each position of the window, the network sees a *profile* (Session 5) instead of a one-hot vector, that is, the frequencies of the 20 amino acids at that position among the protein's homologs (the right panel of the figure above). This helps because homologs can retain similar structures even when their sequences have diverged substantially. A common expression is that *structure is more conserved than sequence*, and Illergård, Ardell & Elofsson (2009) quantified it: the structural cores of proteins evolve three to ten times more slowly than their sequences. Where a chameleon stretch sits in a helix, its homologs tend to vary it in the ways that helices tolerate, and where it sits in a strand, in the ways that strands tolerate. The profile can therefore show a difference that the single sequence hides. This family

information alone added “six to eight percentage points”, and the full method reached **70.8%** three-state accuracy. Profile-based predictors also label chameleon sequences better than single-sequence ones (Li et al. 2015).

PHD combined this input with networks at **three levels**:

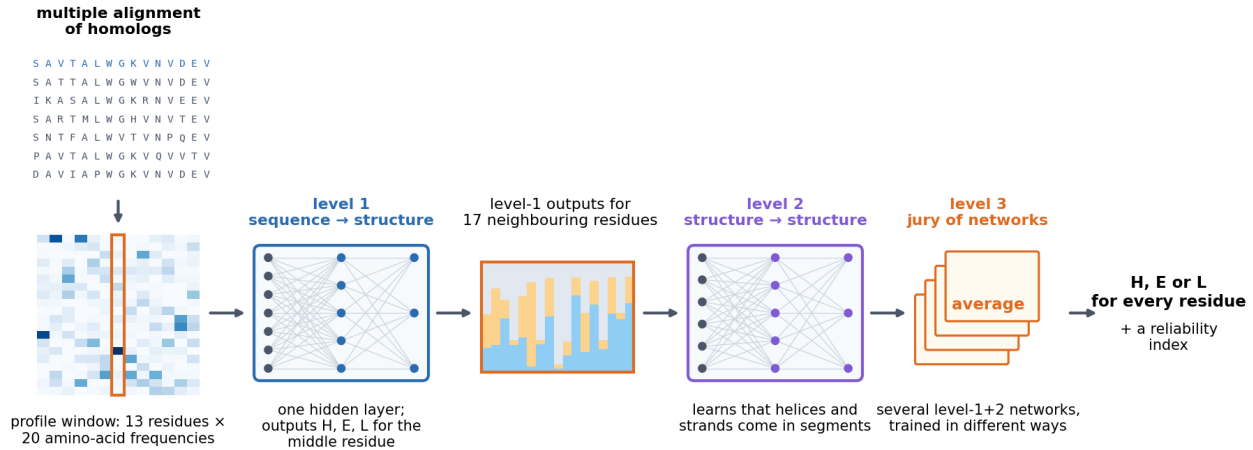


Figure 7.11: PHD’s design. Level 1, a sequence-to-structure network with one hidden layer, reads a window of 13 residues, each described by its 20 profile frequencies from the multiple alignment, and outputs helix (H), strand (E) and loop (L) values for the middle residue. Level 2, a structure-to-structure network, reads the level-1 outputs for a window of 17 residues and learns that helices and strands come in segments of plausible lengths. Level 3, a jury, averages several such networks trained in different ways. PHD also reports how reliable each residue’s prediction is. The alignment and profile shown are schematic.

Each level is a network of the kind described in Section 7.7, only larger: weighted sums, a nonlinear activation, and more weighted sums. The lesson therefore combines both halves of this session. A better representation of the input (profiles) and a model flexible enough to use it (networks with hidden layers) together achieved what additive scores on single sequences could not. Session 9 returns to PHD once you can train such networks yourself.

**What a per-residue predictor has to do: *HBB*.** To see what such a predictor must produce, take beta-globin (*HBB*), this course’s running example. In the crystal structure of human haemoglobin (PDB entry 2HHB), every residue of its chain has a label, either helix (H) or coil (C), and there are no strands.

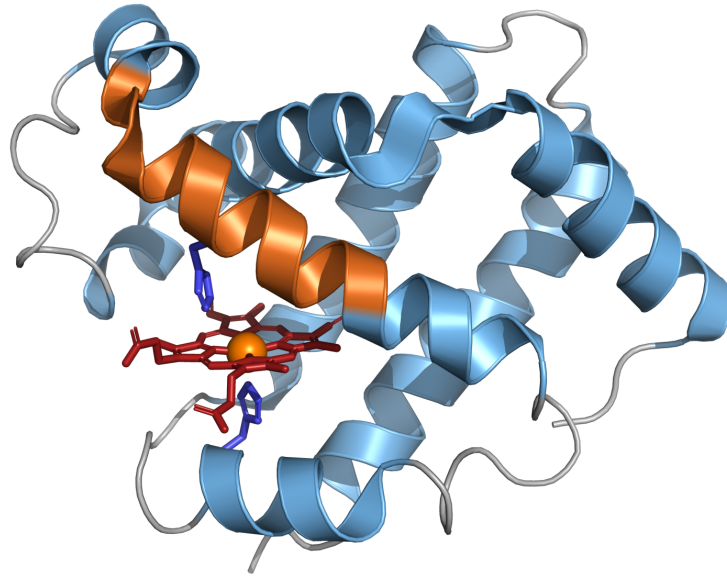


Figure 7.12: *HBB* (chain B of PDB 2HHB): helices in blue, coil in grey. The haem (red, iron in orange) sits in a pocket between helices; His63 (upper, blue sticks) and His92 (lower) are the distal and proximal histidines that Session 4's globin logo showed to be conserved. The orange helix is the 13-residue window around His63 in the next figure. Rendered with PyMOL by `tools/builders/make_session07_hbb_structure.py`.

A secondary-structure predictor has to produce that whole string from the sequence. PHD-style methods label one residue at a time, from a *window* of its neighbours:

Per-residue view: what a secondary-structure predictor must produce (H = helix, C = coil), HBB chain B in 2HHB

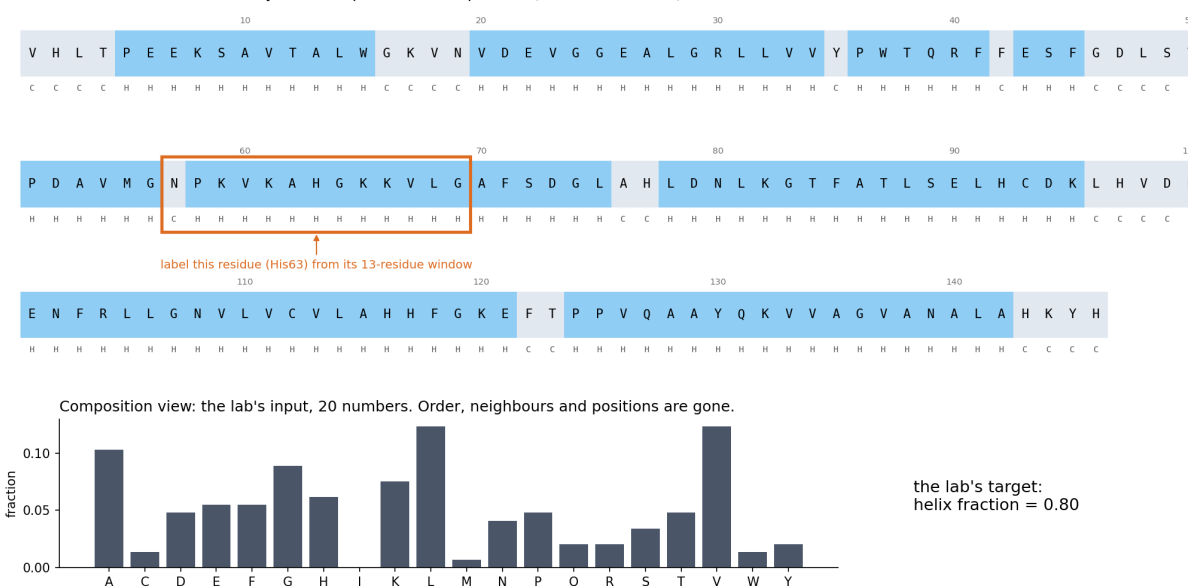


Figure 7.13: Two views of the same protein. Top: HBB's sequence with the secondary structure of each residue in 2HHB (blue = helix, grey = coil), and the 13-residue window a PHD-style predictor reads to label His63. Bottom: the same protein as 20 amino-acid fractions, the input of today's lab, and its single target, the helix fraction. Secondary structure from EBI's PDBe API, drawn by tools/builders/make\_session07\_hbb\_views.py.

The top view keeps what matters for the label: which residues are neighbours, in what order, and where helices start and stop (five of HBB's helices begin at a proline). A single sequence in that window gave Qian and Sejnowski their 64%, and PHD replaced each position of the window by a profile of its homologs (Session 5). The bottom view is what this session's lab uses. *HBB* becomes 20 numbers, and the question becomes "what fraction is helix?" (0.80, 117 of its 146 residues). Everything in the top view that is not a count has been thrown away. Keep both views in mind for the lab.

### Can you do better?

PHD reached about 71% in 1993, a significant improvement over earlier methods. Modern methods that combine profiles or protein language models with deep networks, however, reach substantially higher accuracy on standard benchmarks. In the [capstone project](#) you build your own secondary-structure predictor with the tools of this course. Which input will you give it (a single sequence, a profile, an embedding), which model will you use, and how will you know that it really predicts proteins it has never seen?

### 7.9.1 Further reading

- Wikipedia, [Protein secondary structure](#), [Alpha helix](#), [Beta sheet](#), and [Protein structure prediction](#) — the accuracy figures above
- Chou & Fasman (1974), “Prediction of protein conformation,” *Biochemistry* 13(2):222-245 — [PubMed](#)
- Garnier, Osguthorpe & Robson (1978), “Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins,” *Journal of Molecular Biology* 120(1):97-120 — [PubMed](#)
- Rost & Sander (1993), *Journal of Molecular Biology* 232:584-599 — the original PHD paper
- Illergård, Ardell & Elofsson (2009), “Structure is three to ten times more conserved than sequence — a study of structural response in protein cores,” *Proteins* 77(3):499-508 — [doi:10.1002/prot.22458](#)
- Li, Kinch, Karplus & Grishin (2015), “ChSeq: A database of chameleon sequences,” *Protein Science* 24(7):1075-1086 — [doi:10.1002/pro.2689](#)
- Kabsch & Sander (1984), “On the use of sequence homologies to predict protein structure: identical pentapeptides can have completely different conformations,” *PNAS* 81(4):1075-1078 — [doi:10.1073/pnas.81.4.1075](#)

## 7.10 Representation matters as much as the model

Three examples on this page make the same point, namely that the input representation limits what any model can learn:

- **XOR** becomes a linear problem once the input includes the feature  $x_1x_2$ .
- **Secondary structure prediction** plateaued with single sequences, whatever the method, and improved when the input became a profile of homologous sequences.
- **This session’s lab** predicts a protein’s helix fraction from its 20 amino-acid frequencies. A linear model and a neural network come out essentially tied. Before you read the lab’s explanation, it asks you what information the 20 numbers have thrown away that a per-residue prediction would need. Try to answer it yourself.

A more powerful model cannot recover information that is not in its input. What a model can learn is therefore set by how the data are represented, at least as much as by the model’s capacity. This is one of the central lessons of modern deep learning, and Sessions 10 and 11 build models (convolutional networks, language models) that *learn* their own representations of sequences, including order and local context.

## 7.11 Summary: the two pictures again

### One weighted sum: one straight boundary



PSSM score (no activation) · perceptron (step) · logistic regression (sigmoid) → AND, OR, NAND; not XOR

### Add a hidden layer: several boundaries, combined



multilayer perceptron (MLP), hidden units with tanh or ReLU, e.g. PHD → XOR

*Without the nonlinear activation, the two weighted sums collapse into one, and XOR is impossible again.*

Figure 7.14: The same two pictures as at the start of the page. Everything in this session is one of them.

- **Top:** one weighted sum, then an activation. A PSSM reports the sum as a score, a perceptron thresholds it, and logistic regression passes it through a sigmoid. The weights can be learned from examples by gradient descent (Section 7.4). Whatever the weights, the model draws one straight boundary, so it can represent AND, OR and NAND, but never XOR (Sections 7.5–7.6).
- **Bottom:** a hidden layer of weighted sums with a nonlinear activation, then one more weighted sum. Each hidden unit draws its own boundary, and the output combines them, so XOR becomes possible (Section 7.7). Remove the nonlinear activation and the two layers collapse back into the top picture.
- **Either picture** can only use the information in its input (Sections 7.8–7.10).

If you can explain what changes between the two pictures, and why that change allows XOR, you have understood this session. You do not need to be able to write the PyTorch training loop from memory.

## 7.12 Check your understanding

The questions below give immediate feedback and cover the sections above. They are in the same style as the course’s live in-class peer-instruction quizzes, and the full question bank is in quizzes/session07.md in the repository.

*The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)*

## 7.13 The course notebook

The course notebook for this session ([notebooks/session07-gradient-descent-xor.ipynb](#)) follows the order of this page. It opens by comparing a least-squares solver with gradient descent on the same tiny dataset, and confirms that they agree. Hand-chosen weights then implement AND, OR and NAND (checked against their truth tables), and gradient descent learns AND by itself. The same model trained on XOR visibly fails. Combining the OR and NAND perceptrons with AND solves XOR exactly, and a small network with one hidden layer then learns it, with the shape of every tensor printed and a plot showing why. The same network with its activation removed fails like the linear model, and a single engineered input,  $x_1x_2$ , makes XOR linear without changing the model. The notebook closes by fetching *HBB*'s per-residue secondary structure live from EBI's PDB API and printing the two views of the protein from the figure above: the labels with a 13-residue window, and the 20 composition numbers. The version below is a static, read-only copy. Open it in [Google Colab](#) via the badge at the top of the notebook page to edit and run it yourself.

## 7.14 The lab

**A teaser, not a full treatment.** Session 8 covers train/validation/test splitting and evaluation properly, and Session 9 covers regularisation and architecture choices, so neither is anticipated here. In the lab you assemble a real dataset, the amino-acid composition of 55 real proteins, fetched live from EBI's PDB API, against their real alpha-helix fraction. You then fit linear regression and a small from-scratch multilayer perceptron on the same data and compare their error on held-out proteins. The two come out essentially tied on this small split, and the lab asks why a more flexible model does not help here. You first answer that question yourself (what has a composition vector thrown away?), and a scrambled-sequence demonstration then shows the answer, which links back to the windows and profiles that PHD used. (A cached copy of the dataset is used if the live fetch fails.) [notebooks/session07-lab.ipynb](#) has the real pipeline, end to end.

### 7.14.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## 7.15 What's next

Session 8 moves from a single yes/no output (this session's XOR problem) to multi-class prediction. It also asks the question that matters in practice: how do we know whether a model has learned anything useful? To answer it, Session 8 uses a real multi-class

biological problem, subcellular localisation. Session 9 then trains networks properly, and shows why training sometimes fails.

## 8 Session 8: Machine Learning Foundations & Practical PyTorch

### Note

**Sources for this page:** d2l.ai, *Dive into Deep Learning*, [Softmax Regression](#), [Generalization in Classification](#) and [Builders' Guide](#) (`nn.Module`) — CC BY-SA 4.0; Wikipedia, [Training, validation, and test data sets](#), [Supervised learning](#), [Regression analysis](#), [Statistical classification](#), [Logistic regression](#), [Leakage \(machine learning\)](#), [Confusion matrix](#), [Matthews correlation coefficient](#), [Receiver operating characteristic](#), [Activation function](#) — CC BY-SA 4.0; UniProt, [Subcellular location](#) — CC BY 4.0; EBI's PDBe API for the helix-fraction data; Steinegger & Söding (2017), [MMseqs2](#), *Nature Biotechnology* 35:1026-1028, for sequence clustering. The 3×3 pattern-recognition example, the classifier-evaluation material, and the emphasis on homology reduction of training sets draw on ideas from Magnus Ekman's *Learning Deep Learning* (ch. 2, ch. 4 and Appendix A slide decks in `slides/`, the ch. 4 deck bundling Olof Emanuelsson's "Neural Networks in bioinformatics" slides). That book is copyrighted and not openly licensed, so it is credited here as an internal source and not linked as further reading. All figures on this page were drawn or computed for this course. Every number was computed in code and read back from the notebook's output.

### Overfitting

Why did the deep learning model refuse to attend the party?

It didn't want to overfit in.

### 8.1 Learning goals

By the end of this session you should be able to:

- explain what training, validation and test sets are for, and why performance on the training data says little about a model;
- name the ways information can leak from the test data into a model (through the samples, the preprocessing or the model choices), homologous sequences above all;
- compute and interpret a confusion matrix, precision, recall, F1, MCC and ROC/AU-ROC for a classifier, and RMSE, Pearson  $r$  and  $R^2$  for a regression model, and explain what each of them does and does not tell you;

- tell regression from classification, and choose the output, loss and evaluation metric that go with each;
- write (not yet train) a PyTorch `nn.Module` classifier and count its parameters.

Session 7 asked what a model *can* represent. This session asks the question that matters in practice: **how do we know whether a model has learned anything useful?**

The session has three parts. Section 8.2 poses the problem: is it regression or classification? Section 8.3, the heart of the session, is about evaluation and generalisation. Section 8.4 builds the bridge to Session 9, with activation functions, softmax and a PyTorch model. A tiny 3×3-pixel pattern problem runs through all three parts as a toy example, and a real protein-localisation dataset carries the biology.

## 8.2 Posing the problem

### 8.2.1 Supervised and unsupervised learning

Before choosing a model, we need to know what kind of learning problem we have. Machine learning distinguishes two kinds of learning.

- **Supervised learning** fits a model to *labelled* examples, where the correct answer for each input is already known, so that it can predict the label of new inputs. An example is a perceptron trained on known binding and non-binding sites (Session 7).
- **Unsupervised learning** finds structure in data *without* labels, for example by clustering sequences into families, or by projecting expression profiles with PCA. The MMseqs2 clustering in Section 8.3 is an unsupervised example.

A PSSM (Session 5) sits in between, because it is estimated by counting residues in an alignment of family members rather than fitted to labelled positive and negative examples. Both kinds of learning are useful, but almost everything we train in this course is supervised, so that is what we focus on.

### 8.2.2 Regression vs. classification

Within supervised learning, the kind of problem is decided by the *label*:

- in **regression**, the label is a **number**;
- in **classification**, the label is one of a fixed set of **categories**.

Table 8.1: Regression and classification problems in biology.

| Regression (predict a number)           | Classification (predict a category)        |
|-----------------------------------------|--------------------------------------------|
| Fraction of residues in helices (below) | Subcellular compartment: 5 classes (below) |

| Regression (predict a number)                                                   | Classification (predict a category)                           |
|---------------------------------------------------------------------------------|---------------------------------------------------------------|
| Protein melting temperature, or stability change $\Delta\Delta G$ of a mutation | Is this mutation pathogenic or benign?                        |
| mRNA expression level in a tissue                                               | Does this protein have a signal peptide? (Session 11)         |
| Binding affinity ( $pK_d$ ) of a drug to a target                               | Helix / strand / coil for each residue (Session 7, Session 9) |
| Relative solvent accessibility of a residue (0-1)                               | Is this residue buried or exposed?                            |

The last row shows that the same biology can often be posed either way. Setting a threshold on a continuous quantity turns regression into classification, which makes the problem easier but throws information away.

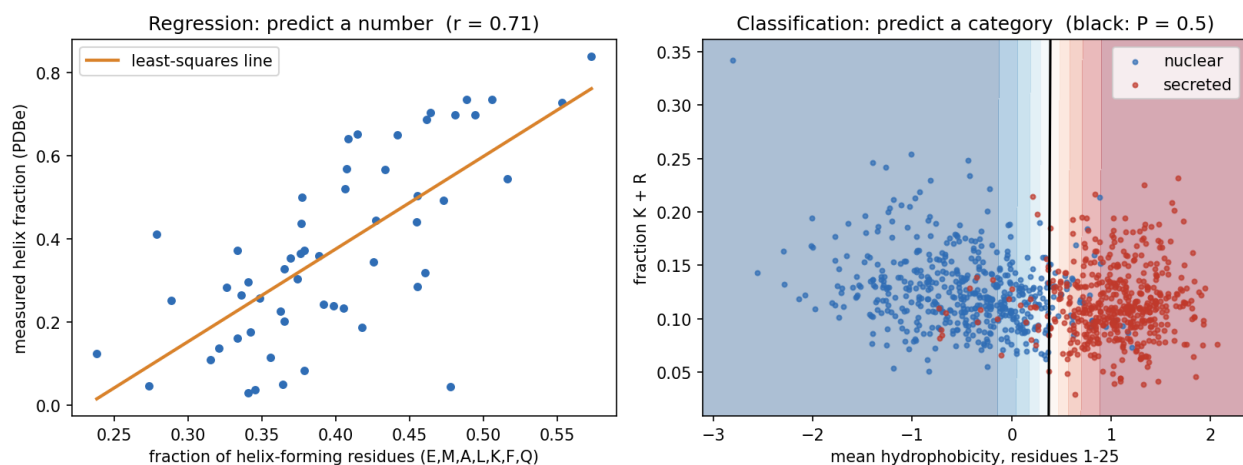


Figure 8.1: Two real problems. **Left, regression:** for 55 PDB entries (fetched live from PDBe), the fraction of residues in helices against the fraction of strong Chou-Fasman helix formers (E, M, A, L, K, F, Q) in the sequence. A straight line captures a real trend ( $r = 0.71$ ) with plenty of scatter. **Right, classification:** 488 secreted and 497 nuclear human proteins from UniProt, plotted by the hydrophobicity of their first 25 residues and their lysine + arginine content. Logistic regression draws the black boundary where  $P(\text{secreted}) = 0.5$ .

**What is a signal peptide?** Proteins are made in the cytoplasm, but many must work elsewhere: outside the cell, in a membrane, or in an organelle of the secretory pathway. Most of them carry this address at their N-terminus as a **signal peptide**, typically 15-30 residues with a hydrophobic core. It sends the new chain into the endoplasmic reticulum (in bacteria, across the inner membrane), and it is then cut off. Session 4 shows its sequence motif, and Session 11 its role in the history of deep learning.

Predicting signal peptides is one of the oldest problems in bioinformatics, and its early history is a short course in this session's theme (Nielsen, Tsirigos, Brunak & von Heijne 2019, [Protein J. 38:200](#)):

- **The first cleavage-site rule** (von Heijne 1983) placed 92% of cleavage sites correctly in the 78 signal peptides used to build it, but only 64% in a larger set a few years later. Even a simple method can be overfitted if the data are few.
- **The first signal-peptide predictor** (McGeoch 1985, [Virus Res. 3:271](#)) used two features much like ours: the length of the uncharged stretch near the N-terminus and the maximal hydrophobicity. The boundary between the classes was drawn *by hand* in that two-dimensional plane, so it could not simply be refitted to new data.
- **The first neural network** (Ladunga et al. 1991, [Bioinformatics 7:485](#)) added hidden units until it classified every training protein correctly, and then performed poorly on new proteins.

Our classifier does what McGeoch did by hand, but it learns the boundary from data and we test it on proteins it has not seen. It tells a biological story. It gives N-terminal hydrophobicity a weight of 4.26 and K+R content a weight of only  $-0.20$ , so it found the hydrophobic core of the signal peptide almost on its own. On held-out proteins it is right 93.5% of the time, from just two numbers per protein:

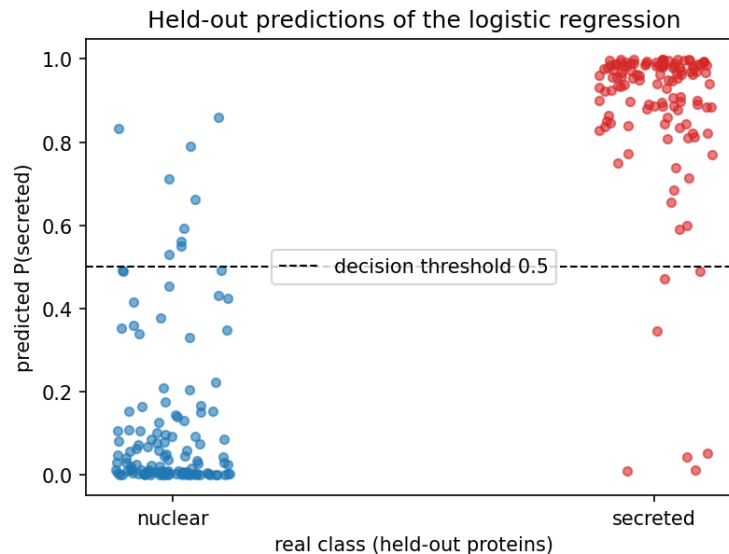


Figure 8.2: The predicted  $P(\text{secreted})$  for each held-out protein, grouped by its real class (spread sideways so the points do not overlap). Almost all nuclear proteins are far below the threshold of 0.5 and almost all secreted proteins far above it; the few on the wrong side are the classifier’s errors.

#### 8.2.2.1 Logistic regression: the bridge between the two

Logistic regression computes the same weighted sum as Session 7’s perceptron,  $z = w_1x_1 + w_2x_2 + b$ . Instead of a hard step, it passes  $z$  through the **sigmoid** function:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

which maps any number to a value between 0 and 1. Despite its name, logistic regression is a *classifier*. Because it is fitted with the log-loss (cross-entropy), its output can be read as an estimate of  $P(\text{secreted} \mid x)$ . In the notebook, for example, one nuclear protein gets 0.008 and one secreted protein 0.980. Whether such estimates are well *calibrated*, meaning that a predicted 0.8 is right 80% of the time, is a separate question that has to be checked on held-out data. It is important to keep two things apart. The **sigmoid** is just a mathematical mapping into the range 0-1, whereas **logistic regression** is a probabilistic model, fitted with the likelihood (cross-entropy) of the labels. Only the second makes the output an estimate of a probability, and a sigmoid applied to an arbitrary score does not.

The decision boundary ( $\sigma(z) = 0.5$ , i.e.  $z = 0$ ) is still the perceptron's straight line. What changes is that the output is now smooth and differentiable, which is what Session 9's training needs.

Once the problem type is chosen, three things downstream follow from it:

Table 8.2: What follows from the choice between regression and classification.

|                                           | Regression                | Classification                                            |
|-------------------------------------------|---------------------------|-----------------------------------------------------------|
| Output unit                               | linear (no activation)    | sigmoid (2 classes) /<br>softmax ( $K$ classes)           |
| Loss minimised in training<br>(Session 9) | mean squared error        | cross-entropy (Session 1)                                 |
| Evaluation metrics<br>(Section 8.3)       | RMSE, Pearson $r$ , $R^2$ | confusion matrix,<br>precision, recall, F1, MCC,<br>AUROC |

### 8.2.3 A running toy example: 3×3 pixel patterns

Deep learning had its first great breakthrough in image classification (Session 10), where a computer learns whether an image contains a cat, a house, and so on. To see how a classifier is evaluated, we play here with the smallest image-classification problem worth the name. Each image has nine pixels on a continuous scale (dark = +1, light = −1), and there are five classes: horizontal, vertical, diagonal, anti-diagonal and cross. Every example is a clean pattern plus random noise on each pixel (Gaussian,  $\sigma = 0.8$ ). “Cross” shares its middle row with “horizontal” and its middle column with “vertical”, so we can expect some confusion there.

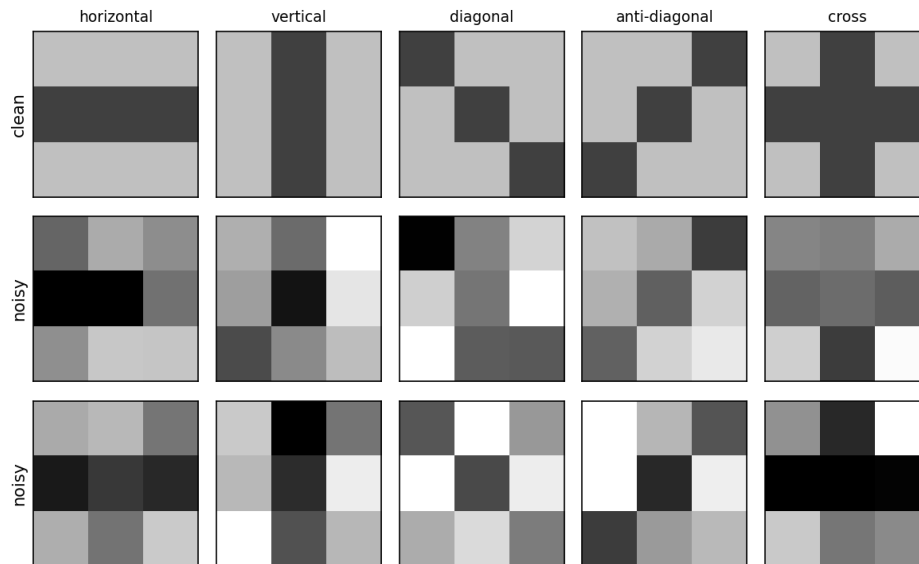


Figure 8.3: The five clean  $3 \times 3$  patterns (top row) and two noisy examples of each. The noise is strong enough that some examples are hard even for a human.

The classifier is five Session 7 perceptrons, one per pattern. Each is trained to output  $+1$  for its own pattern and  $-1$  for every other, and the predicted class is whichever perceptron gives the highest score. This is called **one-vs-rest**. There is no hidden layer, so nothing here needs backpropagation. Section 8.3 uses this model to show what the three data sets are for and what the evaluation metrics mean.

### 8.2.3.1 Further reading

- Wikipedia, [Supervised learning](#), [Regression analysis](#), [Statistical classification](#)
- Wikipedia, [Logistic regression](#)
- Wikipedia, [Chou-Fasman method](#) (helix propensities) and [Hydrophilicity plot](#) (Kyte-Doolittle scale)
- d2l.ai, [Linear Regression](#)

## 8.3 How do we know a model learned anything? Evaluation and generalisation

Datasets should be representative

An often-told story: an army wanted a neural network to recognise tanks in photographs. They took many pictures of an area with tanks, and on another day many pictures of the same area without them. The network learned to separate the two sets perfectly, but on new photographs it was no better than guessing. Why? All the pictures with tanks had been taken on a sunny day and all those

without on a cloudy day, so the network had become a perfect detector of the weather. (The story may be partly legend, but the failure it describes is real and common.)

### 8.3.1 Why three data sets?

A model is only useful if it works on data it has **not** seen. Its performance on the training data says little, because a flexible enough model can memorise every training example. A model that predicts its training data accurately but fails on new examples is said to **overfit**. The labelled data are therefore split three ways. This is not always done in practice, but it should be:

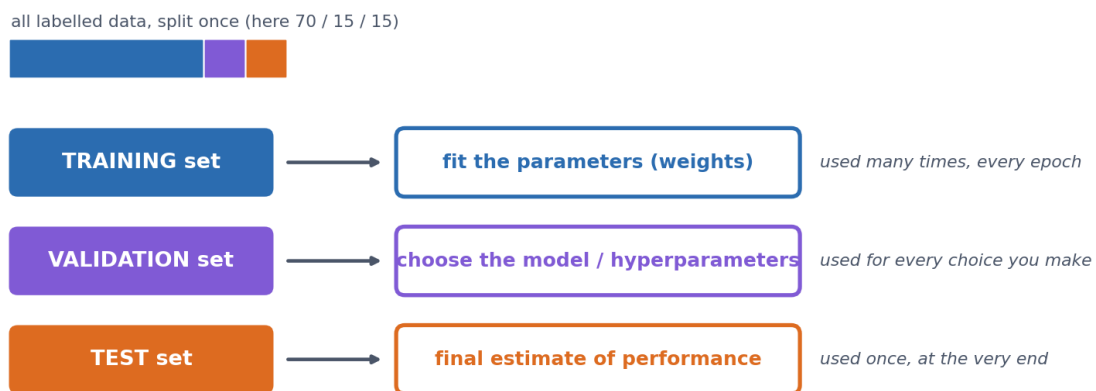


Figure 8.4: The three data sets and what each one is for.

- **Training set:** used to fit the weights.
- **Validation set:** used to make *choices*, such as how long to train, how large a network to use, or which features to include. Every choice tuned on it makes it slightly optimistic.
- **Test set:** looked at **once**, at the very end, to report the final number. If you go back and change the model after looking at the test set, it has become a second validation set and no longer gives an honest estimate.

**On the 3×3 problem.** We draw 100 training, 100 validation and 200 test examples, and first measure the model by its **accuracy**, the fraction of examples, over all five classes, that it classifies correctly. (The next section breaks this single number down.) The one choice to make is how many **epochs** (passes through the training set) to train for, and we make it on the validation set. If we train for 50 epochs and record both accuracies after every epoch, we get this picture (the notebook computes every number and draws the figure):

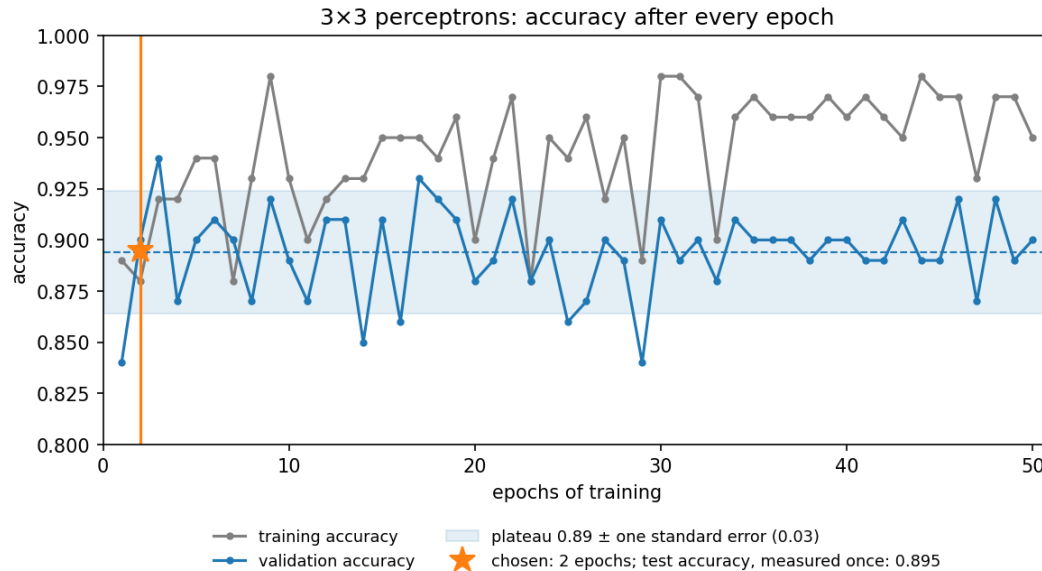


Figure 8.5: Training (grey) and validation (blue) accuracy of the 3×3 perceptrons after every epoch. The shaded band is the plateau of the validation curve (its mean, 0.89) plus or minus the standard error of an accuracy measured on 100 examples (about 0.03). The orange line marks the chosen model, 2 epochs, and the star its test accuracy, measured once: 0.895.

The figure shows three things:

- **Training accuracy never settles.** It moves between 0.88 and 0.98. The noisy, overlapping patterns cannot all be separated by straight lines, so the perceptron rule keeps changing the weights for every example it gets wrong. High training accuracy is not the goal, because the model is judged on data it has not seen.
- **Validation accuracy is noisy.** After the first epoch it moves around a plateau of about 0.89, between 0.84 and 0.94. With only 100 validation examples, one accuracy value is uncertain by about  $\pm 0.03$  (its standard error,  $\sqrt{p(1-p)/n}$ ), so most of the epoch-to-epoch movement is noise. After about 30 epochs, training accuracy is clearly higher than validation accuracy, which is the first sign of overfitting.
- **Choosing the single best epoch is optimistic.** Epoch 3 happens to reach 0.94, but picking the maximum of 50 noisy numbers partly picks luck. This is **selection bias**. Choosing the maximum of many noisy validation scores makes the reported validation performance optimistic, so the chosen model's validation accuracy overstates how well it will do. (Making choices on the validation set is what it is for. The bias lies in reporting the best score as the expected one.) Tuning many choices on the same validation set can, in the end, overfit the validation set too. We therefore use the earliest epoch that has clearly reached the validation plateau, rather than chasing the highest noisy value, which gives **2 epochs** (0.90).

The model trained for two epochs is then measured **once** on the test set, giving **0.895**, in line with the plateau. But is that good? Accuracy alone can mislead, in particular when the classes are unbalanced, and that is the topic of the next section.

### 8.3.2 Evaluating a classifier

How useful is accuracy?

You want to build a predictor of who will pass their PhD defence at a Swedish university. Each year about 1000 PhD students defend, and on average one of them fails. How would you judge the predictor?

If you *predict* that everyone will pass, you are right 99.9% of the time. But does that tell you anything?

The Matthews correlation coefficient (MCC, below) of that prediction is 0.00.

Accuracy is a single number, and it hides *which* mistakes are made. The **confusion matrix** shows them, because it cross-tabulates the true class (rows) against the predicted class (columns).

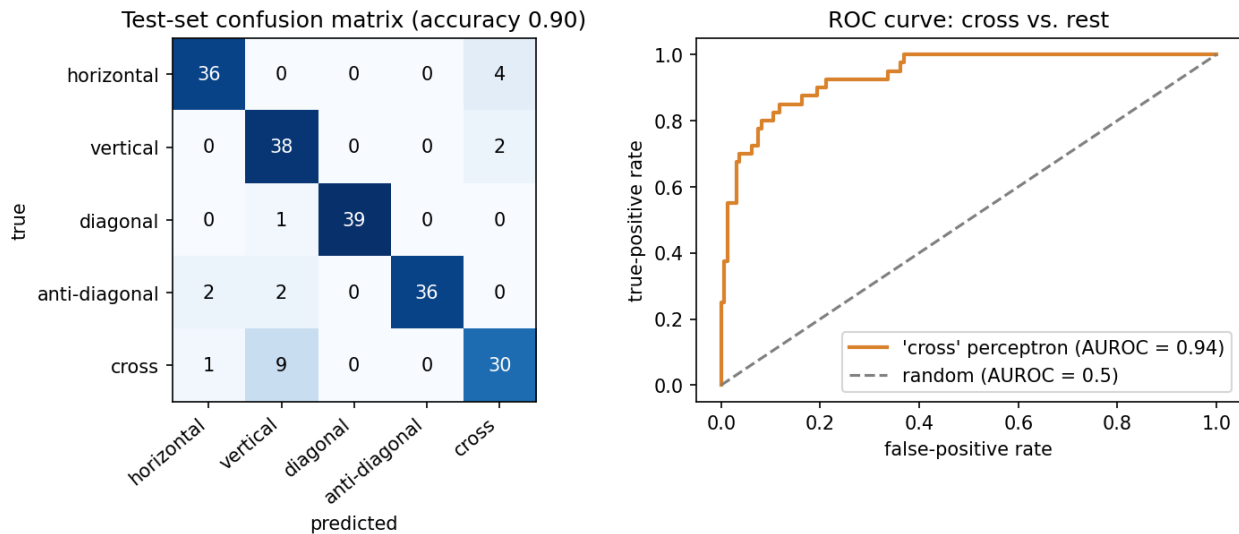


Figure 8.6: Left: test-set confusion matrix for the 3×3 perceptrons. Almost all mistakes involve “cross”: 9 crosses predicted as “vertical”, and 4 horizontals and 2 verticals predicted as “cross”. These are the pairs that share pixels. Right: the ROC curve of the “cross” perceptron’s score, cross vs. everything else.

For any classification task, even one with many classes, we can treat one class as “positive” and all the others together as “negative”. Every prediction is then a true or false positive (TP, FP) or a true or false negative (TN, FN). From these four counts we can define several useful measures, which are worth remembering:

$$\text{precision} = \frac{TP}{TP + FP} \quad \text{recall} = \frac{TP}{TP + FN}$$

- **Precision** asks: of the proteins I called positive, how many really are?

- **Recall** (sensitivity) asks: of the true positives, how many did I find?

For “vertical” on the 3×3 test set, recall is 0.95 but precision only 0.76, because many crosses were called vertical. For “cross”, recall drops to 0.75. There is always a trade-off between precision and recall, and the application decides which matters more. In a screening application, such as flagging potentially pathogenic variants for follow-up, you would favour *recall*. When each positive call triggers an expensive experiment, you would favour *precision*.

Two measures combine these counts into a single number.  $F_1$  balances precision and recall:

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

The **Matthews correlation coefficient** uses all four counts:

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

MCC runs from  $-1$  (perfect disagreement: every prediction inverted) through  $0$  (no better than chance) to  $+1$  (perfect), and it stays honest when the classes are unbalanced (see below). The formula above is for two classes. For problems with more classes, software such as scikit-learn uses a generalisation that works on the whole confusion matrix. Over all five 3×3 classes, that multi-class MCC is 0.871.

### 8.3.2.1 ROC curves

Most classifiers output a *score*, and a threshold turns it into a yes/no call. Lowering the threshold finds more true positives but also lets through more false positives, and raising it does the opposite. Choosing one cutoff is somewhat arbitrary, so it is often better to look at all thresholds at once. The **ROC curve** does this by plotting the true-positive rate against the false-positive rate as the threshold sweeps across all values. To turn the curve into a single number, we use the area under it, **AUROC**. This is the probability that a random positive gets a higher score than a random negative, so it is 0.5 for guessing and 1.0 for a perfect ranking. For the “cross” perceptron above, AUROC is 0.938. The same reasoning lies behind Session 5’s choice of an E-value cutoff, where you also pick a point on this trade-off.

### 8.3.3 Evaluating a regression model

A regression prediction is not simply right or wrong, but more or less off, so it needs other measures. For the helix-content example, we fit the line on 41 proteins and hold out the other 14:

- **RMSE** =  $\sqrt{\frac{1}{n} \sum (y_i - \hat{y}_i)^2}$  is 0.159, in the target's own units (fraction helix). Predicting the training-set mean for every protein gives 0.234, which is the baseline to beat.
- **Pearson  $r$**  between predicted and measured values is 0.760.
- $R^2 = 1 - \sum (y_i - \hat{y}_i)^2 / \sum (y_i - \bar{y})^2$  is 0.526, so about half of the variance is explained.  $R^2 = 0$  means no better than predicting the mean.

The three numbers answer different questions:

- **RMSE**: how numerically wrong are the predictions, in the target's units?
- **Pearson  $r$** : do the predictions follow the ordering and linear trend of the truth?
- $R^2$ : how much better are the predictions than always predicting the mean?

They can disagree sharply. A systematically biased predictor, for example one that outputs  $2\hat{y} + 0.3$  instead of  $\hat{y}$ , keeps exactly the same correlation ( $r = 0.760$ ), because the ordering is unchanged. But its RMSE rises from 0.159 to 0.639, and its  $R^2$  falls to  $-6.67$ , far worse than predicting the mean. A high correlation is therefore not the same as accurate prediction, and treating it as such is one of the most common mistakes in bioinformatics papers.

**Why we need held-out data, in one picture.** A model that is flexible enough can simply memorise its training data. A **1-nearest-neighbour** model is the extreme case, because it predicts the helix fraction of the most similar protein (by amino-acid composition) that it was trained on. If we train it on all 55 proteins and evaluate it on the same 55, every protein's nearest neighbour is itself, so the predictions are perfect ( $R^2 = 1.00$ , RMSE = 0). If we train it on 41 proteins and evaluate it on the other 14, it is no better than predicting the mean ( $R^2 = 0.03$ ):

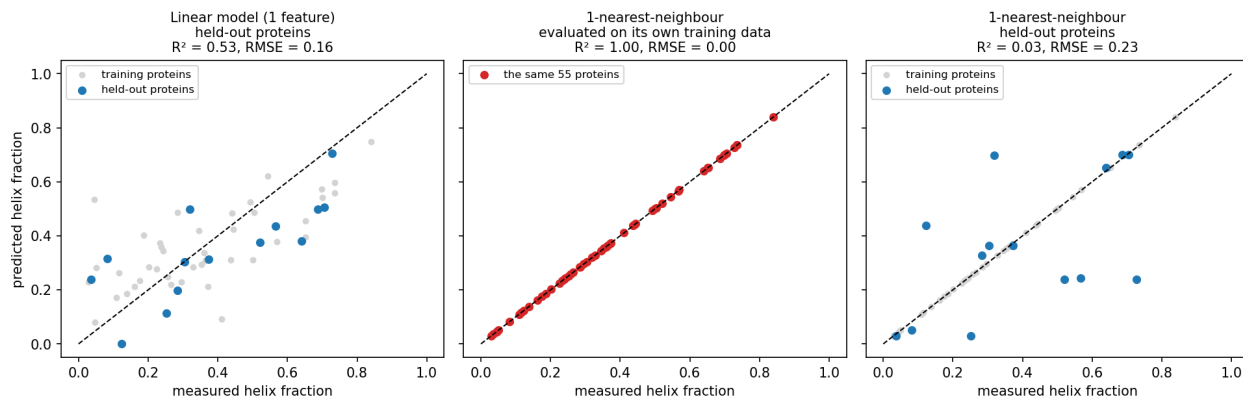


Figure 8.7: Predicted against measured helix fraction; points on the dashed line are perfect predictions. Left: the one-feature line, on the 14 held-out proteins (blue; training proteins in grey). Middle: 1-nearest-neighbour, trained and evaluated on the same 55 proteins: apparently perfect. Right: the same kind of model, trained on 41 proteins and evaluated on the 14 it has not seen.

The middle panel is what you get if you evaluate on the training data, and it says nothing about new proteins. With only 14 test proteins, however, all of these numbers are themselves uncertain, which leads to the next question.

### 8.3.4 How much can you trust one split?

To answer it, we turn to a real, though still small, problem: predicting subcellular localisation. The dataset comes from reviewed human UniProt entries (Session 3) with exactly one annotated location. We use five classes (cytoplasm, nucleus, mitochondrion, secreted, cell membrane) and leave out all others (peroxisome, Golgi, ER, and so on). Each sequence is one-hot encoded (Session 7) over its first 150 residues, because localisation signals are mostly N-terminal, which gives  $150 \times 20 = 3000$  input numbers per protein.

**Class imbalance first.** Before balancing, the classes hold 424, 497, 140, 488 and 338 proteins. A “classifier” that always answers *nucleus* reaches an accuracy of 0.263, above the 0.20 you might expect for five classes, while its MCC is exactly 0. Real proteomes are far more unbalanced than this. In a genome-wide screen for, say, a rare targeting signal, “always no” can be more than 90% accurate. You should therefore always report MCC, or precision and recall, alongside accuracy, and be sceptical of papers that do not. Here we avoid the problem by balancing every class to 140 proteins (700 in total).

**Split-to-split noise.** A quick logistic-regression baseline, refitted on ten different random 70/15/15 splits, gives validation accuracies from 0.505 to 0.610 (mean 0.561, standard deviation 0.031). The data and the model are the same, and only the random draw of the split differs. **k-fold cross-validation** makes better use of limited data and reduces the dependence on one particular split, although it does not remove sampling variability. The data are cut into  $k$  class-stratified folds, the model is trained  $k$  times with a different fold held out each time, and the results are averaged. With  $k = 5$  the notebook gets  $0.540 \pm 0.044$ . Every protein is used for validation exactly once, at  $k$  times the computing cost.

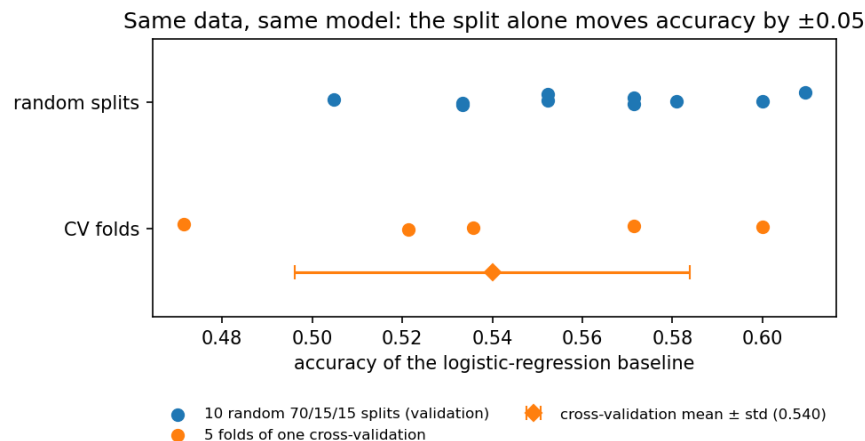


Figure 8.8: Validation accuracy of the same logistic-regression baseline on ten random splits of the same 700 proteins (blue), and on the five folds of one 5-fold cross-validation (orange, with its mean and standard deviation). Only the random choice of proteins differs.

### 8.3.5 The biological trap: homologs on both sides of the split

A random split assumes that the examples are independent. **Proteins are not.** They come in families of homologs (Session 4) that share sequence and, often, structure, localisation and function. Suppose one family member is in the training set and another in the test set. The model can then exploit family-specific sequence similarity rather than relying only on general targeting signals. The test set then partly rewards recognising known families instead of measuring generalisation to new kinds of proteins.

This effect can be measured. Clustering the 700 proteins with **MMseqs2** (a fast, BLAST-like search that groups sequences above an identity cutoff) at 30% identity gives 551 clusters. 75 clusters have more than one member, holding 224 proteins in total, and the largest has 11. Over the same ten random splits:

- **31.3%** of validation proteins have a  $\geq 30\%$ -identity homolog in the training set;
- the baseline classifies those correctly **67.5%** of the time;
- it classifies the remaining proteins correctly only **50.9%** of the time.

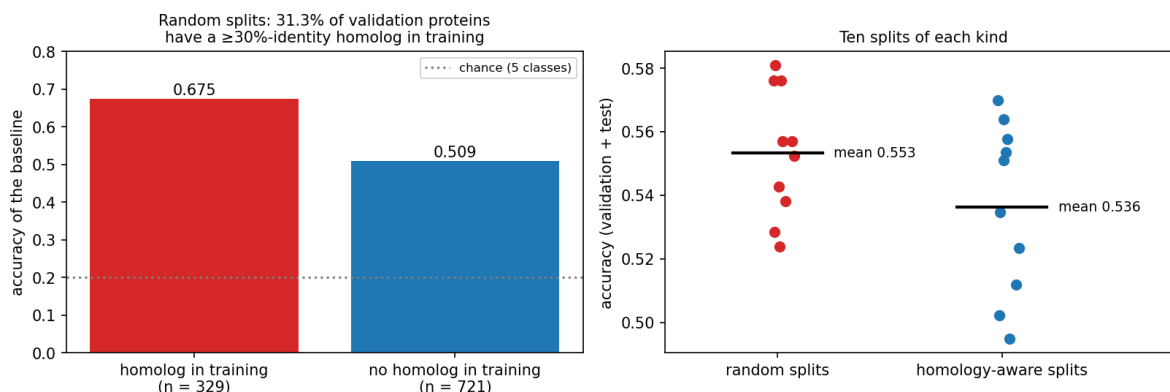


Figure 8.9: Left: over ten random splits, validation proteins that have a homolog ( $\geq 30\%$  identity) in the training set are classified correctly far more often than those that do not. Right: baseline accuracy on ten random and ten homology-aware splits; each dot is one split, the black line the mean.

The fix is **homology partitioning**. We cluster first and then put whole clusters into training, validation or test, so that no cluster of sequences with  $\geq 30\%$  identity crosses the split. This substantially reduces homology leakage, although it cannot guarantee that no remote homolog below the cutoff remains. Averaged over ten splits of each kind, baseline accuracy drops from 0.553 (random) to 0.536 (homology-aware). The overall drop is small *here* only because 476 of the 700 proteins have no homolog in the set at all. In more redundant data, such as a whole proteome or the PDB with its thousands of near-identical structures of popular proteins, the gap can be much larger. **From this session on, the course uses the homology-aware split** (486 training / 105 validation / 109 test proteins), and Sessions 9 and 10 reuse it.

### 8.3.5.1 Other ways biological data leaks

Homology is the classic trap, but not the only one. Information from the test data can leak into a model in three ways: through the **samples** (homologs or samples from the same patient on both sides of the split), through the **preprocessing** (normalising or selecting features using all the data), and through the **evaluation** (repeatedly inspecting the test set while choosing models). Using the *validation* set to choose a model is not leakage, because that is what it is for. The common pattern is that something shared between the training and test examples, other than the signal you want to learn, lets the model cheat.

- **Several samples from one individual.** Split by patient, not by sample. Otherwise the model learns to recognise the patient.
- **Batch effects.** If all tumour samples were sequenced in one lab and all controls in another, a model can learn the lab. Keep batches balanced across classes, or split by batch.
- **Preprocessing on the full dataset.** Normalising, selecting features or running PCA on *all* data before splitting lets test-set information in. Fit preprocessing on the training set only.

- **Time.** A model tested on proteins annotated *before* its training data may benefit from later annotations that were themselves predicted by similar tools. **CASP** (Session 6) avoids all of this by testing on structures that are unpublished at prediction time. This truly blind test is the model for honest evaluation.
- **Unrepresentative data.** Swiss-Prot over-represents well-studied proteins and model organisms. A model trained on human proteins may not transfer to bacteria. The training set should look like the data the model will be used on.

#### 8.3.5.2 Further reading

- Wikipedia, [Training, validation, and test data sets](#), [Cross-validation \(statistics\)](#), [Leakage \(machine learning\)](#), [Batch effect](#)
- Wikipedia, [Confusion matrix](#), [Precision and recall](#), [Matthews correlation coefficient](#), [Receiver operating characteristic](#), [Coefficient of determination](#)
- d2l.ai, [Generalization in Classification](#)
- Steinegger & Söding (2017), [MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets](#); software at [github.com/soedinglab/MMseqs2](https://github.com/soedinglab/MMseqs2)
- Wikipedia, [CASP](#)

## 8.4 Towards training networks

This part prepares Session 9, in which networks are trained. It covers how units are connected, the activation functions that make gradient descent possible, softmax outputs, and how to write a model in PyTorch and count its parameters.

### 8.4.1 Connectivity decides the number of weights

A network connects many units of the kind that Session 7 introduced. How they are connected decides how many weights a layer has, and it also builds an assumption about the data into the model:

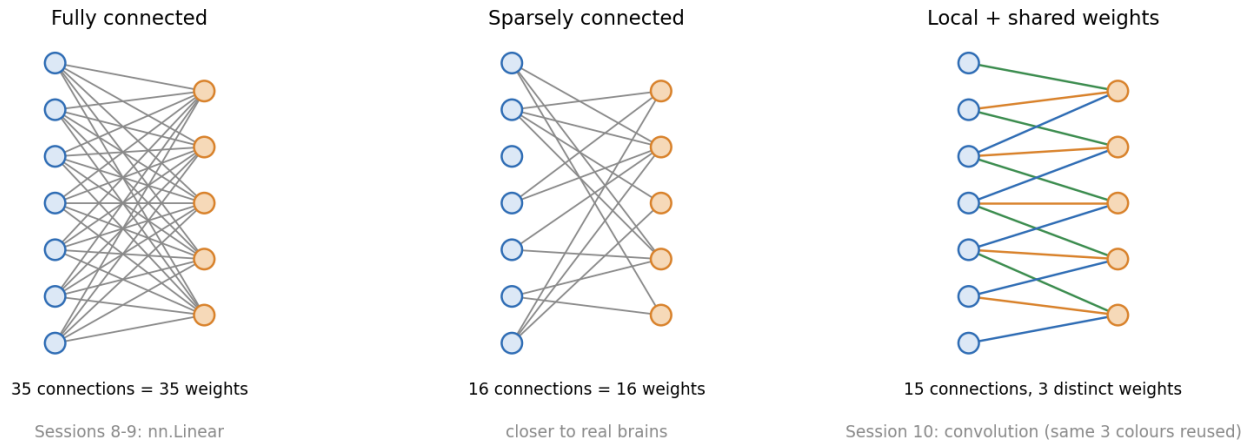


Figure 8.10: How units are connected decides how many weights a layer has. Fully connected: every input to every output (Sessions 8-9). Sparse: only some connections, closer to real brains. Local with shared weights: each output sees only a small window of neighbouring inputs, and every window reuses the same three weights. This is convolution, Session 10's key idea, and it is why convolutional networks need so many fewer parameters.

A fully connected layer (`nn.Linear`) connects every input to every output, so its size grows with the length of the input. The third discussion problem compares it with a convolution, whose number of weights does not depend on the input length at all.

**i** Optional: biological and artificial networks, and the architectures still to come

Session 7 introduced the perceptron as Rosenblatt's simplified neuron. Beyond single units, the analogy with biology becomes loose:

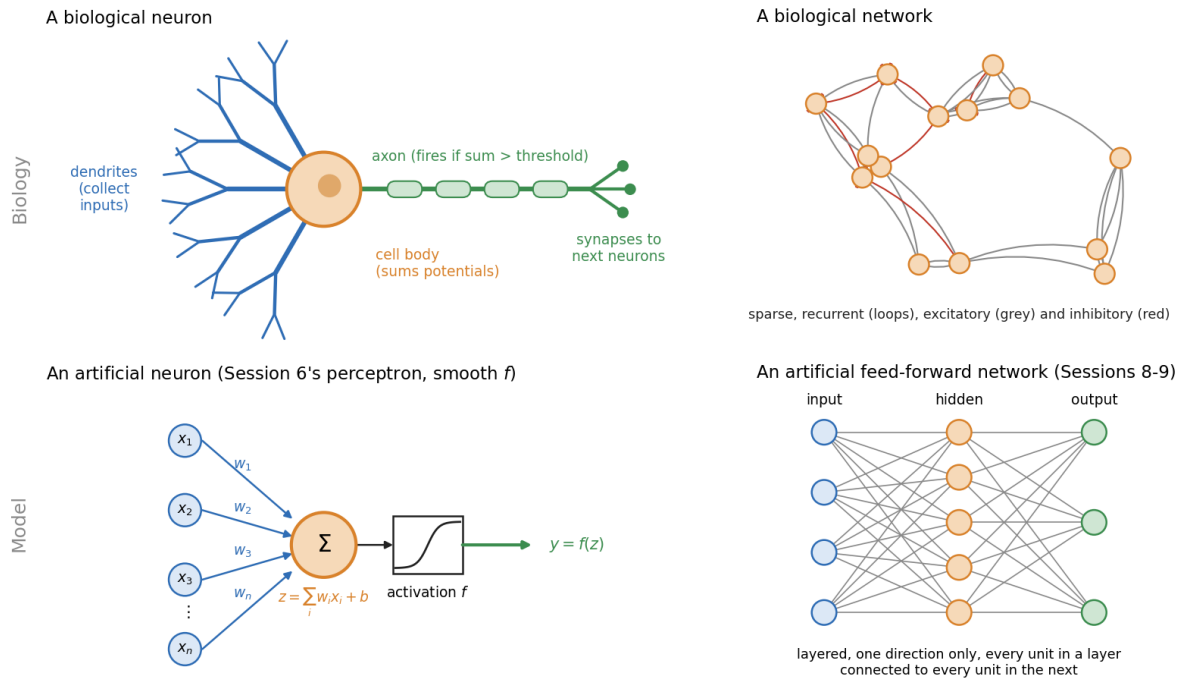


Figure 8.11: Top: a biological neuron and a small biological network, with sparse connections, feedback loops, and both excitatory (grey) and inhibitory (red) synapses. Bottom: the artificial counterparts. A unit computes a weighted sum and applies an activation  $f$ . A feed-forward network arranges units in layers with connections running in one direction only.

- **Real neurons spike.** They send discrete pulses over time. Artificial units output a single number.
- **Brains are recurrent.** Signals loop back. The networks of Sessions 8-10 are strictly feed-forward; Session 11's recurrent networks bring loops back in a controlled form.
- **Brains are sparsely connected.** Each neuron connects to a small fraction of the others.

Each architecture in the rest of the course builds a different assumption about the data into its wiring:

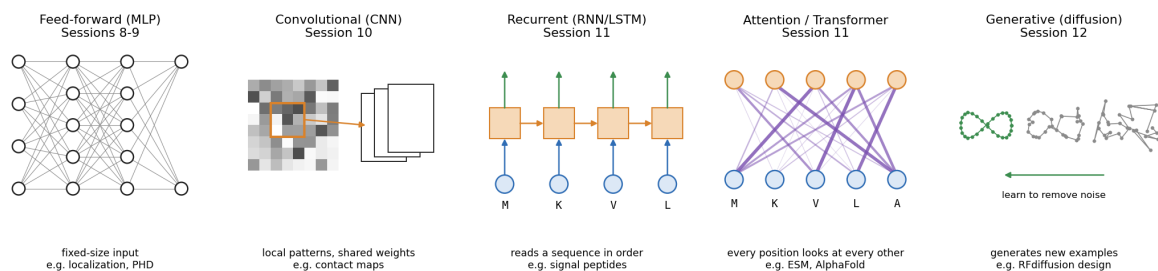


Figure 8.12: The architectures still to come, each with a biological application from this course. Feed-forward networks take a fixed-size input (this session's localisation model; PHD in Session 9). CNNs look for local patterns with shared weights (contact maps, Session 10). RNNs read a sequence one position at a time (signal peptides, Session 11). Transformers let every position attend to every other (language models and SignalP 6.0, Session 11; AlphaFold, Session 12). Generative diffusion models learn to turn noise into new examples (protein design, Session 12).

#### 8.4.2 From step functions to trainable activations

In Session 9 we train networks with backpropagation, which is gradient descent (Session 1) applied layer by layer. For that we need the derivative of every unit's activation function. A step function's derivative is zero everywhere except at the jump, where it is undefined, so it gives gradient descent nothing to follow. Networks therefore replace the step with functions that have a useful gradient. Sigmoid and tanh are smooth. ReLU has a kink at zero, but its slope is 1 or 0 everywhere else, which is enough for gradient descent. You already met the sigmoid in Section 8.2:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad \tanh(z) = 2\sigma(2z) - 1 \quad \text{ReLU}(z) = \max(0, z)$$

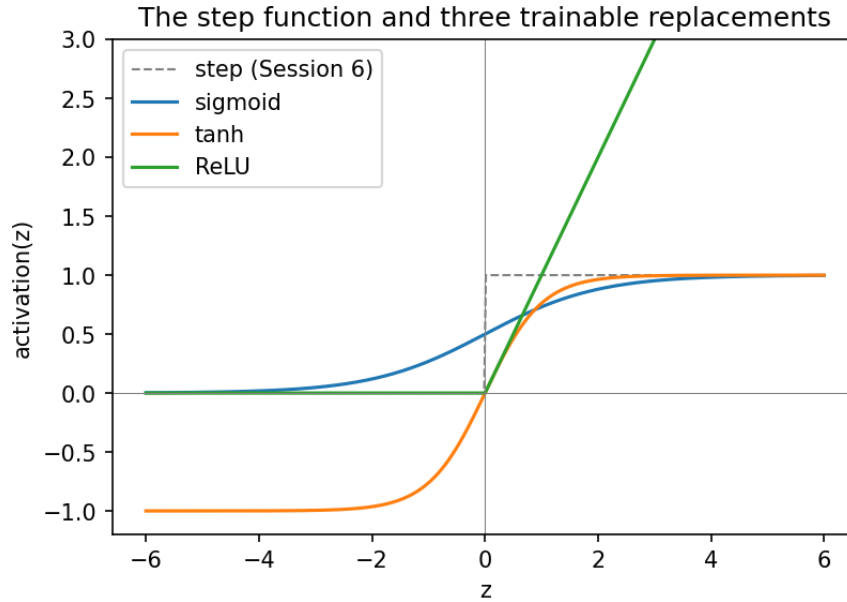


Figure 8.13: The step function of Session 7’s perceptron (dashed) and three replacements that gradient descent can work with.

- **Sigmoid** outputs a value in  $(0, 1)$ . It remains the standard for a binary output unit.
- **tanh** has the same S shape but is centred on zero, with outputs in  $(-1, 1)$ .
- **ReLU** passes positive values unchanged and sets negative ones to zero. It is cheap to compute and is the default for hidden layers in modern networks.

Why the choice matters in deep networks is part of Session 9’s vanishing-gradient discussion, which compares the derivatives of these functions.

### 8.4.3 From binary to multi-class: softmax

With more than two classes, a single threshold is not enough. The  $3 \times 3$  classifier has five outputs and simply picks the largest. **Softmax** turns  $K$  raw scores (**logits**) into  $K$  positive numbers that sum to 1:

$$\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}$$

Take a noisy “horizontal” pattern that the perceptrons misclassify. Its five scores  $(-2.19, -0.80, -2.35, -2.61, 0.16)$  become 0.06, 0.24, 0.05, 0.04 and 0.62, with most of the weight on “cross”. Softmax never changes *which* class wins. It only expresses the relative scores as a normalised distribution. These perceptrons were not trained with cross-entropy, so the numbers sum to 1 but should **not** be read as reliable probabilities. Training with cross-entropy (Session 9) makes softmax outputs interpretable as probability estimates, but whether they are well calibrated depends

on the training and has to be checked on held-out data. For biomedical predictions, where a “90% probability” may drive a decision, that check matters. For two classes, softmax is equivalent to logistic (sigmoid) classification.

Because softmax gives a distribution, its **entropy** (Session 1) measures how spread out the distribution is. It is 1.56 bits for the example above, against a maximum of  $\log_2 5 \approx 2.32$  bits for a model that cannot tell the five classes apart at all, and 0 bits for a completely confident one. Session 9 trains the network by minimising the **cross-entropy** between these values and the true labels.

In PyTorch, a classifier’s last layer outputs raw logits. The loss function (`nn.CrossEntropyLoss`, Session 9) combines log-softmax with the negative log-likelihood in one numerically stable operation, so the model should output raw logits, not softmax values. You apply softmax yourself only when you need normalised scores, for example for a per-class ROC curve.

#### 8.4.3.1 Further reading

- Wikipedia, [Neuron](#), [Neural network \(machine learning\)](#), [Activation function](#), [Softmax function](#)
- d2l.ai, [Softmax Regression](#)

#### 8.4.4 Practical PyTorch: writing a model as a class

A PyTorch model is a class that inherits from `nn.Module`. Its layers are attributes created in `__init__`, and a `forward()` method describes how an input tensor becomes an output tensor. This is the pattern of Session 2’s `Protein` class, with `super().__init__()` and attributes set in `__init__`, now used for a model, and every later model follows it.

The whole  $3 \times 3$  classifier is a single `nn.Linear(9, 5)` layer, with 50 parameters ( $5 \times 9$  weights + 5 biases). When the notebook copies the trained perceptron weights into it, the layer makes exactly the same predictions as the NumPy code, because a perceptron layer and `nn.Linear` do the same computation.

The localisation network is the same pattern at a larger scale: 3000 inputs, a hidden layer of 64 ReLU units, and 5 output logits, for **192,389 parameters**. One forward pass on the homology-aware validation set confirms the shapes ( $105 \times 3000$  in,  $105 \times 5$  out). With random weights it scores what chance looks like: accuracy 0.181, MCC  $-0.027$  and macro AUROC 0.474. The network is **not trained** in this session. Session 9 adds backpropagation and the training loop, and reports the same metrics on this split.

### 8.5 Check your understanding

*The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)*

## 8.6 The course notebook

`notebooks/session08-ml-foundations-and-pytorch.ipynb` computes every number and figure on this page. In order, it covers:

- the helix-fraction regression and the secreted-vs-nuclear logistic regression (both with live data);
- the 3×3 patterns: train/validation/test, confusion matrix, precision/recall/F1/MCC and ROC;
- regression metrics;
- class imbalance, split-to-split noise and cross-validation on the UniProt localisation data;
- MMseqs2 clustering, homology leakage and the homology-aware split. If mmseqs is not installed, as on Colab, a precomputed clustering is downloaded;
- activation functions, softmax, the 3×3 model as an `nn.Linear`, and the untrained localisation network.

Open it in [Google Colab](#) via the badge at the top of the notebook page to edit and run it yourself.

## 8.7 The lab

**Full lab, on blood-cell images.** The lab uses BloodMNIST ([Yang et al. 2023](#)), 17,092 microscope images of single blood cells in eight classes (neutrophils, eosinophils, basophils, lymphocytes, monocytes, immature granulocytes, erythroblasts and platelets). The images were taken with an automated analyser at the Hospital Clínic de Barcelona and labelled by clinical pathologists ([Acevedo et al. 2020](#)), then shrunk to  $28 \times 28$  colour pixels so that a network trains in seconds on a laptop. The lab uses a fixed subset of 7,000 of them. It was adapted from the previous course’s “Neural Networks” lab, which used handwritten digits (MNIST), and rebuilt in PyTorch. Blood cells make the evaluation questions real ones: the classes are unequal in size, and the mistakes the network makes can be explained by cell biology. At the end of the lab, the same machinery is carried straight back to protein localisation. Work through [notebooks/session08-lab.ipynb](#), replacing every `todo(...)` call with your own code:

1. inspect the class balance, and the accuracy you would get by always guessing the commonest cell type;
2. build training, validation and test sets, and standardise the three colour channels using statistics from the training set only;
3. write the model as an `nn.Module` ( $28 \times 28 \times 3 = 2,352$  inputs, 10 tanh hidden units, 8 output logits) and count its parameters;
4. train it with a provided `train_model` function (Session 9 opens that box), then turn logits into probabilities and classes;
5. compute a confusion matrix, and explain its mistakes biologically, then precision, recall, F1, MCC and a ROC curve for immature granulocytes against the rest;

6. see how a rare class (basophils, under 7% of the cells) can almost vanish from the predictions without lowering the accuracy much;
7. shrink the training set to watch overfitting appear, and use the test set once, at the end.

The last section returns to proteins. It measures how many validation proteins in a *random* split of this session's localisation dataset have a homolog in the training set. Together with step 2, it covers two of the three routes for leakage from Section 8.3, namely through the samples (homologs) and through the preprocessing (normalisation statistics).

### 8.7.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## 8.8 What's next

Session 9 trains this network on the homology-aware split. It covers backpropagation, cross-entropy, the training loop, vanishing gradients (including the activation-function derivatives deferred from this session) and regularisation. It closes with Rost & Sander's PHD method as the payoff of the Session 5-9 arc.

## 9 Session 9: How a Neural Network Learns, and Why Training Fails

### Note

**Sources for this page:** [d2l.ai](https://d2l.ai), *Dive into Deep Learning*, [Multilayer Perceptrons](#) — CC BY-SA 4.0; Wikipedia, [Backpropagation](#), [Stochastic gradient descent](#), [Overfitting](#), [Vanishing gradient problem](#), [Regularization \(mathematics\)](#), [Dropout \(neural networks\)](#), [Early stopping](#) — CC BY-SA 4.0. Datasets: the [Breast Cancer Wisconsin \(Diagnostic\)](#) data set (UCI Machine Learning Repository, CC BY 4.0), loaded through [scikit-learn](#); the GB1 fitness landscape of Wu, Dai, Olson, Lloyd-Smith & Sun (2016), [eLife 5:e16965](#) (CC BY 4.0), as distributed by the FLIP benchmark (Dallago et al. 2021, [bioRxiv](#)); Session 8’s UniProt subcellular-localization data. Ioffe & Szegedy (2015), “Batch Normalization,” [arXiv:1502.03167](#); Srivastava et al. (2014), “Dropout,” [JMLR 15:1929-1958](#); Kingma & Ba (2015), “Adam: A Method for Stochastic Optimization,” [arXiv:1412.6980](#); Rost & Sander (1993), [J. Mol. Biol. 232:584-599](#) and [PNAS 90:7558-7562](#) (the PHD method). 3Blue1Brown’s neural-network and cross-entropy videos are embedded as further watching. The regression setup (linear baseline vs. 64/64 network, L2 and dropout), the vanishing-gradient demonstration and the tiny hand-computed backpropagation example follow ideas from Magnus Ekman’s *Learning Deep Learning* (ch. 3, 5 and 6 slide decks in slides/), a copyrighted, not openly licensed textbook, credited here as an internal source and not linked. Every number below was computed in code and read back from the notebook’s output.

### Generalisation

I have a machine learning joke, but it doesn’t perform as well on a new audience.

### 9.1 Learning goals

By the end of this session you should be able to:

- describe the training cycle (predictions → loss → gradients → update) and write the standard PyTorch training loop;
- explain backpropagation as the chain rule applied from the loss backwards through the network;

- choose the loss for classification (cross-entropy) and regression (mean squared error), and explain why confident mistakes are expensive;
- recognise three common ways training fails (equal starting weights, vanishing gradients, a poor optimiser or learning rate) and name the fix for each;
- judge when a hidden layer pays off, and why a very large network can lose to a simple baseline;
- use early stopping, weight decay and dropout against overfitting, and evaluate the result honestly.

Sessions 7, 8 and 9 form one progression. Session 7 asked **what a neural network can represent**. Session 8 asked **how we know whether a model generalises**. This session asks **how we actually train a network, and why training sometimes succeeds and sometimes fails**.

The starting point is concrete. In Session 8 we built a network, but its weights were random, so its predictions were no better than chance. We had labelled examples, so how do we turn the random weights into useful predictions? This session answers that question with four datasets. Breast-cancer biopsies introduce the training loop, and a protein fitness landscape shows why training fails and when a hidden layer pays off. Session 8's protein localisation data show what happens when a network is too large for its data, and blood-cell images are used in the lab.

## 9.2 From an untrained network to a trained network

We start with a problem on which training works well. The **Wisconsin breast-cancer** dataset has 569 fine-needle biopsies, each described by 30 features measured on microscope images of the cell nuclei (radius, texture, perimeter, smoothness, concavity and so on). Each biopsy is labelled malignant (212) or benign (357). We split the data into 398 training, 85 validation and 86 test biopsies, and standardise the features with statistics from the training set only (Session 8). The network has one hidden layer of 16 ReLU units and one output, 513 weights in all.

Training the network means repeating one cycle over and over:

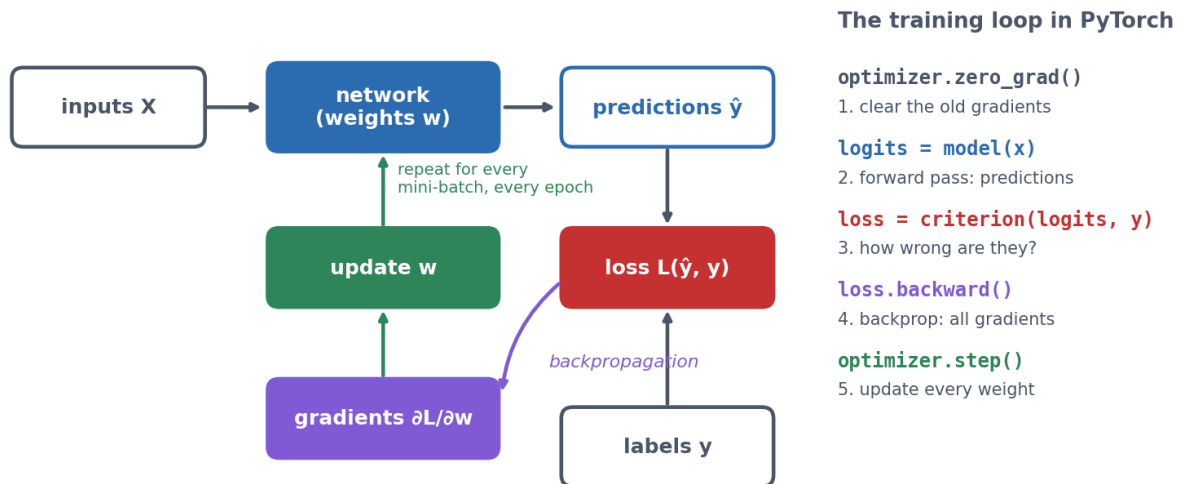


Figure 9.1: The training cycle. The network turns inputs into predictions; the loss measures how wrong they are compared with the labels; backpropagation computes the gradient of the loss with respect to every weight; the optimiser updates the weights. The same five lines of PyTorch (right) do this for every mini-batch, in every epoch.

Before training, the network’s predicted probabilities all sit close to 0.5, so the network is guessing (validation accuracy 0.659, roughly the share of benign biopsies). After 100 passes through the training data, the benign biopsies are pushed towards 0 and the malignant ones towards 1:

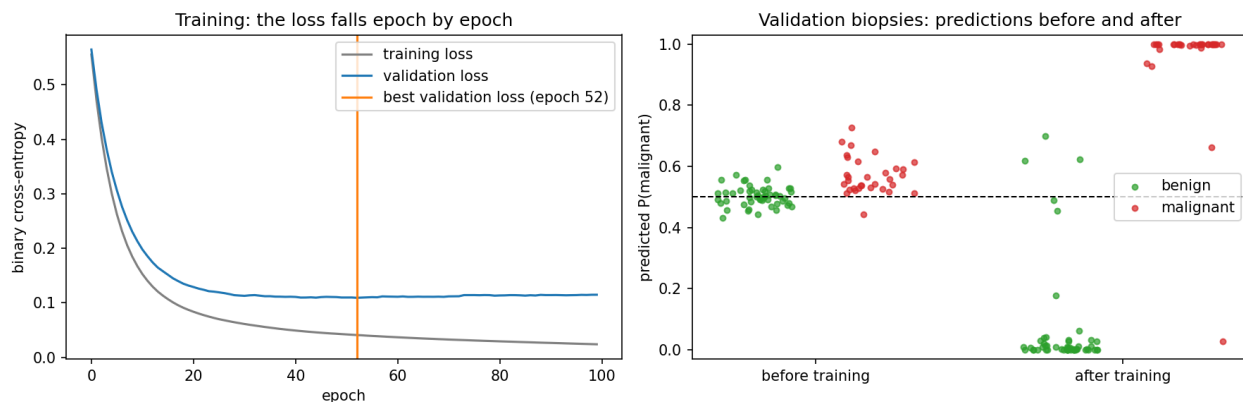


Figure 9.2: Left: training and validation loss per epoch. Both fall quickly; after about 50 epochs the validation loss stops improving while the training loss keeps falling, and we keep the weights from the epoch with the lowest validation loss (early stopping). Right: predicted probability of malignancy for the 85 validation biopsies, before and after training.

On the test set the trained network reaches an accuracy of 0.953, an MCC of 0.900 and an AUROC of 0.984. Only 2 benign biopsies are called malignant, and only 2 malignant

ones are missed.

**Was the hidden layer needed?** Not here. We compare the models by 5-fold cross-validation, because a single test split of 86 biopsies is too noisy for this comparison (Session 8). Logistic regression with 31 weights and the network with 513 both reach an accuracy of **0.979**. Malignant and benign nuclei differ mainly in size and shape, and a weighted sum of those measurements already separates them. Training therefore works, but the extra capacity has nothing to add. Section 9.6 shows a dataset where it does.

### 9.3 How backpropagation finds every gradient

Gradient descent (Session 1, Session 7) needs  $\partial L / \partial w$ , the slope of the loss, for **every** weight. For a single layer that was easy. But what about a weight buried several layers inside a network, whose effect on the loss passes through every later layer? The tiny network below shows how backpropagation answers this question.

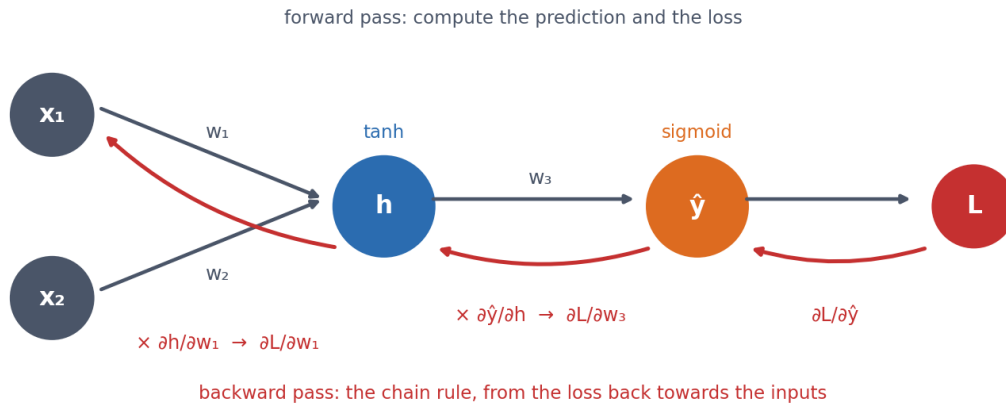


Figure 9.3: A tiny network: two inputs, one tanh hidden unit  $h$ , one sigmoid output  $\hat{y}$ , and the loss  $L$ . The forward pass (grey) computes the prediction and the loss. The backward pass (red) applies the chain rule from the loss back towards the inputs:  $\partial L / \partial \hat{y}$  is computed once and reused for  $w_3$ , then multiplied by  $\partial \hat{y} / \partial h$  and reused for  $w_1$  and  $w_2$ .

For  $w_1$ , the chain rule multiplies the local slopes along the path from the loss back to the weight:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_1}$$

**Backpropagation is an efficient application of the chain rule, from the loss backwards through the network.** Each layer’s “error signal” is computed once and reused for all the weights feeding into it, so computing all the gradients costs about as

much as one forward pass. Without that reuse, networks with thousands or millions of weights could not be trained.

In practice, you do not have to differentiate networks by hand. The notebook does it once for the tiny network above, working out the five gradients with the chain rule. It then builds the same network in PyTorch and calls `loss.backward()`, and all five gradients agree to eight decimal places. From then on, PyTorch's **autograd** does the work for any network.

Putting these pieces together gives the **training loop** that you will write yourself in the lab:

```
for epoch in range(n_epochs):
    for xb, yb in loader:                # one mini-batch at a time
        optimizer.zero_grad()           # 1. clear the old gradients
        logits = model(xb)              # 2. forward pass
        loss = criterion(logits, yb)     # 3. how wrong is it?
        loss.backward()                 # 4. backpropagation: every gradient
        optimizer.step()                # 5. update every weight
```

Real training uses **mini-batches**. A `DataLoader` shuffles the training set every epoch and serves it in small batches (here 32 biopsies), and the weights are updated after every batch. Each **epoch** (one pass through the training data) therefore consists of many cheap, slightly noisy steps instead of one expensive exact step. This is **stochastic gradient descent**. After every epoch we measure the validation loss and keep the best weights (**early stopping**).

### 9.3.1 Further watching

*Backpropagation, intuitively* (3Blue1Brown, Deep Learning ch. 3): the same chain-rule idea as above, built up visually one layer at a time. (Chapter 1, *But what is a neural network?*, is embedded on Session 7.)

Video: [Backpropagation, intuitively, 3Blue1Brown](#)

**Optional video:** *Backpropagation calculus* (3Blue1Brown, Deep Learning ch. 4, 10 min), the full chain-rule mechanics for a general multi-layer network.

## 9.4 Which loss to minimise

The training loop minimises a loss, and the loss decides what “wrong” means. For **regression** we use the **mean squared error** between prediction and target. For **classification** we use **cross-entropy**, which for one example is the negative log of the probability that the model gave the true class,

$$\text{CE} = -\log p(\text{true class}),$$

that is, its *surprise* in the language of Session 1. Averaged over a dataset, cross-entropy is the negative log-likelihood, so minimising it is the same as maximum likelihood (Session 1).

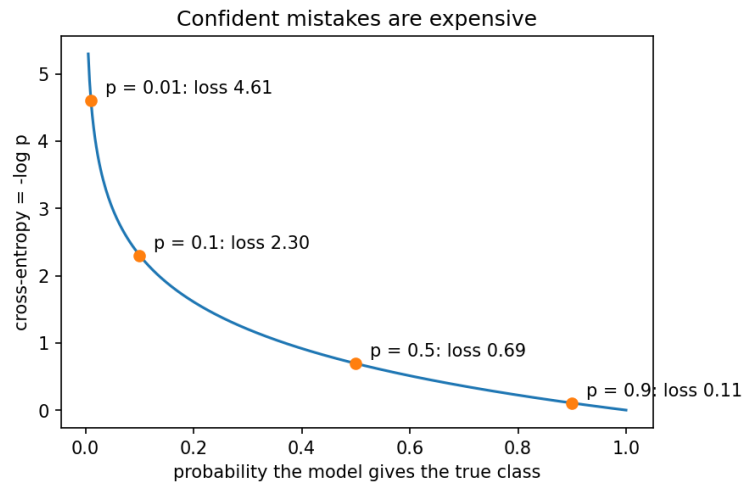


Figure 9.4: Cross-entropy as a function of the probability given to the true class. A confident correct answer costs almost nothing; a confident mistake costs a lot.

Cross-entropy punishes a confident mistake far more than hedging. A probability of 0.9 on the true class costs 0.105, but a probability of 0.01 costs 4.6, over 40 times more. This is what pushes a classifier to separate the classes, as in the right panel of the breast-cancer figure. In PyTorch, `nn.BCEWithLogitsLoss` (for two classes) and `nn.CrossEntropyLoss` (for more classes) combine the sigmoid or log-softmax with this loss in one numerically stable step, so the model outputs raw logits (Session 8). The notebook checks both by hand. The lab compares cross-entropy with the mean squared and mean absolute error for the same blood-cell classifier, and cross-entropy trains best.

#### **i** Optional: cross-entropy and relative entropy

Cross-entropy is built from the relative entropy (Kullback-Leibler divergence) of Session 4. It is the entropy of the true distribution plus the extra cost of describing it with the model's distribution  $Q$ :  $H(P, Q) = H(P) + D_{\text{KL}}(P \| Q)$ . Since  $H(P)$  does not depend on the model, minimising cross-entropy is the same as minimising the KL divergence between the model's predictions and the true labels.

**Optional video:** *But what is cross-entropy?* (3Blue1Brown, Compression is Intelligence part 2, 34 min), cross-entropy from the information-theory side, built up from first principles.

## 9.5 Why training fails

The training loop always runs, but it does not always learn. This section shows three common failures. Each is produced by changing one thing in a network that predicts the fitness of protein variants, using the GB1 data of Section 9.6 (20,000 training variants, each described by the one-hot encoding of four residues). We measure the validation error as the mean absolute error (MAE) of the predicted  $\log_{10}$  fitness. As reference points, predicting the training mean for every variant gives an MAE of 0.488, and a working 64/64 network reaches 0.250 within 20 epochs.

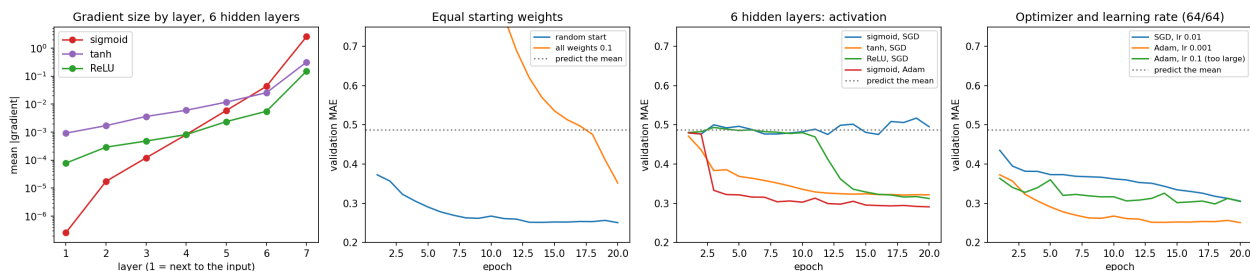


Figure 9.5: Four experiments. (1) Gradient size per layer in a freshly initialised network with six hidden layers: with sigmoid, the first layer receives about ten million times less gradient than the last. (2) Starting every weight at the same value. (3) Six hidden layers with sigmoid, tanh or ReLU, trained with plain gradient descent (SGD), and sigmoid trained with Adam. (4) Optimiser and learning rate. The dotted line is the error of predicting the mean.

### 9.5.1 Failure 1: every weight starts the same

- **Observation.** Start all weights at 0.1 instead of small random numbers. After 20 epochs the validation MAE is 0.352, little better than the additive (linear) model's 0.371, against 0.250 from a random start.
- **Why?** Every hidden unit computes the same thing and receives the same gradient, so the units stay identical for ever. After training, all 64 units in the first layer are still exactly the same, so 64 units behave like one.
- **Fix.** Random initialisation *breaks the symmetry*, and PyTorch does this by default.

### 9.5.2 Failure 2: deep sigmoid networks, and vanishing gradients

- **Observation.** A network with six hidden layers of sigmoid units, trained with plain SGD, does not learn at all. Its validation MAE stays around 0.48, the error of predicting the mean. The same network with tanh (0.321) or ReLU (0.312) does learn.
- **Why?** Backpropagation multiplies one activation *derivative* per layer, and the derivative of the sigmoid is at most 0.25. Even in the best case, six layers therefore multiply the gradient by  $0.25^6 \approx 0.00024$ . Measured in the freshly initialised

network, the first layer's gradient is about **ten million times smaller** than the last layer's (for tanh and ReLU, a few hundred to two thousand times). The first layers barely move.

- **Fix.** Change what the gradient has to pass through. The options are ReLU-like activations, initialisation scaled to each layer's size, standardised inputs, normalisation layers such as batch normalisation (Ioffe & Szegedy, 2015), and, in later sessions, residual connections. Adam (below) gets the sigmoid network to 0.291 by enlarging the tiny steps, but that is a workaround, not a cure, because the gradients still vanish.

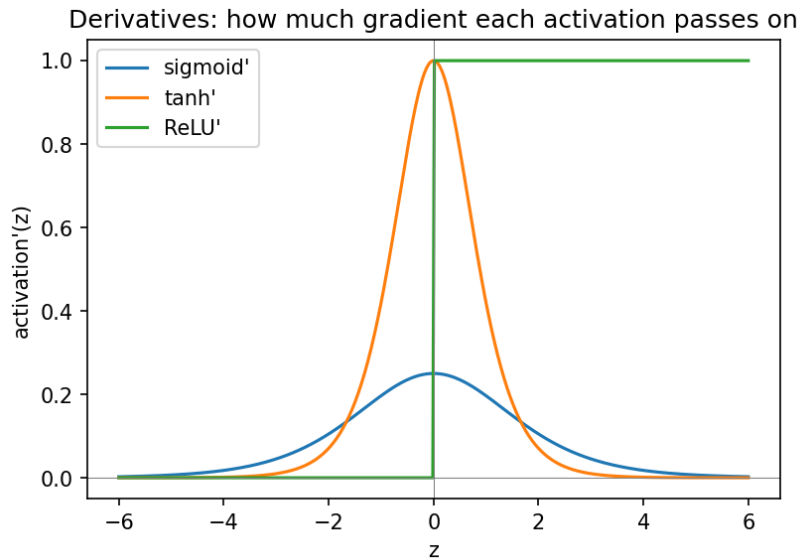


Figure 9.6: Derivatives of sigmoid, tanh and ReLU. Sigmoid's is at most 0.25 (at  $z = 0$ ) and below 0.01 by  $|z| = 6$ . Tanh's peaks at 1 but also vanishes in both tails. ReLU's is exactly 0 or exactly 1.

ReLU (Session 8) never shrinks the gradient for positive inputs, which is why it is the default for hidden layers. Its flat side, however, can leave a unit with no gradient at all (“dying ReLU”, the first discussion problem). Vanishing gradients come back in very deep CNNs (Session 10), recurrent networks and Transformers (Session 11), and each of those sessions shows the architectural fix.

### 9.5.3 Failure 3: the optimiser and the learning rate

- **Observation.** The same 64/64 network reaches a validation MAE of 0.250 with Adam (learning rate 0.001), but only 0.305 with plain SGD in the same 20 epochs. With Adam and a learning rate of 0.1, the error jumps around and never gets below 0.298.
- **Why?** The learning rate sets the step size of gradient descent (Session 1). A rate that is too small is slow, and one that is too large overshoots the minimum. Plain SGD also uses the same step size for every weight, which Adam does not (next subsection).

- **Fix.** Start with Adam and a learning rate around  $10^{-3}$ , and always watch the validation curve.

### 9.5.4 Momentum and Adam: why an optimiser can beat plain SGD

Plain SGD takes each step straight down the gradient of the current mini-batch. This becomes a problem in a long, narrow valley of the loss, which is common because some weights affect the loss far more than others. In such a valley the gradient points mostly *across* the valley. A learning rate large enough to make progress *along* the valley makes SGD zigzag from wall to wall, while a safe one only crawls. The three optimisers differ in how they deal with this:

- **SGD** follows the current gradient.
- **Momentum** follows an average of the recent gradients, like a heavy ball rolling downhill. Directions that agree from step to step add up, and the zigzag across the valley cancels out.
- **Adam** (Kingma & Ba, 2015) combines momentum with a separate step size for every weight, scaled to the size of that weight's recent gradients. Its default learning rate of  $10^{-3}$  works for many networks.

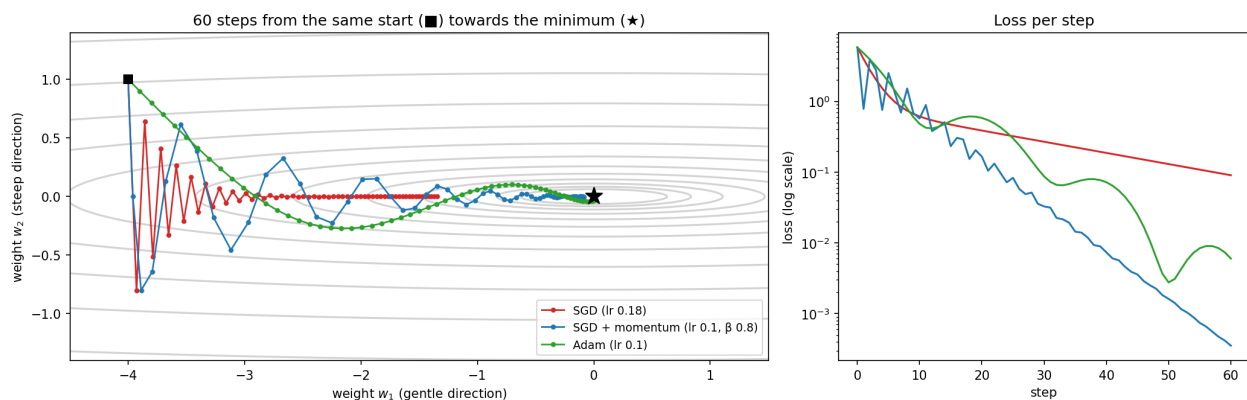


Figure 9.7: Three optimisers, 60 steps each, on a valley 100 times steeper across ( $w_2$ ) than along ( $w_1$ ). SGD zigzags and then crawls (loss 0.09 after 60 steps); momentum (0.0004) and Adam (0.006) reach the minimum.

Adam is a good default, not a universal winner. In the figure, tuned momentum ends even lower, and which optimiser wins depends on the learning rate and the training time. The lab repeats all three failures on blood-cell images, including this comparison, with one rule: *change one thing, see what happens, explain why.*

## 9.6 When a hidden layer pays off: epistasis in a protein

**The data.** GB1 is a small domain of protein G, a bacterial protein that binds antibodies (IgG). Wu et al. (2016) made all  $20^4 = 160,000$  combinations of amino acids at four of

its positions (V39, D40, G41 and V54), and measured by deep sequencing how well each variant binds IgG. Of these, 149,361 variants have a value. Fitness is relative to the wild type ( $\text{VDGV} = 1$ ) and spans several orders of magnitude, so we predict  $\log_{10}(\text{fitness} + 0.001)$ . Each variant is described by the **one-hot encoding** of its four residues (Session 7), that is, by  $4 \times 20 = 80$  numbers.

**Additive or not?** A linear model on this encoding gives every amino acid at every position its own weight and adds them up. It is therefore an **additive** model, exactly like a profile or PSSM (Session 5), in which each position contributes on its own. Such a model cannot represent **epistasis**, where the effect of a mutation depends on which residues sit at the other positions. Epistasis is the non-additive, XOR-like pattern of Session 7, now as a measured property of a real protein, and a hidden layer can represent it.

We train the models with early stopping on 20,000 variants, with 5,000 for validation and 20,000 for testing:

Table 9.1: Models trained on the GB1 fitness data.

| Model                              | Weights | Train MAE | Val MAE | Test MAE | <b>Test <math>R^2</math></b> |
|------------------------------------|---------|-----------|---------|----------|------------------------------|
| Linear (additive)                  | 81      | 0.379     | 0.371   | 0.385    | <b>0.496</b>                 |
| 64/64                              | 9,409   | 0.212     | 0.250   | 0.254    | <b>0.787</b>                 |
| 64/64 + weight decay ( $10^{-3}$ ) | 9,409   | 0.215     | 0.241   | 0.249    | 0.795                        |
| 64/64 + dropout 0.2                | 9,409   | 0.220     | 0.242   | 0.247    | 0.798                        |
| 256/256                            | 86,785  | 0.193     | 0.252   | 0.259    | 0.779                        |

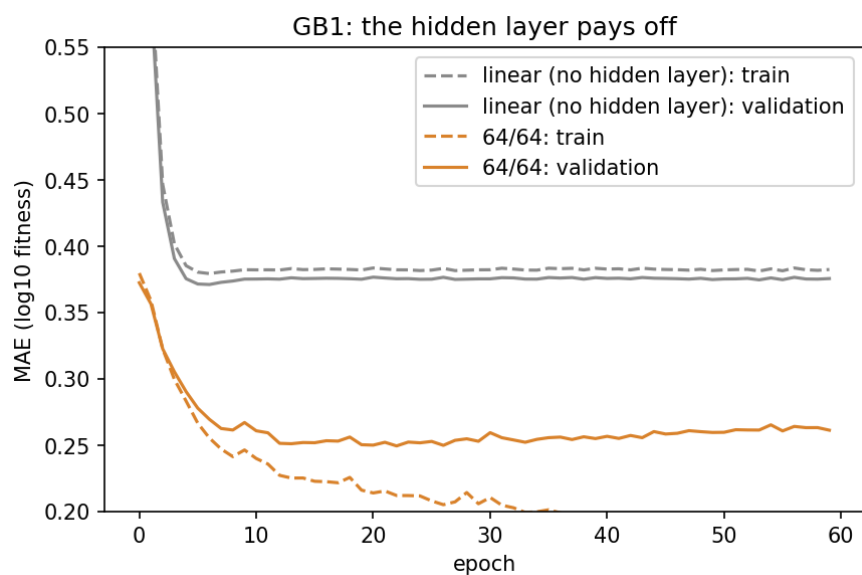


Figure 9.8: Training (dashed) and validation (solid) MAE per epoch, for the additive model and the 64/64 network.

The network explains about 79% of the variance in fitness, against 50% for the additive model. Weight decay and dropout help a little, and a network ten times larger does not help further.

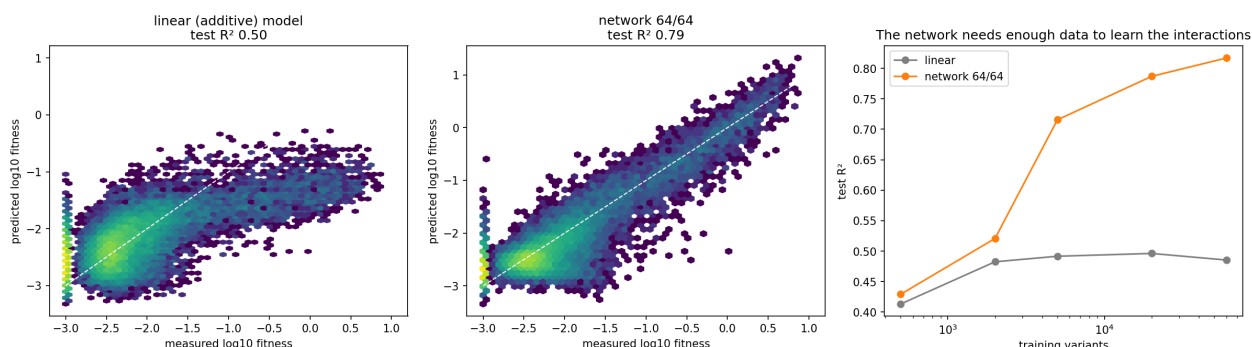


Figure 9.9: Left and middle: predicted against measured  $\log_{10}$  fitness for the 20,000 test variants (colour: how many variants fall in each hexagon). The additive model compresses the predictions into a narrow band and misses the fit variants; the network follows the diagonal. Right: test  $R^2$  against the number of training variants.

**The network needs data to learn the interactions.** With 500 training variants the two models are equally good ( $R^2$  0.43 against 0.41). The interactions exist, but there are too few examples to learn them. From a few thousand variants on, the network pulls away (0.72 at 5,000, 0.82 at 60,000), while the additive model is stuck at about 0.49, however much data it gets. No amount of data lets a sum represent an interaction.

This is Session 7's lesson, now on real biological data. A hidden layer can represent non-additive relationships, and with enough data, training actually finds them. It is also the limitation of profiles noted in Session 5, since a sum over positions cannot capture interactions, and Session 12 returns to it with pairwise models learned from multiple alignments. The second discussion problem asks where the advantage comes from, by giving the linear model hand-made interaction features for each pair of positions, as Session 7 did for XOR with  $x_1x_2$ .

## 9.7 Why not make the network enormous?

If larger networks can represent more, why not always use an enormous one? Session 8's protein-localisation network answers that question. It has 3000 inputs (150 residues, one-hot), a hidden layer of 64 units and 5 outputs, **192,389 weights** in all, and it is trained on only **486 proteins** of the homology-aware split.

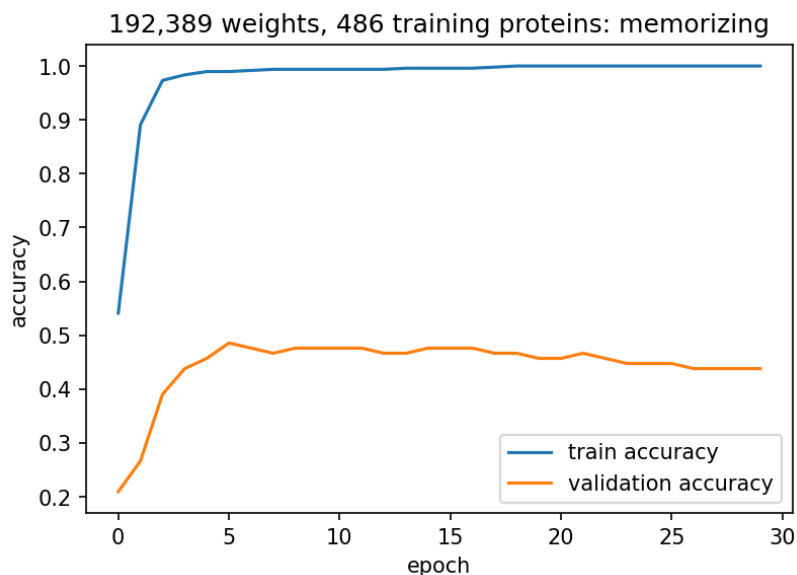


Figure 9.10: Training and validation accuracy of the localisation network per epoch. Training accuracy reaches 1.0 within a few epochs; validation accuracy peaks at 0.486 (epoch 5) and then falls.

Training accuracy reaches **1.000**, so the network memorises every training protein. Validation accuracy, in contrast, peaks at **0.486** after five epochs and then *falls*, to 0.438 by epoch 29. This is **overfitting** in its plainest form. With far more freedom than data, the network learns the idiosyncrasies of the training sequences instead of the general signal.

Moreover, the simple baseline wins, as the test set (used once) shows:

Table 9.2: The localisation classifier: the untrained network of Session 8, the trained network and logistic regression.

| Metric      | Untrained network<br>(Session 8, val) | Trained network<br>(test) | Logistic regression<br>(test) |
|-------------|---------------------------------------|---------------------------|-------------------------------|
| Accuracy    | 0.181                                 | 0.514                     | <b>0.569</b>                  |
| Macro F1    | 0.124                                 | 0.493                     | —                             |
| MCC         | −0.027                                | 0.394                     | <b>0.461</b>                  |
| Macro AUROC | 0.474                                 | 0.831                     | <b>0.849</b>                  |

Training works, since every number is far above chance, but **the linear model is better**. Before believing that a bigger model is better, always compare it against a simple baseline.

## 9.8 What can we do about overfitting?

There are several remedies:

- **More data.** The most reliable fix, and often the only one that really helps.
- **A smaller model, or a better architecture.** Fewer weights for the same input. Session 10's convolutional network needs 19,237 weights instead of 192,389.
- **Early stopping.** Keep the weights from the epoch with the best validation score ([below](#)).
- **Weight decay (L2) and L1 regularisation.** Add a penalty for large weights to the loss ([below](#)).
- **Dropout.** Randomly switch off a fraction of the units in every training step, so the network cannot rely on any single unit ([below](#)).

Applied to the localisation network, however, dropout and weight decay do *not* beat early stopping alone (best validation accuracy 0.476 against 0.486). Training accuracy still reaches 1.000 whatever we do, because with 192,389 weights and 486 examples the network can always memorise. Regularisation is therefore not a substitute for enough data. The following subsections explain how the remedies work.

### 9.8.1 Early stopping: when to stop training

The two curves of a training run answer two different questions:

- **The training loss tells you whether optimisation is working.** If it does not fall, something in the previous sections is wrong (initialisation, gradients, learning rate).
- **The validation loss tells you whether further optimisation still helps generalisation.** When it stops falling while the training loss keeps going down, the network has started to fit the training set rather than the problem.

For the breast-cancer biopsies the validation loss levels off after about 50 epochs while the training loss keeps falling (the first figure of this page). For protein localisation, training accuracy reaches 1.000 within a few epochs, while validation accuracy peaks at 0.486 in epoch 5 and then falls. **Early stopping** keeps the weights from the epoch with the best validation score. It costs nothing and is used in every training loop of this course. Because the epoch is chosen on the validation set, however, only the test set, used once, gives an honest estimate (Session 8).

### 9.8.2 Penalising large weights: L2 and L1

A network that fits noise usually does it with some very large weights, because a big weight makes the output jump when one input changes a little. Regularisation therefore adds a penalty for large weights to the loss, with a strength  $\lambda$  that we choose on the validation set:

$$\text{loss} = \text{cross-entropy} + \lambda \sum_j w_j^2 \quad (\text{L2}) \quad \text{or} \quad \text{cross-entropy} + \lambda \sum_j |w_j| \quad (\text{L1})$$

- **L2** pulls every weight towards zero in proportion to its size (the gradient of  $\lambda w^2$  is  $2\lambda w$ ). Every step therefore shrinks every weight by the same *fraction*, which is why it is called **weight decay** and why PyTorch implements it as the optimiser's `weight_decay` argument. Weights become small, but almost never exactly zero.
- **L1** pulls with the same force,  $\lambda$ , however small a weight already is. Weights that help little are pushed all the way to **exactly zero**, so the model *selects features*.

For linear regression the two have their own names, **ridge** (L2) and **lasso** (L1). The notebook fits both to the 30 features of the breast-cancer biopsies of Section 9.2. It uses logistic regression, so that every feature has exactly one weight:

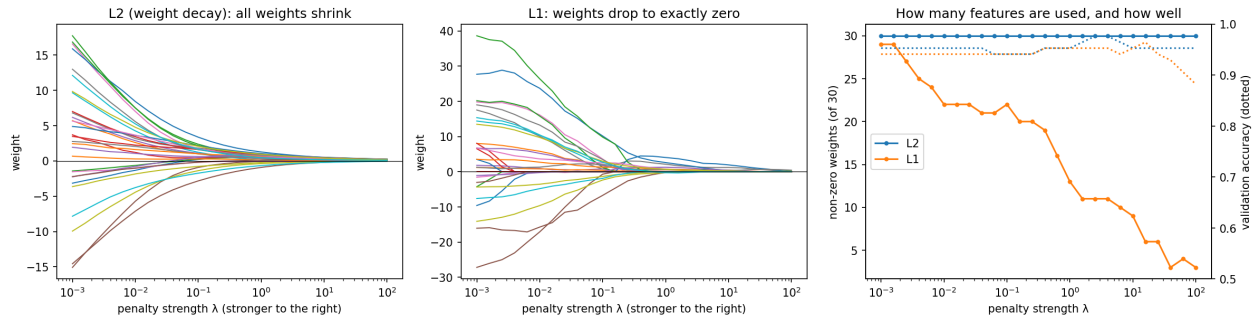


Figure 9.11: Logistic regression on the 30 biopsy features. Left and middle: every line is one feature's weight as the penalty gets stronger (to the right). Right: the number of non-zero weights (solid) and the validation accuracy (dotted).

With  $\lambda$  chosen on the validation set, L2 keeps all 30 weights (test accuracy 0.965), while L1 keeps only **6** (test accuracy 0.942). The six are led by the largest tumour radius and the number of concave points on the cell nuclei. L1 thus gives almost the same accuracy with a model that a pathologist can read. When features are many and examples few, as in gene-expression data with 20,000 genes and a few hundred patients, this is often the main reason to use L1. The third discussion problem checks the same effect by hand.

### 9.8.3 Dropout

**Dropout** (Srivastava et al. 2014, [JMLR 15:1929-1958](#)) switches off each hidden unit with probability  $p$  (often 0.2-0.5), with a fresh random choice for every mini-batch.



Figure 9.12: Dropout with  $p = 0.5$ . In every training step a different random set of hidden units is switched off. At test time every unit is on.

Why should damaging the network help? The answer is that **no unit can count on its partners**. Without dropout, units can form fragile combinations that fit the quirks of the training set. With dropout, any partner may be missing in the next step, so the information is spread over many units.

**Training mode and evaluation mode.** Dropout is used only for training, and at evaluation time all units are on. (PyTorch scales the surviving units by  $1/(1-p)$  during training, so nothing else needs to change.) `model.train()` switches dropout on and `model.eval()` switches it off, which is why every training loop in this course calls both. If you forget `model.eval()`, the same input gives a different prediction every time, as the notebook shows.

On GB1, dropout 0.2 raises the training error a little (MAE  $0.212 \rightarrow 0.220$ ) and lowers the test error a little ( $0.254 \rightarrow 0.247$ ). The gap narrows, as it should with a regulariser. On the localisation network, dropout does not beat early stopping.

## 9.9 Is the improvement real?

Session 8's rules apply to every comparison in this session. Choices (epochs, sizes, regularisation) are made on the validation set, the test set is used once, and every model is compared with a simple baseline. The table puts the three datasets side by side:

Table 9.3: The three datasets of this session side by side.

| Dataset       | Training examples | Network weights | Examples per weight | Network vs. linear                |
|---------------|-------------------|-----------------|---------------------|-----------------------------------|
| Breast cancer | 398               | 513             | 0.78                | tie (CV accuracy 0.979 vs. 0.979) |

| Dataset              | Training examples | Network weights | Examples per weight | Network vs. linear                            |
|----------------------|-------------------|-----------------|---------------------|-----------------------------------------------|
| GB1 fitness          | 20,000            | 9,409           | 2.1                 | <b>network</b> (test $R^2$ 0.787 vs. 0.496)   |
| Protein localisation | 486               | 192,389         | 0.003               | <b>linear</b> (test accuracy 0.514 vs. 0.569) |

- **Breast cancer:** the network trains well, but a linear model is just as good, because the problem is nearly additive.
- **GB1 fitness:** the network clearly wins, because the relationship is non-additive (epistasis) and there are enough data to learn it.
- **Protein localisation:** the network memorises and loses to the baseline, because the data are scarce relative to the model.

Examples per weight is a **warning sign, not a strict rule**. Modern networks often have more weights than training examples and still generalise, thanks to their architecture, regularisation and the way they are trained. When a model is huge relative to its data, however, check it against a baseline before trusting it.

Now you can read the whole PHD pipeline

Rost & Sander's PHD (1993, Session 7) is the recipe of Sessions 5-9 at the scale that mattered historically. It combined **profile input** from a multiple alignment (Session 5) with networks that have a **hidden layer** (Session 7) and **three outputs** (Session 8). The networks were trained by **backpropagation** (this session) and evaluated by cross-validation on test sets purged of similar sequences (Session 8's homology partitioning). PHD reached 70.8% three-state accuracy.

**Next: fewer weights, by using the structure of the data.** A fully connected layer has a separate weight for every connection, which is why the localisation network needs 192,389 of them. A sequence, however, has structure, because the same motif means the same thing wherever it occurs. Can we exploit that and use far fewer weights? That is **convolution**, the topic of Session 10.

### 9.9.1 Further reading

- Wikipedia, [Backpropagation](#), [Stochastic gradient descent](#) (including momentum and Adam), [Vanishing gradient problem](#), [Overfitting](#), [Regularization \(mathematics\)](#), [Dropout \(neural networks\)](#), [Early stopping](#)
- d2l.ai, [Multilayer Perceptrons](#)
- Ioffe & Szegedy (2015), [Batch Normalization](#)
- Kingma & Ba (2015), [Adam: A Method for Stochastic Optimization](#); d2l.ai, [Optimization Algorithms](#) (momentum, Adam and others, with interactive examples)

- Rost & Sander (1993), [Prediction of protein secondary structure at better than 70% accuracy](#), *J. Mol. Biol.* 232:584-599
- UCI Machine Learning Repository, [Breast Cancer Wisconsin \(Diagnostic\)](#)

## 9.10 Check your understanding

The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)

## 9.11 The course notebook

`notebooks/session09-backprop-training-regularization.ipynb` computes every number and figure on this page, section by section. It runs the breast-cancer training and shows the predictions before and after training. It works the backpropagation example by hand and checks it against autograd, and it computes cross-entropy by hand. It then reproduces the three training failures (equal initialisation, deep sigmoid networks, optimiser and learning rate), the GB1 models and the effect of the training-set size, and finally the localisation network's overfitting, regularisation and comparison with logistic regression.

Open it in [Google Colab](#) via the badge at the top of the notebook page to edit and run it yourself.

## 9.12 The lab

**Full lab, on blood-cell images** (BloodMNIST, as in the Session 8 lab). The lab is adapted from the previous course's "Neural Networks" lab and rebuilt in PyTorch, and this time you **write the training loop yourself**. Work through [notebooks/session09-lab.ipynb](#), replacing every `todo(...)` call with your own code:

1. write the four-step mini-batch training loop;
2. compare cross-entropy with mean squared and mean absolute error as the loss for a classifier;
3. add hidden layers (tanh, sigmoid, ReLU) and measure the gradient that reaches the first layer;
4. compare plain SGD, SGD with momentum and Adam, a 128-unit hidden layer with no hidden layer at all, and run a small hyperparameter search chosen on the validation set, using the test set once, at the end.

The best dense network ends up near 78% on the test images. An optional extra at the end of the notebook, not part of the quiz, starts every weight at 1 and shows why random initialisation matters. Session 10's lab trains a convolutional network on the same cells.

### 9.12.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## 9.13 What's next

Session 10 moves from fully connected networks to **convolutional** ones, which share weights across positions. It covers image classification and the history of ImageNet, data augmentation and transfer learning (on colorectal-cancer histology images), and the localisation dataset again, now with 10 times fewer weights. It then turns to DNA sequence (DeepSEA) and protein contact prediction.

# 10 Session 10: CNNs, and How They Took Over Biology

## Note

**Sources for this page:** [d2l.ai](https://d2l.ai), *Dive into Deep Learning*, [Convolutional Neural Networks](#) and [Modern Convolutional Neural Networks](#) — CC BY-SA 4.0. Every historical number on this page is taken from the primary paper or the official challenge report, cited where it is used (ImageNet: Russakovsky et al. 2015, [IJCV 115:211-252](#)). Lab results quoted on the page come from the executed solution of the Session 10 lab (BloodMNIST, Yang et al. 2023, CC BY 4.0). Data: colorectal-cancer histology tiles NCT-CRC-HE-100K and CRC-VAL-HE-7K (Kather, Halama & Marx 2018, [Zenodo](#), CC BY 4.0; a small subset is kept in the course repository), Sessions 8-9's UniProt localization dataset, and PDB entry [4HHB](#) (hemoglobin, CC0). The ImageNet-trained ResNet-18 weights come from torchvision, and the DINOv2 weights (Apache 2.0) from Hugging Face. The classic-architecture and transfer-learning material draws on Magnus Ekman's *Learning Deep Learning* (ch. 7-8 slide decks and notebooks/ch07.ipynb/ch08.ipynb), a copyrighted, not openly licensed textbook, credited here as an internal source and not linked. All figures were drawn or computed for this course. Every number from our own code was read back from the notebook's output.

## Translation

**Why did the CNN fail its language exam?** It was only translation-equivariant.

## 10.1 Learning goals

By the end of this session you should be able to:

- explain why convolution is useful (locality, weight sharing, far fewer weights), and count the weights of a dense and a convolutional layer;
- read image and sequence tensor shapes, and say what kernel size, padding, stride, pooling and the number of filters do;
- distinguish **equivariance** from **invariance**: convolution is equivariant, local pooling adds some tolerance to small shifts, and global pooling turns position-wise features into a position-invariant summary;

- explain how stacking convolutions enlarges the **receptive field**, so that simple local patterns combine into larger ones;
- explain why a CNN can beat a much larger dense network, and why augmentation should encode only label-preserving transformations;
- interpret a first-layer sequence filter as a learned position weight matrix, and say what global pooling throws away when a motif's position matters;
- say what a residual connection does.

The central message of this session is that a dense layer learns a separate weight for every position, while **a convolutional layer assumes that nearby inputs belong together and reuses the same small detector at every position**. This built-in assumption is why CNNs need fewer weights and less data.

## 10.2 Why convolution?

### 10.2.1 The problem with dense layers

Every model so far gives every input position its own weight. For the 150-residue one-hot sequences of Sessions 8-9, that means 3,000 weights per hidden unit, and for a  $224 \times 224$  colour image over 150,000. The same feature detector must then be learned separately at every position, and neighbouring pixels or residues are treated as being as unrelated as distant ones.

### 10.2.2 A small detector, reused everywhere

A **convolutional layer** instead slides a small **kernel** (say  $3 \times 3$  pixels, or 7 residues) across every position, using *the same weights* everywhere. Its number of weights therefore does not depend on the input length. The table compares one layer of each kind on the localisation data (150 residues  $\times$  20 amino acids):

Table 10.1: The number of weights in one dense and one convolutional layer on the localisation data.

| Layer                                                       | Weights                                                  | Count                                       |
|-------------------------------------------------------------|----------------------------------------------------------|---------------------------------------------|
| Dense, 3,000 inputs $\rightarrow$ 64 units                  | inputs $\times$ outputs + biases                         | $3,000 \times 64 + 64 =$<br><b>192,064</b>  |
| Convolution, 20 channels $\rightarrow$ 32 filters, kernel 7 | kernel $\times$ input channels $\times$ filters + biases | $7 \times 20 \times 32 + 32 =$ <b>4,512</b> |

The convolution has 40 times fewer weights, and it keeps that number for a protein of any length. (Whole networks are less simple. In the lab, *adding* a convolutional layer makes the network smaller, because the following dense layer receives a smaller feature map.)

**Shapes.** A batch of images is a tensor  $\mathbf{B} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W}$  (batch, channels, height, width, with  $C = 3$  for colour), and a batch of sequences is  $\mathbf{B} \times \mathbf{C} \times \mathbf{L}$  ( $C = 20$  for one-hot protein, 4 for DNA). A size-7 kernel on protein therefore has  $20 \times 7$  weights, one per amino acid per position. A layer has several kernels (**filters**), and each produces one output channel, that is, one learned feature type. Four more settings control a convolutional layer:

- **kernel size:** the width of the local window;
- **padding:** zeros added at the edges, so that the output can keep the input's length;
- **stride:** how far the kernel moves between steps (1 = every position);
- **pooling:** replacing small regions by their maximum or mean, which shrinks the feature map.

### 10.2.3 Equivariance is not invariance

If the input shifts, a shared kernel sees the same pattern somewhere else, so its output shifts by the same amount. This **translation equivariance** comes from weight sharing itself, and it holds even for a random, untrained kernel:

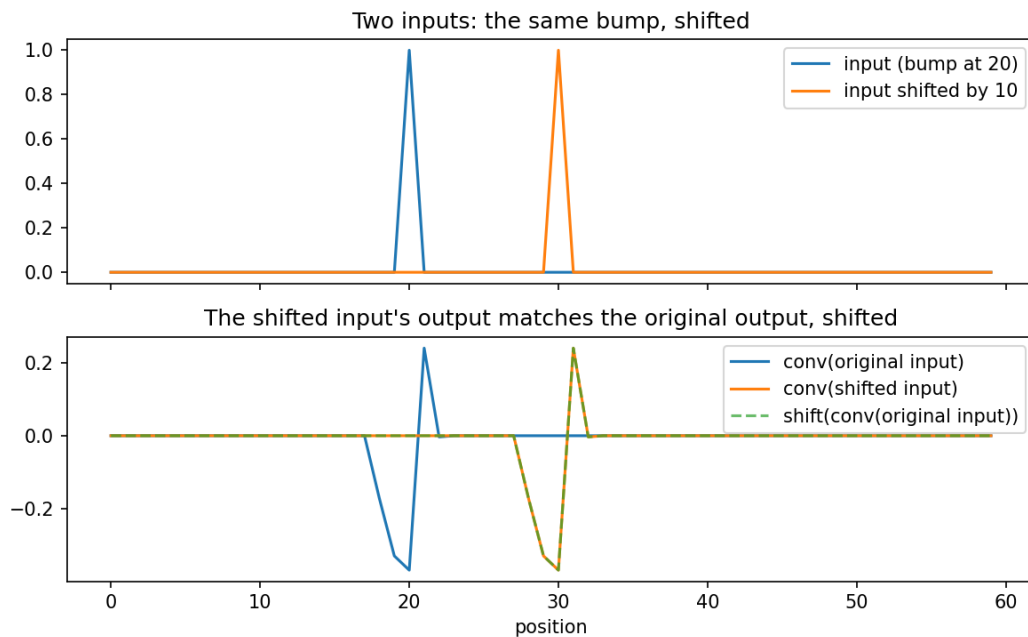


Figure 10.1: A random, untrained 1D convolution applied to a signal and to the same signal shifted by 10 positions. The shifted input's output (green) matches the original output shifted by 10 (orange dashed).

Equivariance is not invariance, however. The feature is found, but *where* it is found still differs, and what happens to that position depends on the rest of the network:

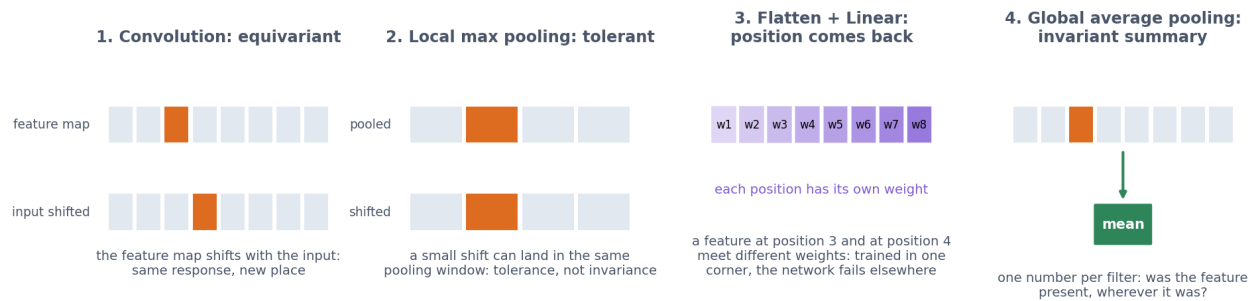


Figure 10.2: Where position information goes. (1) Convolution: the feature map shifts with the input. (2) Local max pooling: a small shift may stay within one pooling window, which gives some tolerance, not invariance. (3) Flatten + Linear: every position gets its own weight again. (4) Global average pooling: one number per filter, whatever the position.

- **Local pooling** (for example  $2 \times 2$  max pooling) gives **some tolerance to small shifts**, because a pattern that moves by one pixel often stays in the same pooling window.
- **Flatten followed by a dense layer** brings absolute position back, because every position of the final feature map gets its own weight. In the lab, CNNs trained on blood cells pasted into one corner reach 0.77-0.86 accuracy on corner cells but only 0.18-0.29 on cells at random positions, close to always guessing the most common class (0.198).
- **Global average pooling** averages each feature map over all positions, giving one number per filter. The network then asks “was this feature present?”, not “was it present at position 17?”. The result is a **position-invariant summary**, used by the localisation CNN below.

Rotation is a different matter, because nothing in convolution or pooling handles it.

#### 10.2.4 Stacking convolutions: the receptive field

A single layer sees only the window of its kernel, but stacking layers widens the view. A unit in a second convolutional layer combines several first-layer units, each with its own window, so it depends on a wider stretch of the input, its **receptive field**.

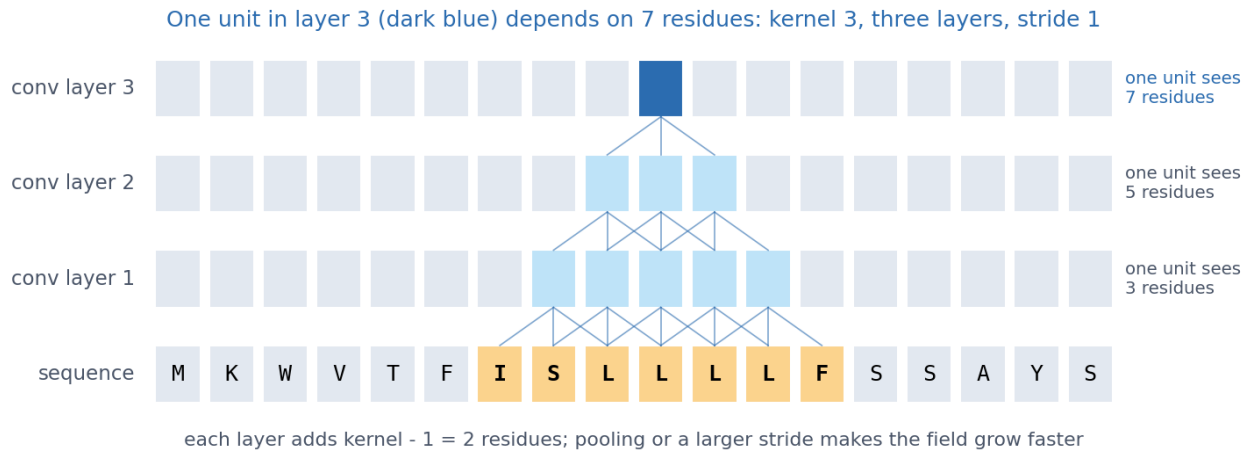


Figure 10.3: Receptive field of stacked 1D convolutions with kernel size 3 and stride 1. A unit in layer 1 sees 3 residues, in layer 2 five, in layer 3 seven.

With kernel size  $k$  and stride 1, every layer adds  $k - 1$  positions, and pooling and larger strides make the receptive field grow faster. This is how deep CNNs build complex patterns from simple ones: edges, shapes and objects in images, and motifs and then combinations of motifs in DNA and proteins. The second discussion problem asks how wide the window must be for a signal peptide.

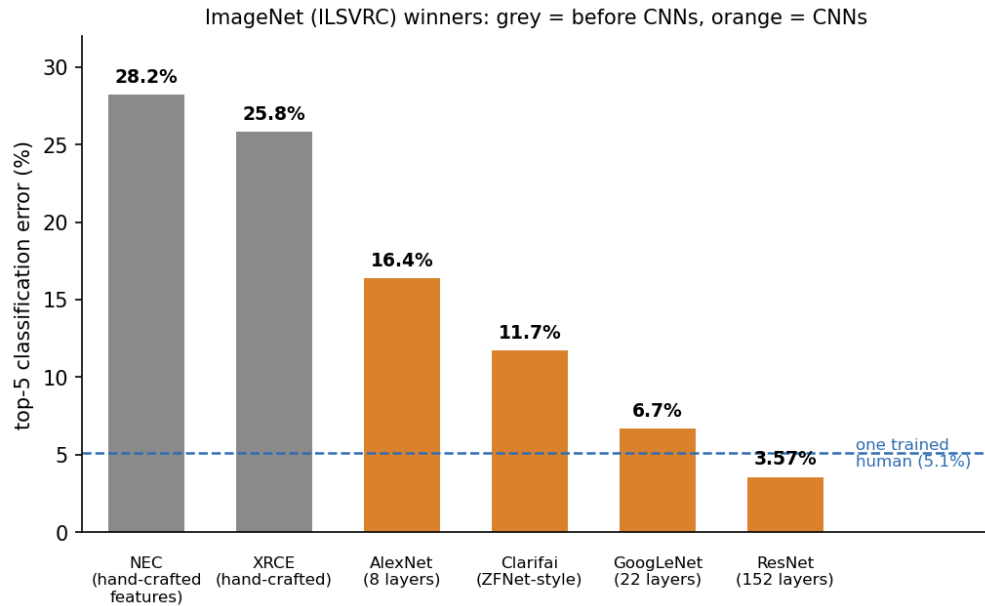
#### 10.2.4.1 Further watching

*But what is a convolution?* (3Blue1Brown): convolution from probability to image blurring, the same sliding weighted sum a CNN learns:

Video: [But what is a convolution?](#), 3Blue1Brown

#### 10.2.5 What actually happened: from LeNet to ResNet

**LeNet** (LeCun et al. 1989, [Neural Computation 1:541-551](#)) already read handwritten zip codes. The breakthrough for natural images, however, came in the 2012 **ImageNet** challenge (1.2 million photos, 1,000 classes):



2012: 16.4% with the provided training data (15.3% with extra data); runner-up 26.2%.

Figure 10.4: Top-5 error of the winning ImageNet entry each year (Russakovsky et al. 2015; He et al. 2016). The first two winners used hand-crafted features; from 2012 every winner was a CNN.

- **AlexNet** (Krizhevsky, Sutskever & Hinton 2012, [NIPS 25](#)) cut the error by about 10 percentage points by combining convolution with ReLU, dropout (Session 9), data augmentation and GPUs.
- **VGG** (2014) used deep stacks of small  $3 \times 3$  convolutions, which give large receptive fields from small kernels.
- **ResNet** (He et al., [CVPR 2016](#)) introduced **residual connections**. Each block learns a correction to its input and returns  $x + F(x)$ , so the gradient has a short path back and 100+ layers become trainable. In the lab, a three-block ResNet beats a three-block VGG network (0.771 against 0.722).

The lesson is architectural: **convolution + depth + data + GPUs**, with residual connections making the depth trainable.

#### 10.2.5.1 Further reading

- d2l.ai, [Modern Convolutional Neural Networks](#) (AlexNet, VGG, ResNet in detail)
- Russakovsky et al. (2015), [ImageNet Large Scale Visual Recognition Challenge](#)

## 10.3 What assumptions does a CNN build in?

### 10.3.1 More weights are not better: dense vs. CNN on blood cells

The lab trains four models on the same BloodMNIST blood-cell images, and the result shows that more weights are not better. The best dense network, with one hidden layer of 128 units and **302,216 weights**, reaches a validation accuracy of 0.793. A small CNN with two convolution blocks and only **17,640 weights** reaches **0.887**. The CNN wins with 17 times fewer weights because it does not have to learn locality and weight sharing from the data, since they are built in. This is Session 9's lesson again, seen from the other side: more weights do not help, but the right assumptions do.

### 10.3.2 Few labels: transfer learning on cancer histology

Biology rarely has 1.2 million labelled images. Kather et al. (2019, [PLoS Med 16:e1002730](#)) cut H&E-stained slides of colorectal cancer into small tiles and labelled each with one of **nine tissue types**:

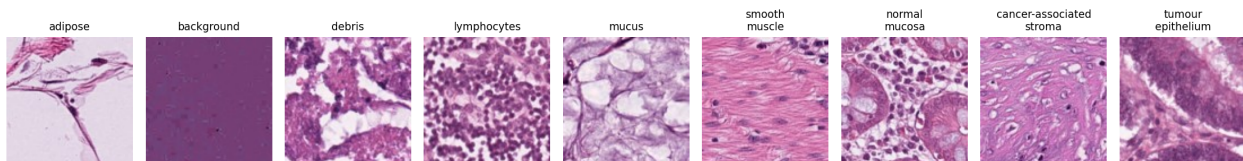


Figure 10.5: One training tile of each of the nine tissue types (NCT-CRC-HE-100K; Kather, Halama & Marx 2018, CC BY 4.0).

To imitate a small dataset, the notebook pretends that we have only **40 labelled tiles per class** for training and 40 for validation, all from the same 86 slides. It tests on 100 tiles per class from **50 other patients**:

Table 10.2: Colorectal-cancer tissue classification from few labelled tiles: training from scratch and transfer learning.

| Approach                                             | Validation accuracy | Test accuracy (new patients)     |
|------------------------------------------------------|---------------------|----------------------------------|
| Small CNN from scratch (242,697 weights)             | 0.861               | 0.745 (three seeds: 0.721-0.786) |
| Same CNN + data augmentation                         | 0.855               | 0.792 (three seeds: 0.769-0.821) |
| Transfer: frozen ImageNet ResNet-18 + new last layer | 0.856               | 0.780                            |
| Transfer: frozen DINOv2 + new last layer             | 0.917               | <b>0.832</b>                     |

- **From scratch**, the CNN memorises its 360 training tiles. With so little data, the random seed alone moves its test accuracy by more than 6 percentage points.
- **Data augmentation** (below) helps on average, though not in every seed.
- **Transfer learning** is what the field actually did. The layers of a network trained on millions of images are kept frozen, as a feature extractor, and only a new last layer is trained. ResNet-18 never saw a tissue tile, yet its features match the augmented CNN.
- **Validation from the same slides, test from new patients.** Every model scores clearly higher on validation than on test, because the validation tiles share patients, staining runs and scanner with the training tiles. This is Session 8's homology trap again. When examples come in groups (proteins in families, tiles in patients), the honest split is by group.

#### **i** Transfer learning from very large image collections

A network trained on a huge image collection has learned features that are useful for a new task with few labels. DINOv2 (Oquab et al. 2023, [arXiv:2304.07193](https://arxiv.org/abs/2304.07193)) was trained on 142 million images *without any labels* (self-supervised), and its features still work best on tissue it has never seen. Such reusable models are called **foundation models**, and Session 11 shows the same recipe for proteins.

### 10.3.3 Which transformations should not matter?

A CNN handles small shifts through its architecture, but everything else it must learn from data. **Data augmentation** is how we tell it what should not matter: in every epoch, the network sees a slightly changed copy of every training image, with the same label. Augmentation should therefore encode only transformations that **preserve the label**, which is a statement about the biology, not about the network. A tissue section or a blood cell on a slide has no “up”, so rotations and mirror images are valid, and small colour changes imitate staining differences between labs. A handwritten 6 rotated by 180° becomes a 9, however, so the same augmentation would be wrong for digits.

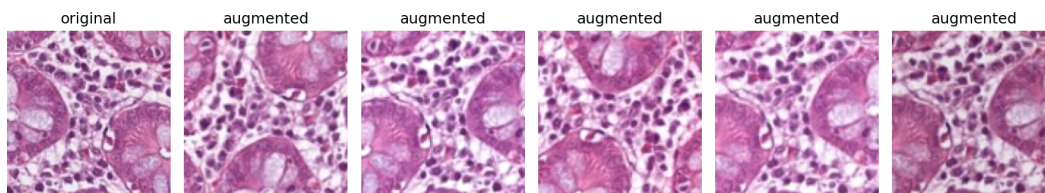


Figure 10.6: One training tile (normal mucosa) and five augmented versions of it.

Does the network need augmentation? The notebook counts how often the **predicted class changes** when the same test tile is shifted, rotated or mirrored. An invariant classifier would change none:

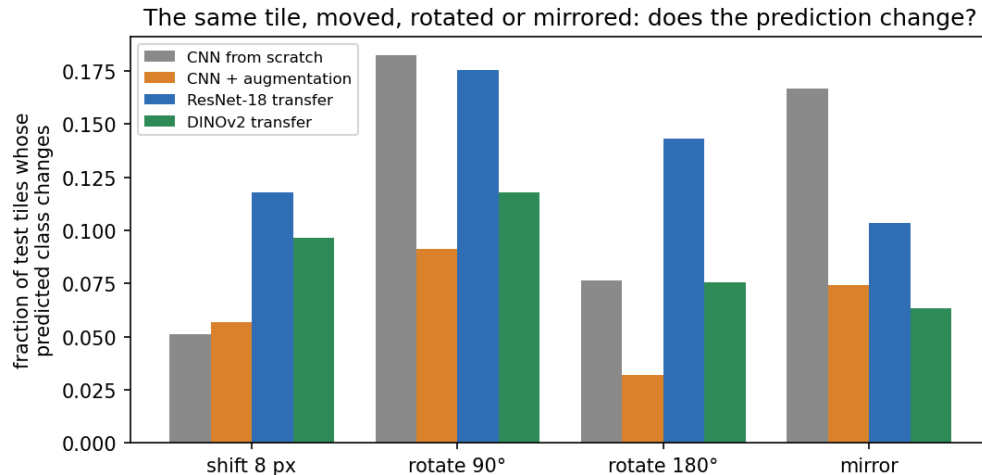


Figure 10.7: Fraction of the 900 test tiles whose predicted class changes when the tile is shifted by 8 pixels, rotated or mirrored, for the CNN without and with augmentation and for the two transfer-learning classifiers.

Without augmentation, an 8-pixel shift changes the CNN’s answer for 5% of tiles, but a 90° rotation changes it for 18% and a mirror image for 17%. Training with rotated and mirrored copies roughly halves the last two (9% and 7%), so the network has learned from data a symmetry that the architecture could not give it. The pretrained networks are not rotation-invariant either, since their training photos are almost always upright. This is why histology pipelines augment with all rotations and flips, and why rotation-equivariant CNNs were developed for pathology (Veeling et al. 2018, [MICCAI](#)).

## 10.4 CNNs on biological sequences

### 10.4.1 Filters become motifs

A first-layer filter on one-hot DNA has shape  $4 \times k$ , with one weight per base per position. Sliding it along the sequence is exactly how a **position weight matrix** scores a sequence (Session 5), except that the matrix is learned instead of counted from aligned sites. Two examples show this:

- **DeepBind** (Alipanahi et al. 2015, [Nat Biotechnol 33:831](#)) trained CNNs on protein-DNA and protein-RNA binding data, and its learned specificities could be “readily visualized as ... position weight matrices”.
- **DeepSEA** (Zhou & Troyanskaya 2015, [Nat Methods 12:931](#)) read 1,000 bp of sequence and predicted 919 chromatin features at once. Comparing its predictions for the reference and a mutated sequence scores the effect of a **non-coding variant**.

Later networks found that many of their filters matched known transcription-factor motifs, rediscovered from sequence alone. Longer-range models needed larger receptive fields and, eventually, Transformers (Session 11).

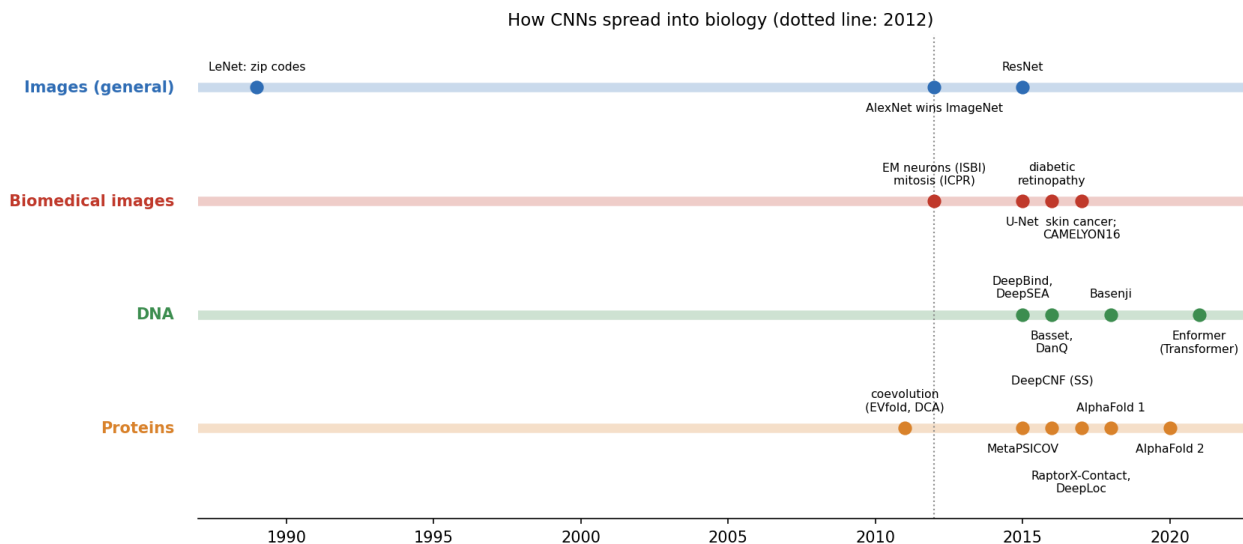


Figure 10.8: How CNNs spread from general computer vision into biomedical imaging, DNA and proteins. Dates are publication years of the papers cited.

The spread was as fast in biomedical imaging. **U-Net** (Ronneberger et al. 2015, [arXiv:1505.04597](https://arxiv.org/abs/1505.04597)) became the standard for segmenting cells and tissues, and ImageNet-pretrained CNNs fine-tuned on retinal, skin or pathology images reached expert level on the tasks they were trained for (e.g. Esteva et al. 2017, [Nature 542:115](https://doi.org/10.1038/nature24256)).

#### 10.4.2 Proteins: our own CNN finds the signal peptide

The same idea works on protein sequences. The localisation network of Sessions 8-9 becomes a 1D CNN with two convolutional layers of 7-residue filters, followed by global average pooling and a linear layer to the five classes. It is trained and evaluated exactly as in Session 9, on the same homology-aware split:

Table 10.3: The CNN compared with the models of Session 9 on the localisation data.

| Metric        | Session 9: flat network | Session 9: logistic regression | Session 10: 1D CNN |
|---------------|-------------------------|--------------------------------|--------------------|
| Weights       | 192,389                 | 15,005                         | <b>19,237</b>      |
| Test accuracy | 0.514                   | 0.569                          | <b>0.587</b>       |
| Macro F1      | 0.493                   | —                              | <b>0.586</b>       |
| MCC           | 0.394                   | 0.461                          | <b>0.484</b>       |
| Macro AUROC   | 0.831                   | 0.849                          | <b>0.869</b>       |

The CNN beats the flat network on every metric with a tenth of the weights. It does so because it builds in an assumption instead of learning 192,389 free weights from 486 proteins: the same local pattern can be detected with the same weights wherever it occurs. Its margin over logistic regression, in contrast, is within the noise of a 109-protein test set (retraining from other random initial weights gave 0.569-0.587). The robust finding is therefore the gap to the flat network.

**What did it learn?** Each first-layer filter is a  $20 \times 7$  matrix, that is, a learned PWM over a 7-residue window. The five filters most selective for secreted proteins correlate with the Kyte-Doolittle hydrophobicity scale at  $r = 0.70$ - $0.83$  (all 32 filters: mean  $r = 0.12$ ). They fire at a median residue position of 11-12 in secreted proteins, against 62-76 elsewhere:

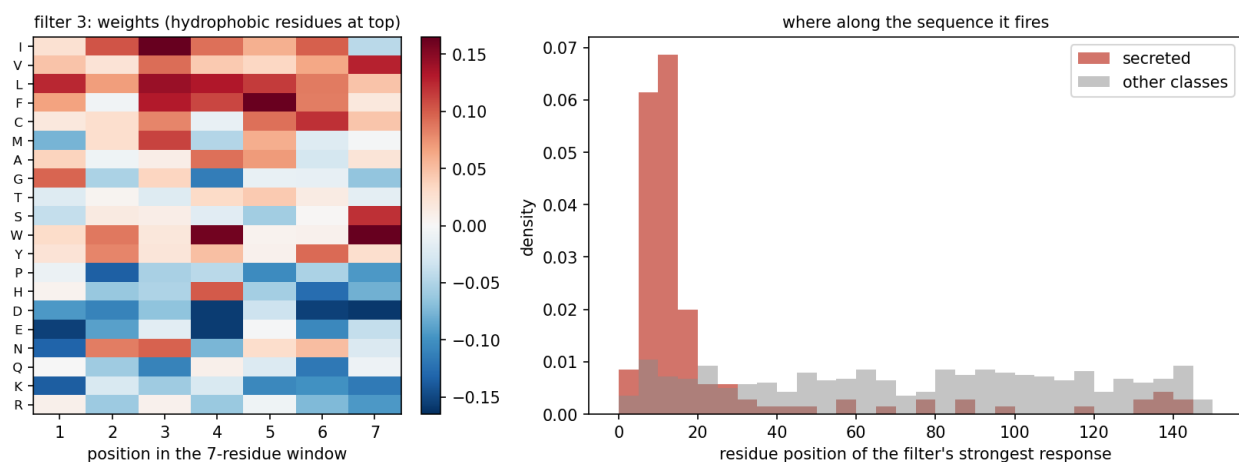


Figure 10.9: Left: weights of the filter most selective for secreted proteins, with amino acids ordered from most hydrophobic (top) to least. Right: where along the sequence that filter responds most strongly, in secreted proteins and in the other four classes.

Nobody told the network about signal peptides, yet it found a detector for their hydrophobic core on its own, just as the DNA filters rediscovered transcription-factor motifs. Each unit of its second layer sees 20 residues (7 in layer 1, then  $2 \times$  pooling and another 7-wide kernel), about the length of a signal peptide.

**But position matters in proteins.** Convolution assumes that a pattern can be *detected* the same way everywhere, not that it *means* the same thing everywhere. A hydrophobic stretch at the N-terminus is a signal peptide, whereas in the middle of a protein it may be a transmembrane helix, and some targeting signals sit specifically at the C-terminus. Global average pooling throws away *where* the filter fired, so our network only knows that a hydrophobic stretch occurred somewhere. **If signal peptides matter at the N-terminus, what has the network lost, and how could you give it back?** Later layers, positional features or position-aware pooling are options. Session 11's models, which read the sequence in order and encode position explicitly, return to signal peptides in depth.

The real tool, DeepLoc (Almagro Armenteros et al. 2017, [Bioinformatics 33:3387](#)), puts

convolutions first and a bidirectional LSTM with attention (Session 11) after them, so that the order-aware part restores position.

#### 10.4.2.1 Further reading

- Zhou & Troyanskaya (2015), [Predicting effects of noncoding variants with deep learning-based sequence model](#)
- Kelley, Snoek & Rinn (2016), [Basset](#)
- Wikipedia, [U-Net](#)

### 10.5 Beyond sequences: a contact map is an image

Convolution is not limited to sequences and photographs. A **contact map** is an  $L \times L$  grid with one pixel per residue pair, set to 1 if the two residues are close in the structure ( $C\beta$  atoms within 8 Å). You first saw this representation in Session 6. For the  $\beta$  chain of haemoglobin (PDB 4HHB) it looks like this:

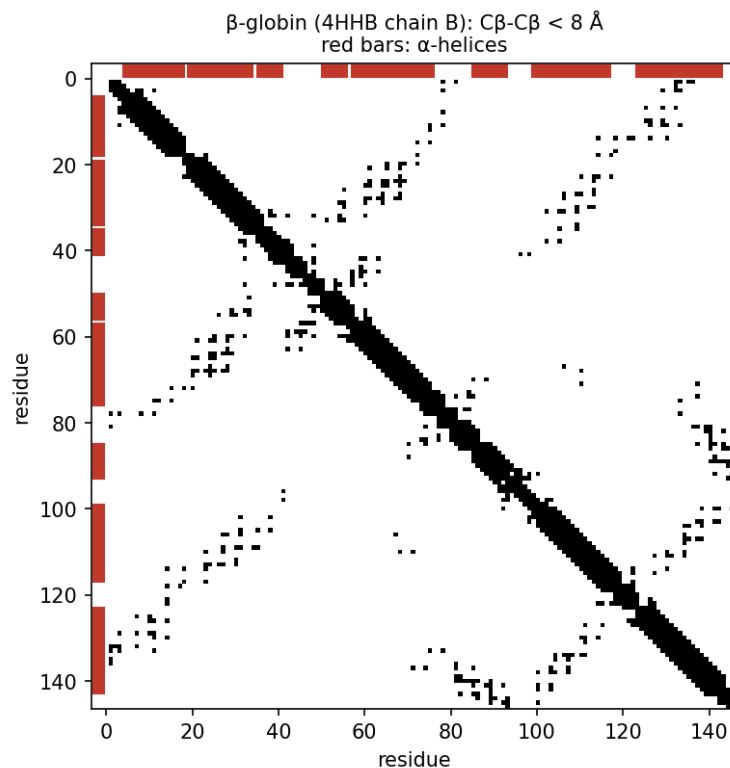


Figure 10.10: Contact map of  $\beta$ -globin (4HHB chain B, 146 residues). The band along the diagonal is each helix contacting itself; the off-diagonal blocks are pairs of helices packed against each other. Red bars: the eight  $\alpha$ -helices annotated in the PDB file.

Contacts form local patterns, as helices give bands and strand pairs give lines along or across the diagonal. Once a protein is an  $L \times L$  map, 2D convolutions are therefore a natural way to predict it. From 2016, deep residual CNNs that predict contact and distance maps transformed structure prediction, and Session 12 tells that story. The third discussion problem asks you to read fold classes from contact maps.

## 10.6 Check your understanding

The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)

## 10.7 The course notebook

`notebooks/session10-cnns-in-code.ipynb` computes every number on this page:

1. the equivariance demonstration; cancer histology from scratch, with augmentation and with transfer learning (ResNet-18, DINOv2); the shift/rotation test;
2. the localisation 1D CNN on the homology-aware split, and its first-layer filters;
3. the contact map of  $\beta$ -globin (PDB 4HHB).

The first part of the notebook trains a small CNN six times for 100 epochs, so use a GPU runtime on Colab (*Runtime*  $\rightarrow$  *Change runtime type*), or lower EPOCHS on a CPU. Open it in [Google Colab](#) via the badge at the top of the notebook page.

## 10.8 The lab

**Full lab, on blood-cell images** (BloodMNIST, as in the Session 8 and 9 labs). The lab is adapted from the previous course's "Neural Networks" lab. Work through [notebooks/session10-lab.ipynb](#), replacing every `todo(...)` call with your own code:

1. count the weights of a dense and a convolutional layer, then of whole models, and see why adding a convolutional layer can make the whole model *smaller* (the next dense layer receives a smaller feature map);
2. train four models side by side, two dense networks of 23,000 and 302,000 weights, a one-layer convolutional network and a small CNN, and see that the CNN, with 17,640 weights, beats them all;
3. find out how many unpadded  $5 \times 5$  convolutions fit on a  $28 \times 28$  image;
4. build a VGG block, and test it on cells placed at positions it never saw during training, before and after training on randomly placed cells, to see that equivariance is not invariance.

Two optional extras, which are not part of the quiz, follow if you have time. In the first you build a ResNet block with a skip connection and compare it with VGG. In the second you rotate and mirror the cells and measure what augmentation buys you.

### **10.8.1 Submitting this lab**

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## **10.9 What's next**

CNNs are excellent when local patterns matter, but they see only as far as their receptive field, and global pooling forgets where things were. Session 11 turns to models that read a whole sequence *in order* and relate distant positions: recurrent networks and LSTMs, the history of language models, and the Transformer. SignalP, from its 1995 feed-forward network to its 2022 protein language model, follows that whole history.

# 11 Session 11: Sequences, Language Models and SignalP

## i Note

**Sources for this page:** d2l.ai, *Dive into Deep Learning*, [Recurrent Neural Networks](#), [Modern Recurrent Neural Networks](#) and [Attention Mechanisms and Transformers](#) — CC BY-SA 4.0. The history of SignalP follows Nielsen, Teufel, Brunak & von Heijne (2024), “[SignalP: The Evolution of a Web Server](#)”, *Methods Mol Biol* 2836:331-367, a history written by the SignalP authors themselves (copyrighted; its findings are described and cited here, and its figures are not reproduced). Every other historical number is taken from the primary paper, cited where it is used. Data: 4,910 reviewed UniProt entries (CC BY 4.0), a copy of which is kept in the course repository; the pretrained ESM-2 model (Lin et al. 2023) via fair-esm. 3Blue1Brown’s videos on language models, Transformers and attention are embedded as further watching. The recurrent-network and attention material follows ideas from Magnus Ekman’s *Learning Deep Learning* (ch. 9-12 slide decks and notebooks), a copyrighted, not openly licensed textbook, credited here as an internal source and not linked. Every number from our own code was read back from the notebook’s output.

## Attention

“Attention is all you need.” — every Transformer, and every teacher during a lecture.

## 11.1 Learning goals

By the end of this session you should be able to:

- explain how a recurrent network carries context along a sequence, why its gradients can vanish or explode, and what the LSTM’s additive memory path changes;
- explain, for residue-level signal-peptide annotation, why a bidirectional LSTM beats a forward-only one, and when a bidirectional model is impossible;
- explain what a language model learns without labels, the difference between masked (BERT-style) and next-token (GPT-style) models, and how attention lets every position consult every other;
- explain what a **contextual embedding** is, and why the same amino acid gets different vectors in different proteins;

- explain, from the classroom version of SignalP 6.0's idea, why pretrained protein-language-model embeddings help most when labelled proteins are scarce.

The whole session answers two questions:

1. **How can a model use context from the rest of a biological sequence?**
2. **How can we learn useful biological representations when labels are scarce but unlabelled sequences are abundant?**

Session 10's CNNs share local detectors and widen their receptive field with depth. But what if residue 10 needs information from residue 200? There are three ways to bring in such context, and this session covers the last two:

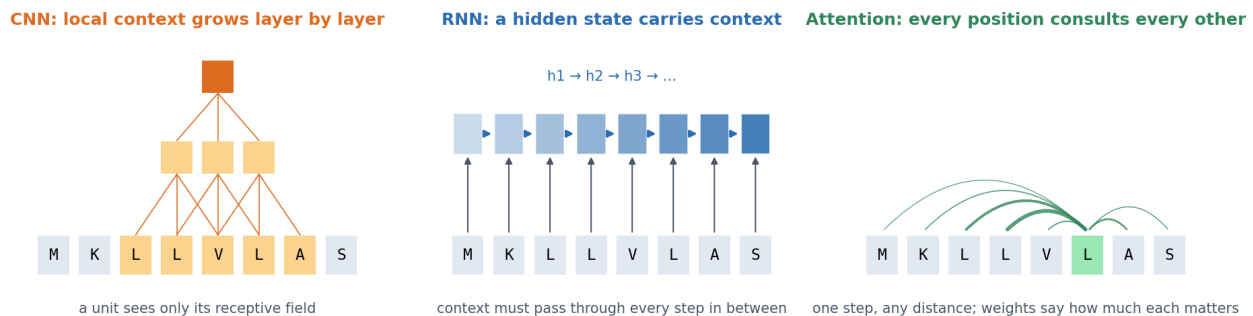


Figure 11.1: Three ways to use sequence context. A CNN unit sees only its receptive field, which grows with depth. An RNN carries a hidden state along the sequence, so context must pass through every step in between. Attention lets every position consult every other directly, with learned weights.

## 11.2 How can a model remember sequence context?

### 11.2.1 A hidden state carried along the sequence

The first way to carry context is to remember it. A **recurrent neural network (RNN)** reads a sequence one position at a time and keeps a **hidden state**, a vector that summarises what it has read so far. At every position it updates this memory from the current residue and the old memory, **new memory = function(current residue, old memory)**, and it uses the same weights at every position:

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

In the notebook, each residue  $x_t$  enters as a learned **embedding** of 24 numbers (nn.Embedding, a trainable vector per amino acid instead of a one-hot vector), and the hidden state  $h_t$  has 48 numbers.  $W_{xh}$  is therefore  $48 \times 24$  and  $W_{hh}$  is  $48 \times 48$ , whatever the length of the protein, so one network handles sequences of any length.

### 11.2.1.1 Why gradients vanish or explode through time

Carrying context through many steps has a cost during training. Training an RNN means backpropagating through every step it took (**backpropagation through time**). The gradient that reaches an early position is a product of many factors, one per step, each involving the recurrent weights and the derivative of the activation. If their effective size is repeatedly below 1, the product tends to vanish, and if it is above 1, the product can explode. This is Session 9’s vanishing-gradient problem again, now along the sequence instead of through the layers. A single repeated factor shows the scale of the effect:

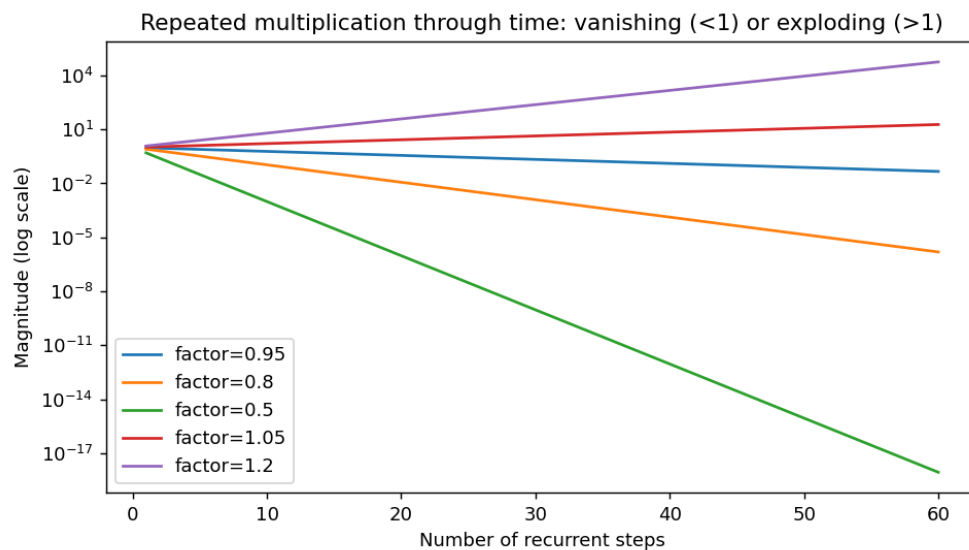


Figure 11.2: Repeated multiplication by a fixed factor over many recurrent steps. At 0.8 the signal shrinks to about  $1.5 \times 10^{-6}$  after 60 steps; at 1.2 it grows to about  $5.6 \times 10^4$ .

### 11.2.2 LSTM: an additive memory path

The **Long Short-Term Memory** network (Hochreiter & Schmidhuber, 1997, [Neural Computation 9:1735](#)) was designed to solve this problem. It adds a separate memory, the **cell state**, which is updated by *adding* to it rather than by recomputing it. Small sigmoid **gates** decide what to keep, what to add and what to expose. Because the memory is carried forward by addition, the error signal can flow back over many steps without shrinking at every one, as long as the forget gates stay close to 1 (it still shrinks where they close). This mitigates the vanishing gradient; it does not abolish it. LSTMs therefore make much longer dependencies trainable than a simple RNN can, although how long in practice depends on the task. The notebook writes the gate equations out and checks them against PyTorch’s `nn.LSTMCell`. A **GRU** is a later, simpler gated recurrent network, with fewer gates and no separate cell state.

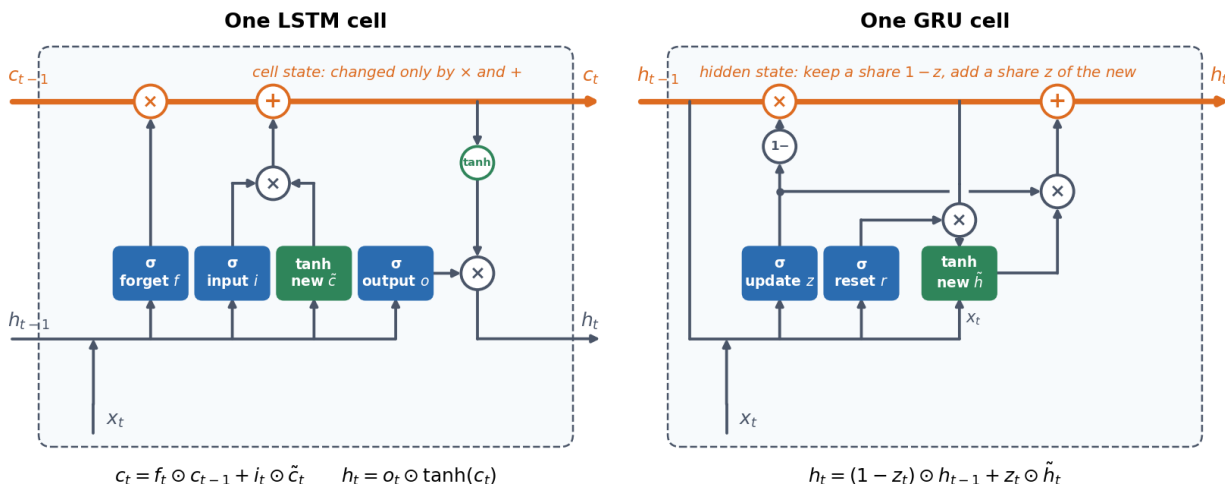


Figure 11.3: One LSTM cell and one GRU cell. Blue boxes are sigmoid gates, with outputs between 0 and 1; green boxes compute a new candidate value with  $\tanh$ ;  $\times$  and  $+$  act element by element. In the LSTM (left), the cell state  $c$  (orange) is changed only by the forget gate's multiplication and the addition of new content, which is the additive memory path. The GRU (right) has no separate cell state: its update gate  $z$  mixes the old hidden state with a new candidate, and its reset gate  $r$  decides how much of the old state the candidate sees.

An LSTM that reads from left to right still knows only what came *before* each position. A **bidirectional LSTM** (Schuster & Paliwal, 1997) removes this limitation. It runs one LSTM from left to right and another from right to left, and it combines them at every position, so each position also sees what comes *after* it.

### 11.2.3 Case study: which residues belong to a signal peptide?

A good test case for these models is the signal peptide. **Signal peptides** are short N-terminal sequences that send a protein into the secretory pathway, a discovery for which Günter Blobel received the [1999 Nobel Prize](#). They have three regions: a positively charged **n-region**, a hydrophobic **h-region**, and a polar **c-region** that ends at the cleavage site. Our dataset has 4,910 reviewed UniProt proteins from eukaryotes and bacteria, in three groups:

- 1,936 proteins with an **experimentally supported** signal peptide (ECO:0000269). Many UniProt signal peptides are predicted by SignalP itself, and training on those would teach a model to imitate SignalP.
- 1,988 cytoplasmic or nuclear proteins.
- 986 transmembrane proteins. These are *hard* negatives, because a transmembrane helix is also a hydrophobic stretch.

The regions show up when the experimentally supported signal peptides are aligned on their cleavage site:

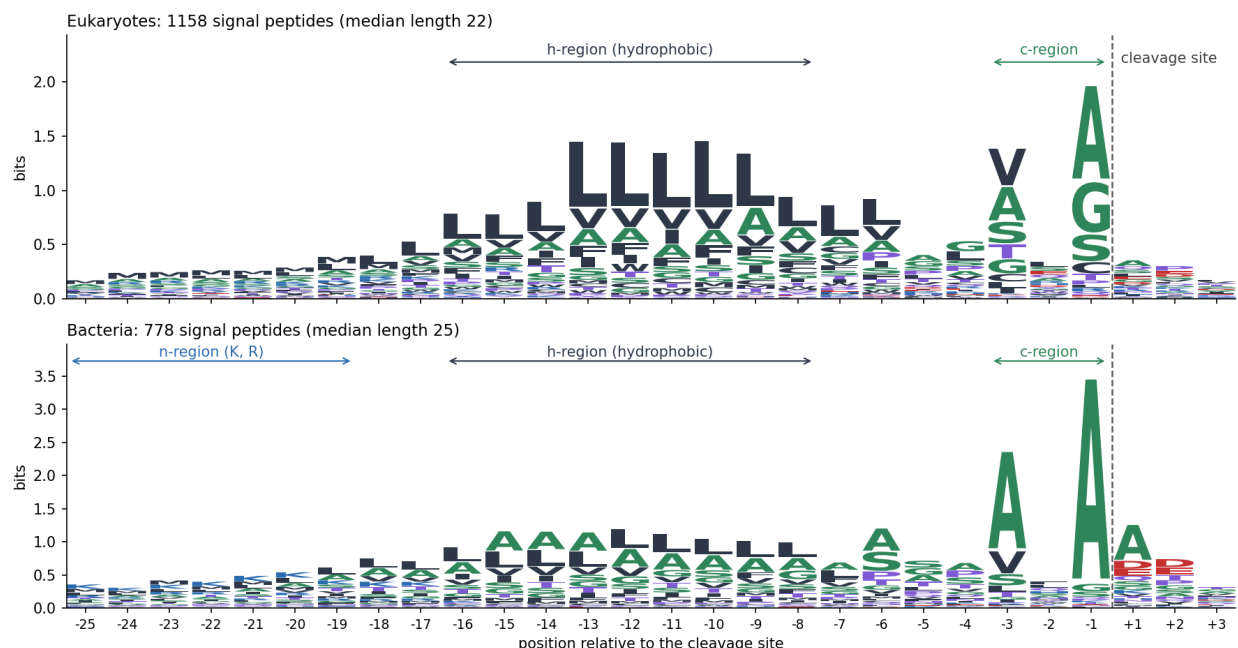


Figure 11.4: Sequence logos of the experimentally supported signal peptides in our dataset, aligned on the cleavage site (dashed line). Each stack shows how conserved a position is (in bits); letters are coloured by property: hydrophobic dark grey, small (A, G, S) green, positive blue, negative red, other polar purple. Leucines dominate the hydrophobic h-region, and small residues at  $-3$  and  $-1$  mark the cleavage site (the “ $-3, -1$  rule”). The positively charged n-region (K, R) is clearest in bacteria, where the signal peptides are longer. The two logos have different y-scales.

The split is homology-aware (Session 8). The proteins fall into 3,205 MMseqs2 clusters at 30% identity, which are divided into 3,432 training, 737 validation and 741 test proteins. Each of the first 70 residues is labelled as signal peptide or not:

Table 11.1: Per-residue signal-peptide labelling on the test set.

| Model (per-residue labelling, test set)          | Weights | MCC          | AUPR         |
|--------------------------------------------------|---------|--------------|--------------|
| Sliding window (15 residues) + linear classifier | —       | 0.539        | 0.681        |
| Simple RNN                                       | 4,105   | 0.620        | 0.770        |
| LSTM                                             | 14,761  | 0.631        | 0.767        |
| <b>Bidirectional LSTM</b>                        | 29,017  | <b>0.820</b> | <b>0.920</b> |

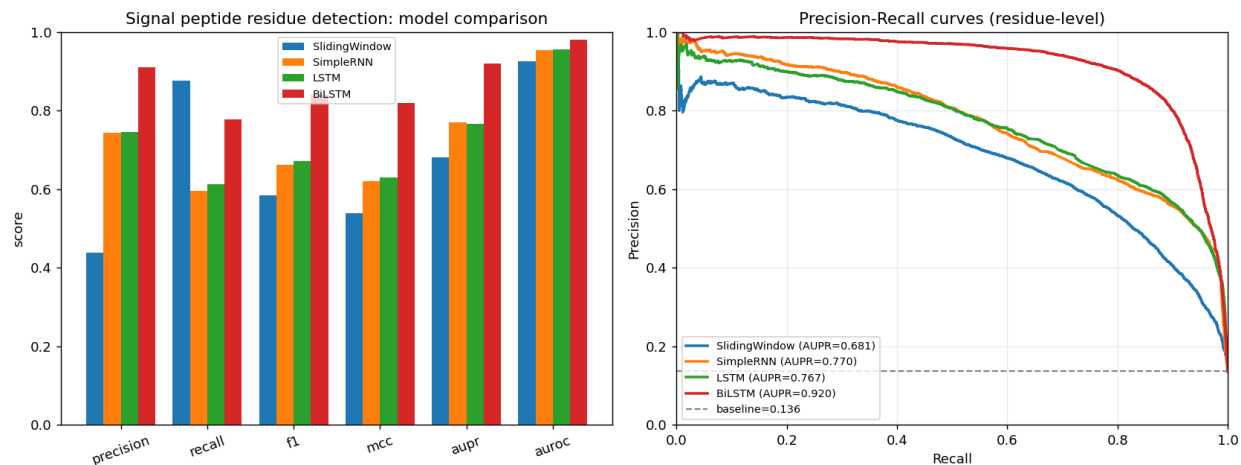


Figure 11.5: Residue-level comparison of the four models on the homology-aware test set: summary metrics (left) and precision-recall curves (right).

**Why does reading both directions help so much?** Think of residue 3, in the n-region. When a forward LSTM labels it, it has seen only “M-x-x”, and nothing yet of the hydrophobic h-region that identifies a signal peptide. Whether residue 3 belongs to a signal peptide is decided by what comes *after* it. The first discussion problem measures this effect. The forward LSTM mislabels 76% of the first five residues of signal peptides, and the bidirectional one only 12%. Near the cleavage site, in contrast, both err about equally often (0.20 and 0.23). The forward model also labels the N-terminus of many proteins without a signal peptide as part of one, because it cannot yet see that no h-region follows.

SignalP 5.0 (2019) was built from these parts. It is a deep network with a bidirectional LSTM and, on top, a **conditional random field** (CRF). The CRF is a structured output layer that favours biologically plausible label sequences, for example a signal peptide that starts at residue 1 and does not stop and restart.

### 11.2.3.1 Further reading

- d2l.ai, [Modern Recurrent Neural Networks](#) (LSTM, GRU, bidirectional RNNs)
- Wikipedia, [Long short-term memory](#)

## 11.3 Can we learn from sequences without labels?

So far every model has needed labels, such as residues marked as signal peptide by experiment. Labels are expensive, but sequence databases hold hundreds of millions of unlabelled proteins (Session 3). Can a model learn useful biology from the sequences alone?

### 11.3.1 What a language model is

A **language model** learns the regularities of sequences by predicting missing or future tokens, without any biological labels, so the sequences provide their own training signal. There are two kinds:

- **autoregressive** (GPT-style) models predict the *next* token from the previous ones;
- **masked** (BERT-style) models, including protein models such as ESM-2, hide some tokens and predict them from the context on *both* sides.

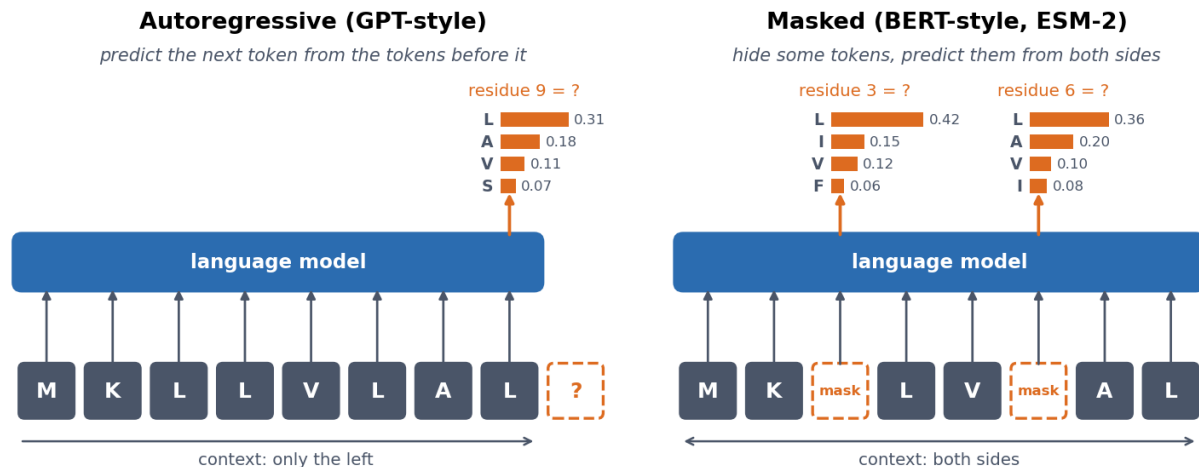


Figure 11.6: The two kinds of language model, here reading the start of a signal peptide. An autoregressive model (left) sees only the tokens to the left and predicts the next one; a masked model (right) predicts hidden tokens from both sides. The output is a probability for every amino acid; the numbers shown are illustrative.

To predict well, a model must learn what tends to go with what. These learned regularities are stored in its internal vectors (**embeddings**), and they turn out to be useful for many other tasks.

**A tiny one, in the notebook.** An LSTM trained to predict the next residue reaches 4.071 bits per residue on the test proteins. This is only a little better than knowing the amino-acid composition (4.164 bits). Large models get much further, with far more data and a better architecture.

#### 11.3.1.1 Further watching

[Large Language Models explained briefly](#) (3Blue1Brown): a short overview of what a language model is and how it is trained:

Video: [Large Language Models explained briefly](#), 3Blue1Brown

### 11.3.2 Attention, from scratch

The better architecture is built on attention. For every position, **self-attention** asks: *which other positions should I take information from?* For residue  $i$  it:

1. compares residue  $i$  with every residue  $j$ ;
2. gives each  $j$  a relevance score;
3. normalises the scores so that they sum to 1 (softmax);
4. updates residue  $i$  with a weighted average of what the other residues contribute.

To do this, each residue's vector is turned into three vectors by learned linear maps: a **query** (what am I looking for?), a **key** (what do I contain?) and a **value** (what do I pass on?). The score of  $j$  for  $i$  is the match between the query of  $i$  and the key of  $j$ . In matrix form, for all positions at once:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right) V$$

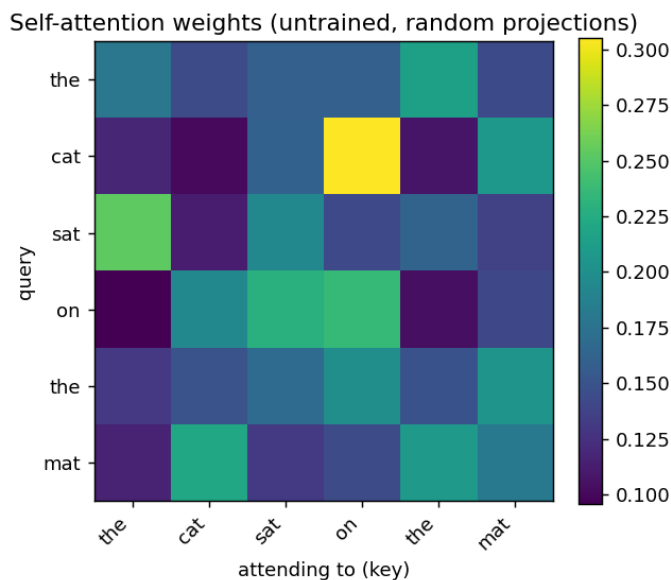


Figure 11.7: Self-attention weights for a six-token toy sentence with random (untrained) projections. Each row, one query, is a probability distribution over all positions.

Unlike an RNN, attention connects any two positions in one step, however far apart they are, and it computes all positions at once. Two cautions apply. First, a high attention weight means that residue  $j$  contributes strongly when residue  $i$  is updated. It does not prove a 3D contact, and it is not by itself a biological explanation (the second discussion problem tests how much structural signal attention maps actually carry). Second, attention by itself has no idea of *order*, so “ACDE” and “EDCA” give the same set of outputs, just shuffled. The Transformer adds the order back.

### 11.3.2.1 Further watching

[Attention in transformers, step-by-step](#) (3Blue1Brown, Deep Learning ch. 6): queries, keys, values, multiple heads and the causal mask, animated:

Video: [Attention in transformers, step-by-step, 3Blue1Brown](#)

Going further: [How might LLMs store facts](#) (3Blue1Brown, Deep Learning ch. 7), about the feed-forward layers inside a Transformer.

### 11.3.3 The Transformer: attention plus four tricks

Attention alone is not yet a complete model. A **Transformer** (Vaswani et al. 2017) is a stack of identical **blocks**. Each block takes one vector per token and returns one vector per token of the same size, so blocks can be stacked as deep as the budget allows.

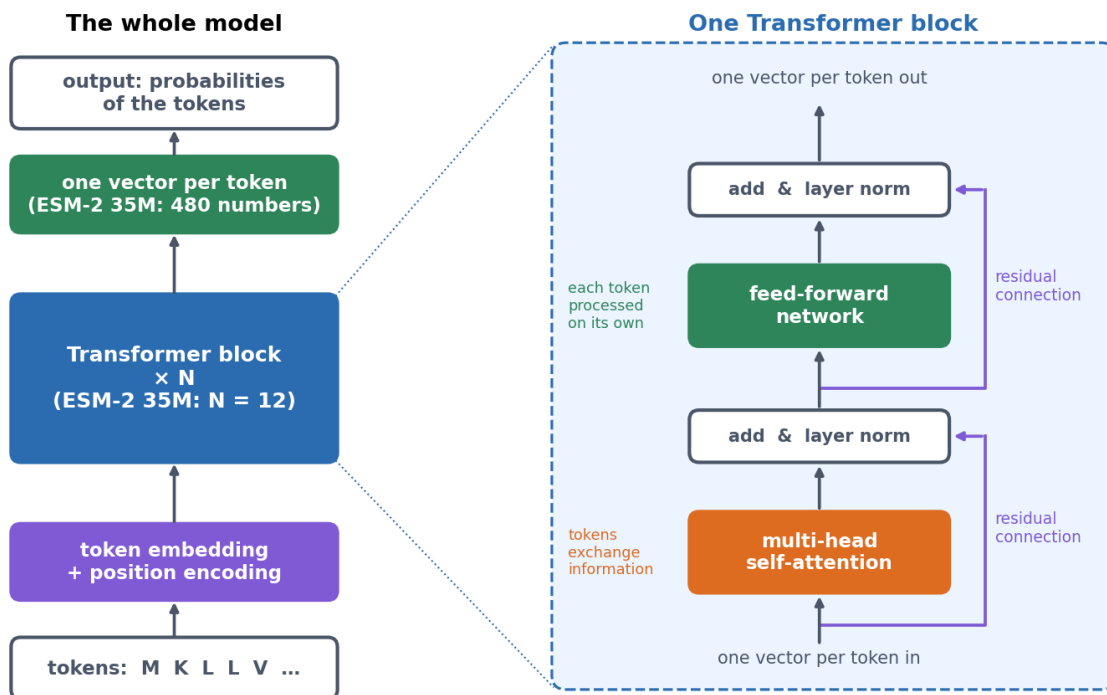


Figure 11.8: A Transformer. Left: tokens become vectors, pass through N identical blocks, and come out as one vector per token, from which the model predicts tokens. Right: one block. This is the original (2017) arrangement; many newer models, ESM-2 among them, apply the layer normalisation before each step instead of after it.

Besides self-attention, a block uses four tricks, most of which you have met before:

- **Several heads.** Several attentions run side by side (20 in ESM-2 35M), and each is free to follow a different relationship.

- **A feed-forward network** (Session 7’s MLP) is applied to every token on its own. Attention moves information *between* tokens, and the feed-forward network processes it *within* each token.
- **Residual connections and layer normalisation.** Each step adds its output to its input (Session 10’s ResNet idea), so gradients survive dozens of blocks.
- **Position encoding.** Information about the position of each token is added before or inside the attention. Without it, “ACDE” and “EDCA” would look the same, and a protein model would not know which residue comes first.

The notebook’s ESM-2 has 12 blocks and 480 numbers per residue (33.5 million weights), while the largest ESM-2 has 48 blocks and 15 billion weights.

**BERT vs. GPT is one mask**, and it is the same distinction as between a forward and a bidirectional LSTM. A BERT-style encoder lets every position attend to every other position and learns by filling in masked tokens. This is fine when the whole sequence is known, as in annotation. A GPT-style decoder masks out the *future*, so each position sees only earlier ones and learns by predicting the next token. This is what lets it generate sequences:

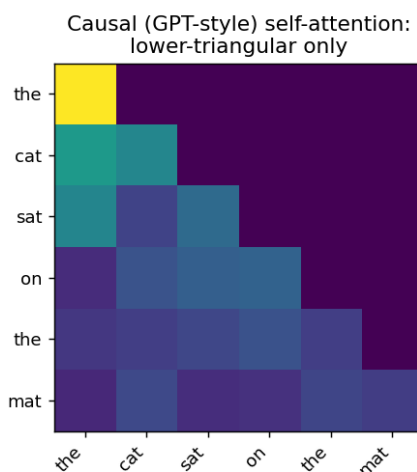


Figure 11.9: Causal (GPT-style) attention: each token can only attend to itself and earlier tokens.

Protein language models such as ESM-2 are encoders, trained BERT-style, so every residue sees the whole sequence.

### 11.3.3.1 Further watching

[Transformers, the tech behind LLMs](#) (3Blue1Brown, Deep Learning ch. 5): how a GPT-style model turns tokens into vectors and predicts the next token:

Video: [Transformers, the tech behind LLMs, 3Blue1Brown](#)

### 11.3.4 What actually happened

These pieces arrived in a particular order. Language models started by counting which word follows which (n-grams) and then moved to recurrent networks and LSTMs that learned word vectors. **Attention** (Bahdanau, Cho & Bengio 2014, [arXiv:1409.0473](#)) was first added to recurrent translation models. The **Transformer** (Vaswani et al. 2017, [arXiv:1706.03762](#)) then kept only the attention and could process all positions in parallel. This made **large-scale pretraining** practical: a model is trained once on huge amounts of unlabelled text and then adapted to many tasks. The two branches are **GPT** (Radford et al. 2018, [paper](#); next-token, causal) and **BERT** (Devlin et al. 2018, [arXiv:1810.04805](#); masked, bidirectional). Scaled up, the GPT branch became today's large language models.

### 11.3.5 Protein language models

The same recipe works for proteins. There are hundreds of millions of unlabelled sequences, and a model that learns to fill in missing residues has to learn something about structure, function and evolution. Protein language models developed in three steps:

- **An early precursor.** SignalP's authors note that the network program their group used before SignalP could erase the central residue of a window and predict it. This was "presumably ... the first machine learning language model" in biological sequence analysis (Nielsen et al. 2024, [Methods Mol. Biol. 2836:331](#)).
- **Recurrent protein language models** (2019), such as UniRep and SeqVec, learned embeddings from tens of millions of unlabelled sequences.
- **Transformer protein language models** (2021 onwards) then took over. They include **ProtBERT** (Elnaggar et al. 2022, [IEEE TPAMI 44:7112](#)), the model inside SignalP 6.0, and the **ESM** family. This session uses **ESM-2** (Lin et al. 2023, [Science 379:1123](#); 8 million to 15 billion weights, masked language modelling). The attention maps of these models turned out to contain information about residue contacts, although they were never trained on structures (Rao et al. 2021, [ICLR](#)). The largest ESM-2 models even predict 3D structure from a single sequence (ESMFold), which is the bridge to Session 12.

**The model knows what a signal peptide looks like.** In the notebook, ESM-2 (35M parameters) is given a real secreted protein in which residue 11, inside the hydrophobic h-region, is masked. It predicts leucine with probability 0.41, followed by A, V and G, and the true residue is indeed L. At the same position in a cytoplasmic protein, in contrast, its guesses are spread out, with no residue above 0.11. Nobody told the model about signal peptides.

### 11.3.6 Same residue, different context: contextual embeddings

In a one-hot encoding (Sessions 7-10), every leucine is the same vector, wherever it occurs. (You have met one learned representation before: in Session 6, Foldseek

turned the 3D environment of each residue into a letter of a learned 3Di alphabet.) A protein language model instead computes the vector of each residue from the whole sequence around it. This is a **contextual embedding**, and with it a leucine in a signal peptide and a leucine in a cytoplasmic enzyme get different vectors. To see this, the notebook collects the ESM-2 vectors of 2,924 leucines from the first 70 residues of 450 proteins:

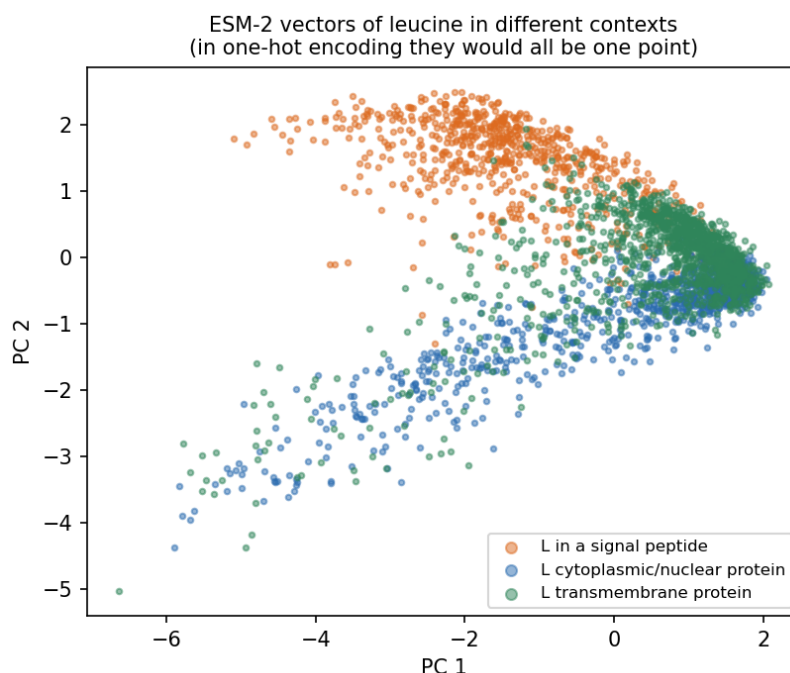


Figure 11.10: PCA of the ESM-2 vectors of leucines in signal peptides (orange), in cytoplasmic or nuclear proteins (blue) and in transmembrane proteins (green). In a one-hot encoding all of them would be a single point.

In one-hot encoding, every pair of these leucines has cosine similarity 1. In ESM-2, leucines inside signal peptides have a mean cosine similarity of 0.745 with each other, but only 0.633 with leucines in cytoplasmic proteins, so the vector carries information about the surroundings of the residue. This is what a classifier on top of a protein language model gets to work with, and it is probably the single most important idea of this part.

### 11.3.7 Foundation models: pretrain once, reuse everywhere

ESM-2 is a **foundation model** (Bommasani et al. 2021, [arXiv:2108.07258](https://arxiv.org/abs/2108.07258)). Such a model is trained once on huge unlabelled data and then reused for many tasks, in three steps:

1. **Pretrain** on everything available, with no labels (UniRef for proteins, the internet for text, billions of photos for images).
2. **Represent** each new example by the model's embedding.

3. **Adapt** with a small amount of labelled data, either by training a small model on top of the frozen embeddings or by fine-tuning the whole network a little.

You have met this recipe three times: DINOv2 on histology tiles (Session 10), ESM-2 here, and SignalP 6.0 below, which is ProtBERT plus a small model on top. A foundation model helps most when labelled data are scarce, but it does not remove the need for an honest split.

#### 11.3.7.1 Further reading

- d2l.ai, [Attention Mechanisms and Transformers](#)
- Vaswani et al. (2017), [Attention Is All You Need](#)
- Devlin et al. (2019), [BERT](#)
- Lin et al. (2023), [Evolutionary-scale prediction of atomic-level protein structure with a language model](#)

## 11.4 Pretraining changes biological prediction

### 11.4.1 SignalP: thirty years in three steps

SignalP has been online since 1995, is one of the most used tools in bioinformatics, and runs through this whole course. Its history shows each of the model types we have met:

Table 11.2: SignalP versions and the models behind them.

| Versions                | Model                                                                                                                |
|-------------------------|----------------------------------------------------------------------------------------------------------------------|
| SignalP 1-4 (1997-2011) | feed-forward networks on sliding windows (Session 7), combined with hidden Markov models (Session 5) in versions 2-3 |
| SignalP 5.0 (2019)      | a deep recurrent network: bidirectional LSTM + CRF (above)                                                           |
| SignalP 6.0 (2022)      | the pretrained protein language model <b>ProtBERT</b> + CRF                                                          |

(Nielsen et al., [Protein Eng 10:1](#); Almagro Armenteros et al., [Nat Biotechnol 37:420](#); Teufel et al., [Nat Biotechnol 40:1023](#); history by the authors themselves in Nielsen et al. 2024, [Methods Mol. Biol. 2836:331](#).)

**What changed when a representation pretrained on huge unlabelled protein collections became available?** The SignalP 6.0 paper reports four changes:

- **Few-shot learning.** Two signal-peptide types had almost no training data (36 and 113 sequences). With ProtBERT, both became predictable for the first time, which “confirms the importance of LMs for low-data problems”.
- **Better generalisation.** The two versions were equal on close homologs of the training data, but “at sequence identities lower than 60%, SignalP 6.0 outperforms SignalP 5.0”.
- **No organism input needed.** The language model infers the taxonomic context from the sequence itself, so SignalP 6.0 also works on metagenomic proteins of unknown origin.
- **Homology partitioning, again.** For SignalP 6.0, the authors found that their earlier partitioning had left similar sequences on both sides. They built a new tool (GraphPart; Teufel et al. 2023, *NAR Genom Bioinform* 5:lqad088), and even with it they could not get below 30% identity between folds. This is Session 8’s lesson, learned the hard way.

### 11.4.2 Reproducing the idea at classroom scale

Can we see the same effects ourselves? We use a simpler task, *does this protein have a signal peptide?*, on the same 4,910 proteins and their first 70 residues. We compare three classifiers:

- one-hot encoding with logistic regression;
- one-hot encoding with a small 1D CNN (Session 10);
- logistic regression on the **frozen ESM-2 embedding** (35M parameters, a stand-in for the larger ProtBERT), with the residue vectors averaged into 480 numbers.

**On the homology-aware test set** (741 proteins, used once):

Table 11.3: Classifiers on the homology-aware test set (741 proteins).

| Classifier                                   | MCC          | Accuracy     |
|----------------------------------------------|--------------|--------------|
| one-hot + logistic regression                | 0.745        | 0.879        |
| one-hot + 1D CNN                             | 0.825        | 0.916        |
| <b>ESM-2 embedding + logistic regression</b> | <b>0.955</b> | <b>0.978</b> |

**Few-shot.** To mimic scarce labels, we train on random subsets of the training proteins:

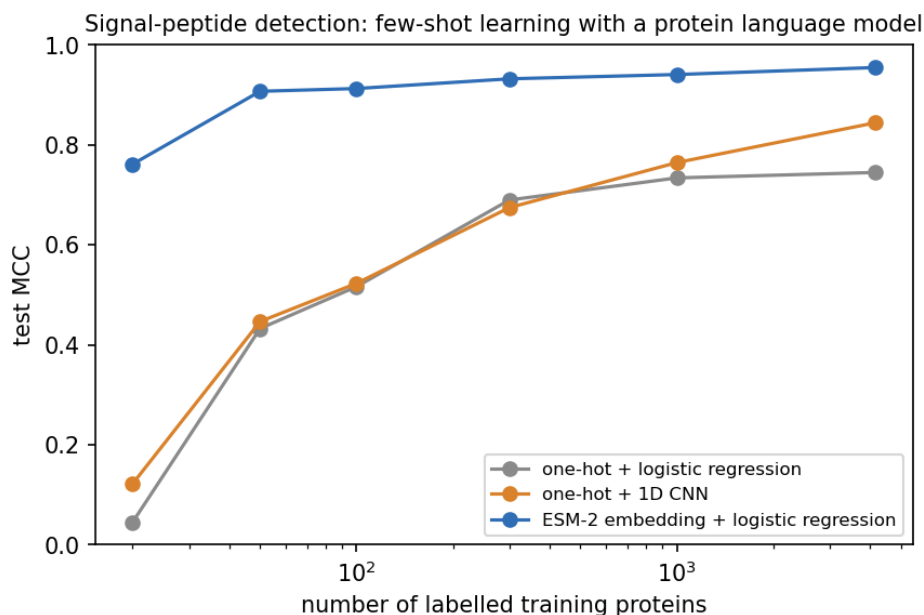


Figure 11.11: Test MCC against the number of labelled training proteins (log scale). With 20 proteins, the ESM-2 classifier reaches MCC 0.762; the one-hot models are near zero (0.045 and 0.122).

With only 50 labelled proteins, the ESM-2 classifier (MCC 0.907) already beats both one-hot models trained on all 4,169 (0.745 and 0.845). This is the central result of the session: **pretraining transfers information learned from millions of unlabelled sequences into a task where labelled examples are scarce**. The language model has already learned most of what distinguishes a signal peptide, and the labels only need to point at it.

**Generalisation.** SignalP 6.0's authors compared performance on test proteins at different identities to the training set. To see the whole range of identities, we use a plain random 80/20 split and measure the identity of each test protein to its most similar training protein with MMseqs2. This is a diagnostic that shows why random splits can be deceptively easy, not an argument for using them. The homology-aware result above remains the honest estimate for new protein families (Session 8).

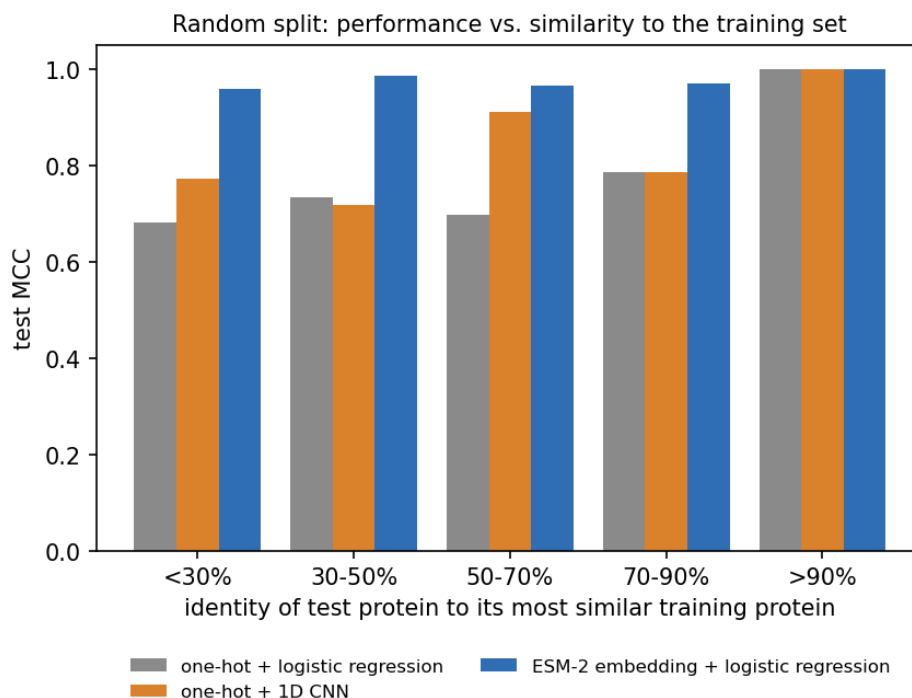


Figure 11.12: Test MCC in bins of identity to the most similar training protein. Proteins with no detectable homolog (37% of the test set) fall in the lowest bin.

Close homologs (>90% identity) are predicted perfectly by all three classifiers. Without a close homolog (<30%), however, the one-hot classifiers drop to MCC 0.681 and 0.773, while the ESM-2 classifier stays at 0.960. This is the same pattern that SignalP 6.0's authors found. (Our task is easier than SignalP's, and our model is smaller and not fine-tuned. ESM-2 has seen these sequences during pretraining, but never their labels, which is also true of ProtBERT inside SignalP 6.0.)

#### 11.4.2.1 Further reading

- Teufel et al. (2022), [SignalP 6.0 predicts all five types of signal peptides using protein language models](#) (open access)
- Teufel et al. (2023), [GraphPart: homology partitioning for biological sequence analysis](#)
- SignalP 6.0 [web server](#)

## 11.5 Check your understanding

The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)

## 11.6 The course notebook

`notebooks/session11-rnns-lstm-gru-in-code.ipynb` computes every number on this page:

1. Vanishing and exploding gradients through time, an LSTM cell written out by hand, and residue labelling with a sliding window, a simple RNN, an LSTM and a bidirectional LSTM on the homology-aware split.
2. A next-residue LSTM language model, self-attention and the causal mask from scratch, ESM-2 filling in a masked residue, and the contextual embeddings of leucine.
3. One-hot vs. frozen ESM-2 embeddings: the homology-aware test, the few-shot curve, performance vs. identity, and the PCA.

The dataset is loaded from the course repository. On Colab, run `!pip install fair-esm` first. Everything runs on a CPU in a few minutes. Open the notebook in [Google Colab](#).

## 11.7 The lab

**Full lab:** the lab is adapted from the previous course’s “Neural Networks” labs and rebuilt in PyTorch on biological data. Work through [notebooks/session11-lab.ipynb](#), replacing every `todo(...)` call with your own code:

1. label signal-peptide residues with a feed-forward sliding window of 1, 11 and 41 residues, and see how the window size matters;
2. build a forward and a bidirectional LSTM tagger for the same task, and explain why reading the sequence in both directions helps;
3. compute ESM-2 embeddings and cosine similarities for two myoglobins and an unrelated membrane transporter;
4. embed eleven PDB proteins, plot them with PCA, cluster them, and check the clusters against their Pfam families.

An optional extra at the end of the notebook, not part of the quiz, lets you write one step of a simple RNN by hand and check it against `nn.RNN`.

### 11.7.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course’s Canvas page for the current quiz, the deadline and how to submit.

## 11.8 What's next

This session has shown three ways to use context. A CNN widens local context layer by layer, an RNN carries context step by step, and attention lets any position use any other directly. It has also shown two ways to learn a representation. A supervised model learns it from task labels, while a protein language model learns it first, from millions of unlabelled sequences, so that a few labels go a long way.

Session 12 is about **AlphaFold** and its relatives. They use attention over multiple sequence alignments and over residue pairs to predict 3D structure, and protein language models such as ESM-2 can do the same from a single sequence. Along the way, Session 10's contact maps become distances, and then coordinates.

## 12 Session 12: AlphaFold and Generative Models of Protein Structure

### **i** Note

**Sources for this page:** Arne Elofsson's AlphaFold lecture (*AlphaFold 2026, undervisning*) and diffusion-models lecture (*slides/Diffusion models (undervisning).pdf*), which this page follows; every paper named below is the primary source for its numbers, cited where it is used. Data and models are fetched live from the RCSB PDB, the [AlphaFold Protein Structure Database](#) (CC BY 4.0) and the ESM Atlas API. All figures were drawn or computed for this course, and every number from our own code was read back from the notebook's output. The "four regions" heuristic in Section 12.4 is taken from the lecture; it has not yet been published as a paper.

### A solved problem

Protein folding was a 50-year-old problem. Then AlphaFold solved it, and created 200 million new ones.

### 12.1 Learning goals

By the end of this session you should be able to:

- explain why long-range information is essential for structure prediction, and how evolutionary covariation provided it;
- describe what AlphaFold2 represents internally (an MSA representation and a residue-pair representation that update each other, and a geometry-aware structure module) and why attention and pair representations suit proteins;
- interpret pLDDT and PAE correctly, and explain why confidence is not the same as correctness;
- explain why complexes are harder than single chains, and why a confident complex model does not show that two proteins interact in the cell;
- explain at a high level what a diffusion model does and what AlphaFold3 changed;
- say what current structure predictors cannot tell us.

In Session 6 we treated structure as the answer, and we learned to inspect and compare it. Now we ask how that answer can be predicted from sequence. This session closes the course, and nearly every earlier session contributes a piece to it:

Table 12.1: Where the ideas of modern structure prediction come from in this course.

| From         | Idea used in modern structure prediction                                          |
|--------------|-----------------------------------------------------------------------------------|
| Sessions 4-5 | alignments and profiles: the evolutionary record of a family                      |
| Session 6    | structure comparison (TM-score), distance and contact maps, conformational states |
| Session 7    | nonlinear models, and secondary structure from local sequence                     |
| Session 8    | honest evaluation on proteins unlike the training data                            |
| Session 9    | training deep networks                                                            |
| Session 10   | pairwise maps treated as images, residual networks, invariance                    |
| Session 11   | attention and protein language models                                             |

The session starts from one question: **what information is missing if we try to predict a residue's structure from its local sequence alone?** The answer is the contacts and interactions with residues far away in the sequence. The rest of the session is therefore about where this long-range information comes from, how AlphaFold2 learned to use it, and what we can trust in the result.

## 12.2 Where does long-range structural information come from?

### 12.2.1 The folding problem

For many proteins, the sequence determines the structure (Anfinsen, 1973, [Science 181:223](#)). A protein cannot, however, find that structure by trying every conformation. With even a few states per residue, a random search would take longer than the age of the universe, while real proteins fold in milliseconds to seconds (**Levinthal's paradox**, 1969). Dill & MacCallum ([Science 338:1042](#), 2012) summarised fifty years of work as three questions, the last of which is “can we devise a computer algorithm to predict protein structures from their sequences?”

### 12.2.2 Three generations of structure prediction

1. **Physics and templates (1970s onwards).** The first approaches either simulated the atoms or borrowed from known structures. Molecular dynamics simulates the atoms, which is possible only for small, fast-folding proteins, even on special-purpose computers. **Homology modelling** copies the structure of a relative, and **threading** finds the known fold that a sequence fits best (Session 5's profile

idea applied to structures). **Rosetta** assembles the chain from short fragments of known proteins with a knowledge-based energy function. Blind tests held every two years since 1994, the **CASP** experiments (Moult et al. 1995, [Proteins 23:ii](#)), showed steady progress for proteins with templates, and little for the rest.

2. **Contact prediction (2011-2019)**. Coevolution in large protein families gave long-range contacts, then distances, cleaned up by deep networks (below).
3. **End-to-end learned geometry (2020)**. AlphaFold2 learned the whole path from alignments to atoms.

What all these approaches lacked was long-range information. Local sequence is not enough, because the same ten residues can be a strand in one protein and a helix in another (Session 7's chameleon sequences).

### 12.2.3 Coevolution: long-range restraints from evolution

The second generation found this long-range information in evolution. If two residues touch, a mutation in one is often compensated by a mutation in the other, so across a large family of homologs the two residues **co-evolve**. The idea is old (Göbel et al. 1994, [Proteins 18:309](#)), and three developments made it work two decades later:

1. **Better statistics**. Simple correlation between two alignment columns is swamped by phylogeny and by indirect effects (A-B and B-C contacts make A and C look coupled). **Direct coupling analysis** (DCA; Weigt et al. 2009, [PNAS 106:67](#); Morcos et al. 2011, [PNAS 108:E1293](#)) solves this by fitting one global statistical model to the whole alignment. The model is the maximum-entropy model that reproduces the single-column and column-pair frequencies of the alignment (Session 1), and it separates direct couplings from indirect ones. The idea itself is a decade older and was long overlooked. Alan Lapedes and colleagues identified both problems, phylogeny and indirect couplings, in 1999 (Lapedes et al., [IMS Lecture Notes 33:236](#)). In a 2002 manuscript they fitted this same maximum-entropy model to protein alignments and showed that it predicts contacts better than simple correlation, and also the effects of combined mutations. The manuscript was not published in a journal and only appeared on arXiv in 2012 (Lapedes, Giraud & Jarzynski, [arXiv:1207.2484](#)), after DCA had made the idea famous. Because these pairwise couplings capture epistasis, the same kind of model also predicts mutation effects better than an independent profile (EVmutation; Hopf et al. 2017, [Nat. Biotechnol. 35:128](#)). These are the interactions Session 9 measured in GB1, now learned from an alignment.
2. **Bigger families**. Sequence databases grew far faster than structure databases, so families became large enough for the statistics (hundreds to thousands of effective sequences).
3. **Machine learning on top**. Contacts are not independent pixels. Helices give bands, and strand pairs give lines along or across the diagonal, so a network that looks at a whole neighbourhood of the map (Session 10's convolution) can clean up the noisy coevolution signal. In the notebook,  $3 \times 3$  windows centred on real contacts show only 146 of the 256 possible patterns. When the centre pair is in contact, a neighbouring pair is in contact 23-42% of the time, against a base rate

of 3.0%. The first discussion problem asks you to read these patterns, which are exactly what a learned **pair representation** exploits.

The figure shows this progress on one protein, the bacterial response regulator CheY, using an alignment of 11,084 homologs. For each method, the 128 highest-scoring residue pairs ( $L = 128$ , the length of the protein) are plotted over the true contacts. Precision rises from 0.49 (mutual information) to 0.73 (DCA), 0.89 (a deep-learning contact predictor) and 0.99 (the contacts in the AlphaFold2 model):

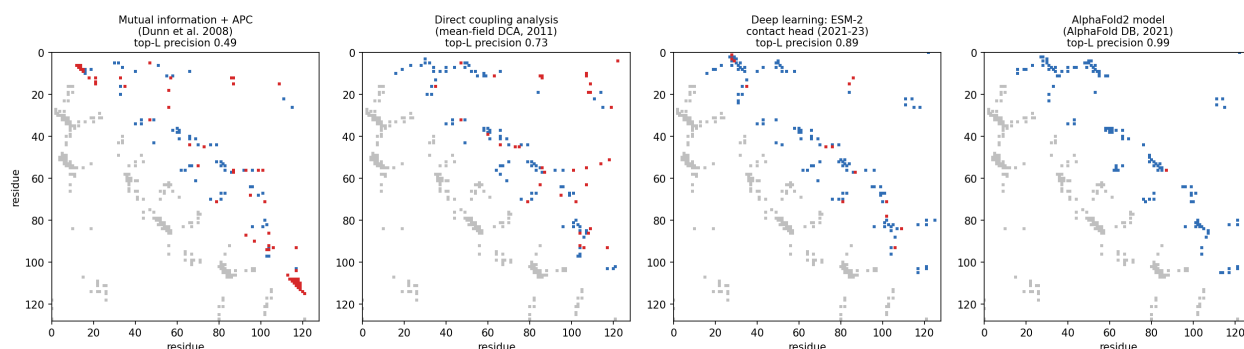
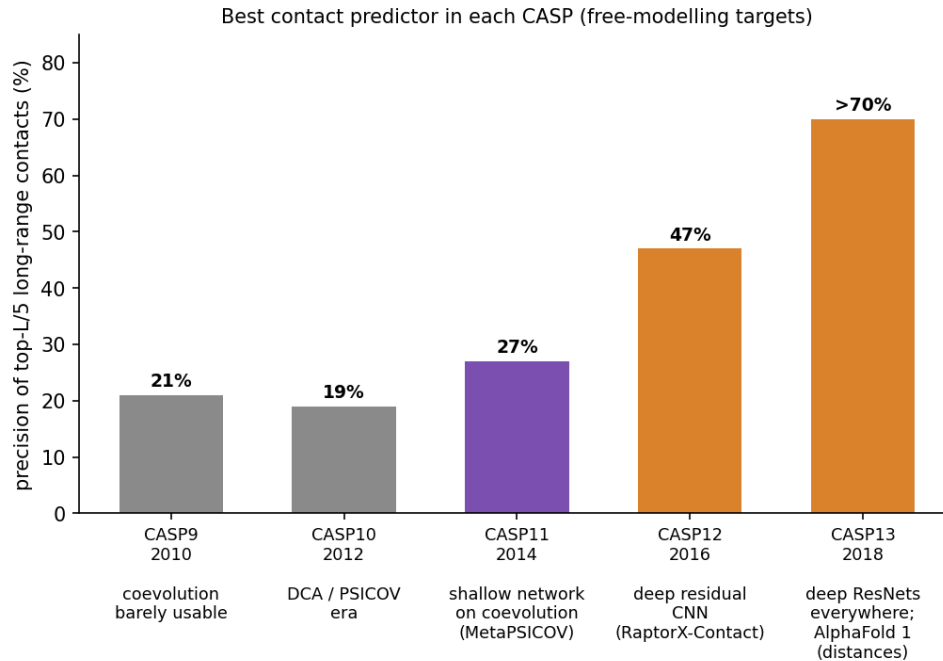


Figure 12.1: Contact prediction for CheY (PDB 3CHY), computed in the notebook. Upper triangle: each method's 128 highest-scoring residue pairs (sequence separation of at least 6), blue if the pair is a true contact, red if not. Lower triangle, grey: the true contacts ( $C\beta$  atoms within 8 Å). The deep-learning panel uses the contact head of the ESM-2 language model (650M parameters), a stand-in for the contact predictors of 2017-2019, whose software no longer installs. Mutual information's false positives cluster in blocks, from phylogeny and conservation; DCA's are scattered.

The next step was from contacts to **distances**. AlphaFold 1 (Senior et al. 2020, *Nature* 577:706) and trRosetta (Yang et al. 2020, *PNAS* 117:1496) predicted distance distributions between residue pairs with deep residual networks, and they folded proteins by minimising against these distributions. The blind CASP assessments show how sudden the change was:



CASP14 (2020): top-5 average 64%, no further gain; AlphaFold 2 made contact prediction a by-product of full 3D prediction.

Figure 12.2: Precision of the best group's top-L/5 long-range contact predictions on free-modelling targets in each CASP, from the official assessment papers (CASP10: Monastyrskyy et al. 2014; CASP11: 2016; CASP12: Schaarschmidt et al. 2018; CASP13: Shrestha et al. 2019; CASP14: Ruiz-Serra et al. 2021).

From CASP9 to CASP11 (2010-2014), contact precision stayed around 20-27%. It then reached 47% in CASP12 (2016) and more than 70% in CASP13 (2018). In CASP14, AlphaFold2 made contacts a by-product of predicting the whole structure directly.

### **i** Our own lineage: from PconsC to fold-and-dock

The Stockholm group (this course's teachers) contributed to this generation:

- **PconsC2** (Skwark, Raimondi, Michel & Elofsson, 2014, [PLoS Comput Biol 10:e1003889](#)) used "a deep learning approach to identify protein-like contact patterns", namely random forests applied iteratively over an  $11 \times 11$  window of first-round predictions. It helped most for  $\beta$ -sheet proteins.
- **PconsFold** (Michel et al., 2014, [Bioinformatics 30:i482](#)) folded proteins from these contacts, improving model TM-scores by 33% over EVfold. **PconsC3** (2017) predicted accurate contacts for "more than 4000" Pfam families of unknown structure.
- **RaptorX-Contact** (Wang et al. 2017, [PLoS Comput Biol 13:e1005324](#), Session 10) treated the whole contact map as an image for a deep residual network. **PconsC4** (Michel, Menéndez Hurtado & Elofsson, 2019, [Bioinformatics 35:2677](#)) made this fast and "hassle-free", with no GPU needed.

- **AlphaFold 1** (Senior et al. 2020, first shown at CASP13 in 2018; [Nature 577:706](#)) and **trRosetta** (Yang et al., 2020, [PNAS 117:1496](#)) predicted **distance distributions** (and, in trRosetta, orientations) instead of binary contacts. They folded by minimising against them. By CASP13 most single-domain proteins could be predicted “quite accurately”.

## **i** Docking before AlphaFold

Historically, complexes were predicted by docking separately determined or modelled components, whereas modern models predict the chains and their interface jointly. The older route is summarised here for reference.

Docking asks how two proteins of known structure fit together. It followed its own path, along the same three routes as structure prediction.

- **Shape and physics.** Katchalski-Katzir et al. (1992, [PNAS 89:2195](#)) put both proteins on a 3D grid. They scored shape complementarity for every relative translation at once with the fast Fourier transform (FFT), and they sampled the rotations exhaustively. This became the standard engine, and later methods refined it:
  - **GRAMM** used low-resolution grids that tolerate errors in the structures (Vakser 1995, [Protein Eng 8:371](#));
  - **ZDOCK** added electrostatics and desolvation (Chen, Li & Weng 2003, [Proteins 52:80](#)).

All of them generate thousands of candidate poses, so the hard part is ranking them. This is the ranking problem again.

- **Data.** **HADDOCK** drives docking with experimental information, such as mutations or NMR data, marking the interface residues (Dominguez, Boelens & Bonvin 2003, [J Am Chem Soc 125:1731](#)).
- **Templates.** Complexes can also be modelled from a related complex of known structure. Templates “are available to model nearly all complexes of structurally characterized proteins” (Kundrotas, Zhu, Janin & Vakser 2012, [PNAS 109:9438](#)). Petras Kundrotas, the first author, teaches on this course.

**CAPRI**, the Critical Assessment of PRedicted Interactions, has been docking’s CASP since 2001 (Janin et al. 2003, [Proteins 52:2](#)). DockQ was developed to score its models (Basu & Wallner 2016, [PLoS One 11:e0161879](#)). Until 2018 the picture was clear. Targets with a template were predicted “very good to excellent”, but for 11 difficult targets without one, “medium and high quality models [were] submitted for only 3” (Lensink et al. 2019, CASP13-CAPRI, [Proteins 87:1200](#)).

**Coevolution across an interface.** Residues in contact between two proteins co-evolve too, provided we can pair up which sequence in one family interacts with which sequence in the other. This is easy for bacterial proteins encoded next to each other, but hard in general. Ovchinnikov, Kamisetty & Baker (2014, [eLife 3:e02030](#)) and Hopf et al. (2014, [eLife 3:e03430](#)) used such paired alignments to predict interfaces. Combined with folding, this idea led to fold and dock:

- **Fold and dock.** Joining the alignments of two proteins lets the folding machinery predict how they bind. Pozzati et al. (2022, [Bioinformatics 38:954](#)) did this with trRosetta, and their PconsDock score separated correct from incorrect models 98% of the time. With AlphaFold2 the same idea became FoldDock (Section [12.4](#)), and folding and docking had become one problem.

### 12.2.3.1 Further reading

- Dill & MacCallum (2012), [The protein-folding problem, 50 years on](#)
- Lapedes, Giraud & Jarzynski (written 2002, posted 2012), [Using sequence alignments to predict protein structure and stability with high accuracy](#)
- Morcos et al. (2011), [Direct-coupling analysis of residue coevolution captures native contacts across many protein families](#)
- Skwark et al. (2014), [Improved contact predictions using the recognition of protein like contact patterns](#)

## 12.3 How AlphaFold2 combines the course's ideas

AlphaFold2 (Jumper et al. 2021, [Nature 596:583](#)) was “the first computational method that can regularly predict protein structures with atomic accuracy even in cases in which no similar structure is known”. At CASP14, its median backbone accuracy was 0.96 Å, against 2.8 Å for the next best method. It is trained end to end to map **sequence-derived, evolutionary and template information** (the target sequence, an alignment of its homologs, and optionally structures of relatives) to atomic coordinates. This distinguishes it from single-sequence methods such as ESMFold, which we meet below.

### 12.3.1 The data flow

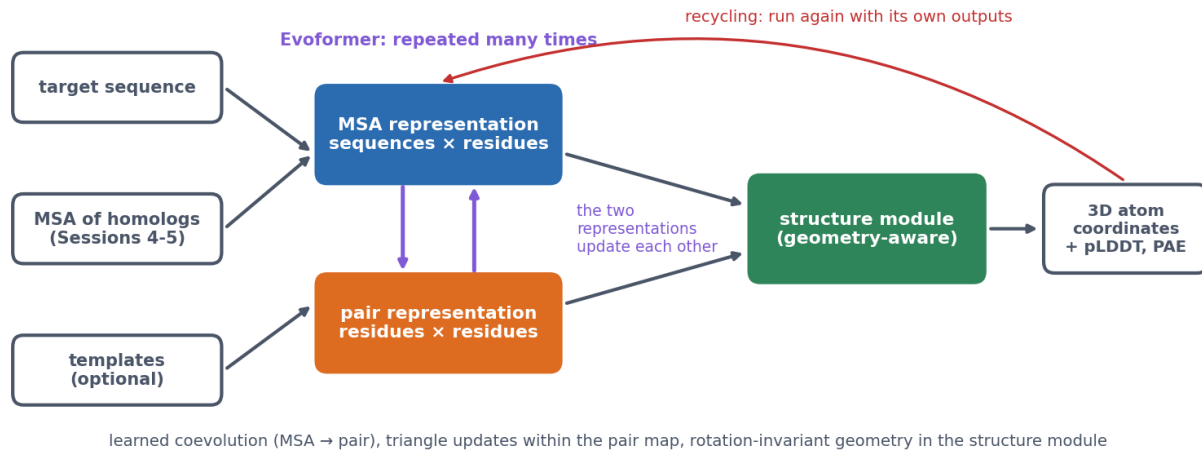


Figure 12.3: The data flow of AlphaFold2, simplified. Inputs are embedded into an MSA representation and a pair representation, which the Evoformer updates many times, each informing the other. A geometry-aware structure module turns them into 3D coordinates with confidence scores, and the whole network is run again on its own outputs (recycling).

AlphaFold2 carries two representations through the whole network:

- the **MSA representation**, with a vector for every residue of every aligned sequence;
- the **pair representation**, with a vector for every pair of residues. It is a learned, much richer version of the contact and distance maps above (and of the distance map of  $\beta$ -globin in Session 6).

The **Evoformer** updates both representations, many times over. Three of its operations carry the main ideas:

- **MSA attention.** Within each aligned sequence, residues attend to each other (Session 11's attention, biased by the current pair representation). Within each column, sequences attend to each other, so the network learns which homologs are informative for each position.
- **MSA → pair.** For every pair of columns, the way the two columns vary together across the family updates the pair representation. This is a learned version of coevolution.
- **Triangle updates.** Pair  $(i, j)$  is updated with information passing through every third residue  $k$ . Residue-pair relationships are not independent: if  $i$  is close to  $k$  and  $j$  is close to  $k$ , that constrains how  $i$  and  $j$  can be related. The authors were inspired by geometric constraints such as the triangle inequality on distances, but the updates are learned operations, not rules that enforce it.

The paper's own figure shows how these operations fit together in one Evoformer block (panel a). Read it from left to right. The MSA representation (top row;  $s$  sequences  $\times$   $r$  residues  $\times$   $c$  channels) passes through row-wise attention, which is biased by the pair representation (the arrow from below), column-wise attention and a **transition**, a small feed-forward network applied to each vector. The **outer product mean** then sends what the alignment shows down to the pair representation (bottom row). This passes through the four triangle operations and its own transition. Every step adds its result to its input (the  $\oplus$  symbols), the residual connections of Session 11's Transformer. The block is repeated 48 times, each with its own weights. Panels b and c show the pair representation as the edges of a graph, and which edges each triangle update combines to update the edge  $ij$ :

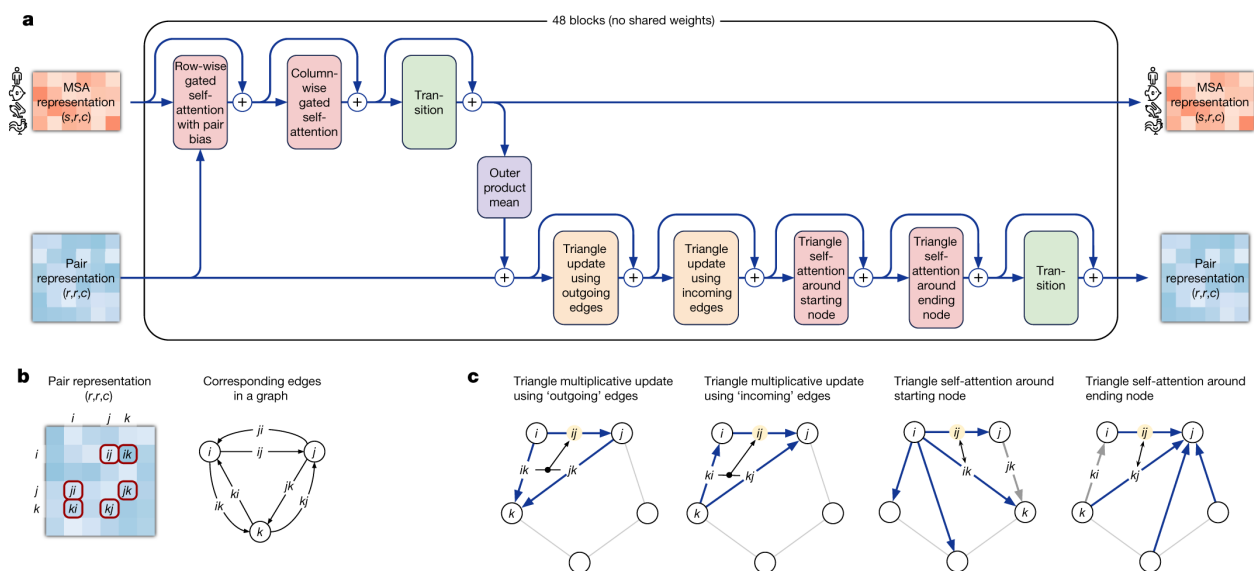


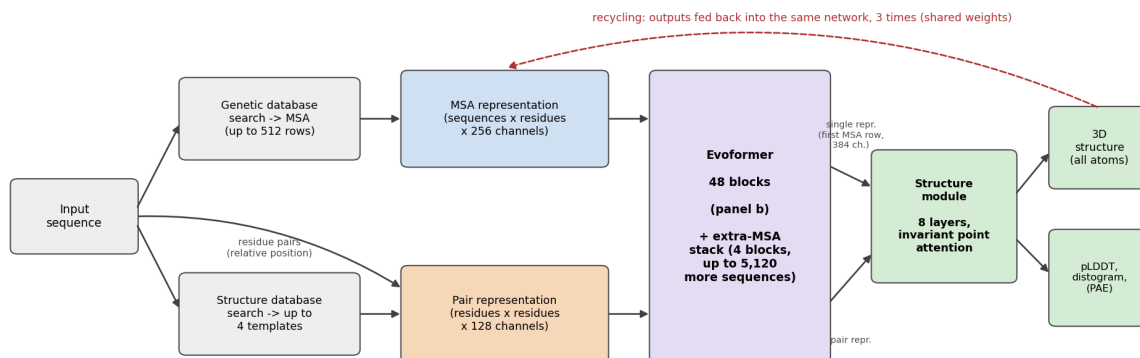
Figure 12.4: The Evoformer, from the AlphaFold2 paper. (a) One Evoformer block; the shape of each array is given in parentheses ( $s$  sequences,  $r$  residues,  $c$  channels). (b) The pair representation read as directed edges in a graph. (c) The two triangle multiplicative updates and the two triangle self-attention operations; the edge being updated is  $ij$ . Reproduced from Jumper et al. (2021), [Nature 596:583](#), Fig. 3a-c (panels d-f left out), under a [CC BY 4.0](#) licence.

The **structure module** then builds coordinates. Each residue is a rigid **frame** (the position and orientation of its backbone). **Invariant point attention** reasons about these frames in 3D in a way that does not depend on how the whole protein happens to be rotated or translated. This is the invariance idea of Session 10, now in three dimensions. Finally, **recycling** runs the whole network again with its own outputs as extra inputs, which refines the prediction a few times.

In short, AlphaFold2 kept the ingredients of the second generation (alignments, co-evolution, contacts, distances, physical plausibility) and turned each into a learnable operation inside one network, trained on the final coordinates.

## i For reference: AlphaFold2 in detail

### a The whole network



### b One Evoformer block (repeated 48 times)

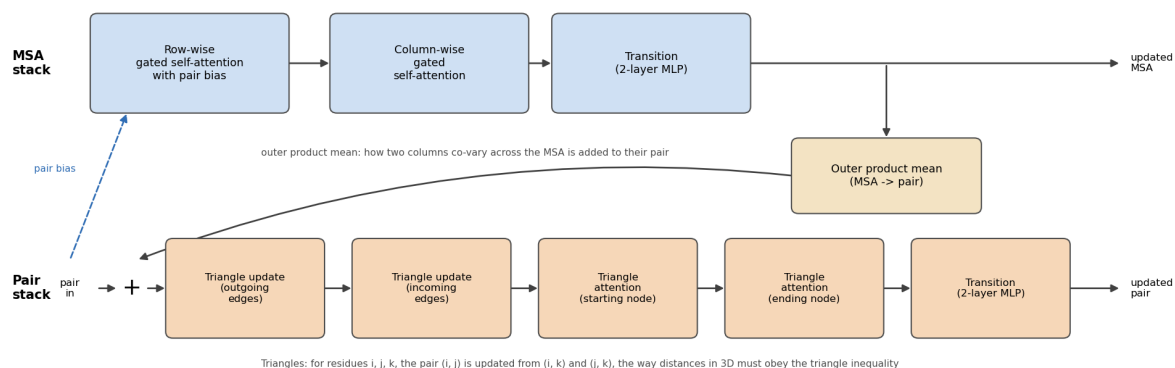


Figure 12.5: AlphaFold2's architecture, drawn for this course after Jumper et al. (2021). (a) The whole network. (b) The operations of one Evoformer block, in the order they are applied.

The numbers below are from the paper and its Supplementary Information.

#### 1. Inputs. Three kinds of information go in.

- **An MSA.** Genetic databases are searched for homologs. At inference the network reads a representation of 512 rows: cluster centres of the alignment, plus template information added as extra rows. Up to 5,120 further sequences enter through a lighter **extra-MSA stack** (4 blocks, with a cheaper "global" column attention).
- **Templates.** Up to 4 structures of related proteins, found in the PDB. Only two of the five released models use them, and the network works well without them.
- **The sequence itself,** including the relative position of every pair of residues along the chain.

These are embedded into two representations that are carried through the whole network:

- the **MSA representation**: one 256-number vector for every residue in every sequence (sequences  $\times$  residues  $\times$  256);
- the **pair representation**: one 128-number vector for every pair of residues (residues  $\times$  residues  $\times$  128). This is a learned, much richer version of Section 12.2's contact and distance maps.

**2. The Evoformer: 48 blocks.** Each block updates both representations and lets them talk to each other. In order (figure, panel b):

1. **Row-wise gated self-attention with pair bias.** Within each aligned sequence, every residue attends to every other residue (Session 11's attention). The attention weights get a bias from the pair representation: what the network currently believes about residues  $i$  and  $j$  steers how much they attend to each other.
2. **Column-wise gated self-attention.** Within each column, every sequence attends to every other sequence. The network learns which homologs are informative for this position.
3. **MSA transition.** A small two-layer network applied to every vector.
4. **Outer product mean.** For every pair of columns ( $i, j$ ), the outer product of their vectors is averaged over all sequences and added to pair ( $i, j$ ). This is the learned version of coevolution: how two columns vary together across the family.
5. **Triangle multiplicative updates**, "outgoing" and "incoming" edges. Pair ( $i, j$ ) is updated from the pairs ( $i, k$ ) and ( $j, k$ ) for every third residue  $k$ . In the authors' words, "many constraints must be satisfied including the triangle inequality on distances. On the basis of this intuition, we arrange the update operations on the pair representation in terms of triangles".
6. **Triangle self-attention**, around the starting node and around the ending node. The same triangle idea with attention: pair ( $i, j$ ) attends to all pairs ( $i, k$ ), with a bias from pair ( $j, k$ ).
7. **Pair transition.** A two-layer network on every pair vector.

Every step is added to its input (residual connections, Session 10's ResNet idea) and uses layer normalisation. Dropout is applied during training. The MSA Transformer (Rao et al. 2021) used the same row-and-column attention at about the same time. The idea of learning directly from raw alignments had been explored in rawMSA (Mirabello & Wallner, 2019, [PLoS One 14:e0220182](#)).

**3. The structure module: 8 layers.** It turns the representations into coordinates. Its input is the pair representation, plus a **single representation**: the first row of the MSA representation (the target sequence), projected to 384 numbers per residue.

- Each residue is a **rigid frame**: the position and orientation of its backbone N, C $\alpha$  and C atoms, as a rotation and a translation. All frames start at the origin with the same orientation, which the authors call "black hole initialisation".
- **Invariant point attention (IPA)** updates the single representation. It is attention that combines three terms: the residue vectors, the pair representation, and 3D points attached to each residue's current frame. Because

distances between points do not change when the whole protein is rotated or moved, the result is **invariant** to global rotations and translations. The network cannot cheat by learning where proteins usually sit in space.

- After each layer, every frame is updated by a small rotation and translation. The 8 layers share their weights, like 8 steps of one refinement.
- Finally, 7 torsion angles per residue (backbone  $\omega$ ,  $\phi$ ,  $\psi$  and side-chain  $\chi_1$ - $\chi_4$ ) are predicted and turned into all-atom coordinates.

**4. Recycling.** The whole network is run again with its own outputs fed back as extra inputs: the predicted structure, the pair representation and the first MSA row. At inference it runs 4 times, that is, 3 recycles, with the same weights. Each pass refines the last.

#### **5. Training.**

- **The main loss is FAPE**, frame aligned point error. Every atom is compared with the true structure in the local frame of every residue, and errors are clamped at 10 Å. Unlike a global RMSD, this penalises wrong local geometry everywhere, and it cannot be satisfied by a mirror image.
- **Auxiliary losses** keep the intermediate representations meaningful:
  - a **distogram** head predicts binned residue-residue distances from the pair representation (64 bins, 2-22 Å): AlphaFold 1's output, now just a side task;
  - a **masked-MSA** head predicts residues hidden in the alignment (15%), like Session 11's BERT;
  - a **pLDDT** head learns to predict the model's own local accuracy (50 bins);
  - fine-tuning adds penalties for impossible geometry, such as clashes and bad bond lengths.
- **Data.** PDB structures released before 30 April 2018. The network was then used to predict about 350,000 sequences from Uniclust30, and these predictions were added to the training data (**self-distillation**).
- **Scale.** Training used crops of 256 residues, then 384 in fine-tuning, on 128 TPU v3 cores: about a week of initial training plus about four days of fine-tuning in the simplified protocol described in the paper.
- **PAE** was added by a further short fine-tuning, to the "pTM" models, with a head that predicts the aligned error between every pair of residues (64 bins, 0-31.5 Å).

**6. Five models.** Five networks were trained with different random seeds, two with templates and three without. All five are run, and the predictions are ranked by pLDDT.

#### **12.3.1.1 Further watching**

*Highly accurate protein structure prediction with AlphaFold* (Simon Kohl, DeepMind),

a detailed walk through AlphaFold2's architecture:

Video: [Highly accurate protein structure prediction with AlphaFold, Simon Kohl](#)

### 12.3.2 Confidence: pLDDT and PAE

A predicted structure is only useful if we know how far to trust it. AlphaFold2's confidence measures are therefore as important as the structure itself, and they answer two different questions:

- **pLDDT** (0-100, per residue) asks *how trustworthy is the local structure around this residue?* Above 90 is very high, 70-90 confident, and below 50 very low.
- **PAE** (predicted aligned error, per pair of residues) asks *if I align the model on residue (or region) A, how certain is the position of B relative to A?* Low PAE within a domain and high PAE between two domains means that each domain may be right while their relative orientation is unknown. This matters for multi-domain proteins, for flexible regions and, below, for complexes.

**Low pLDDT means low confidence, not necessarily disorder.** Intrinsic disorder is one common reason, but other reasons are too little evolutionary information, several conformational states, an uncertain domain orientation, or simply the limits of the model. Moreover, **confidence is not correctness**, because a confidence score is the model's own estimate, learned from its training data. The second discussion problem tests how well pLDDT tracks the real error. Checking this on famous proteins is harder than it looks, because well-studied proteins are likely to have close relatives in the training data (Session 8).

**In the notebook.** The AlphaFold DB model of human  $\beta$ -globin (mean pLDDT 97.2) matches the crystal structure 4HHB with TM-score 0.988 and RMSD 0.51 Å. For p53, 30% of the residues have pLDDT below 50. The DNA-binding domain averages 95.3, while the N-terminal transactivation region (47.6) and the C-terminus (45.8) are not confident. For p53 that low confidence matches biology, since these regions are known from experiments to be intrinsically disordered:

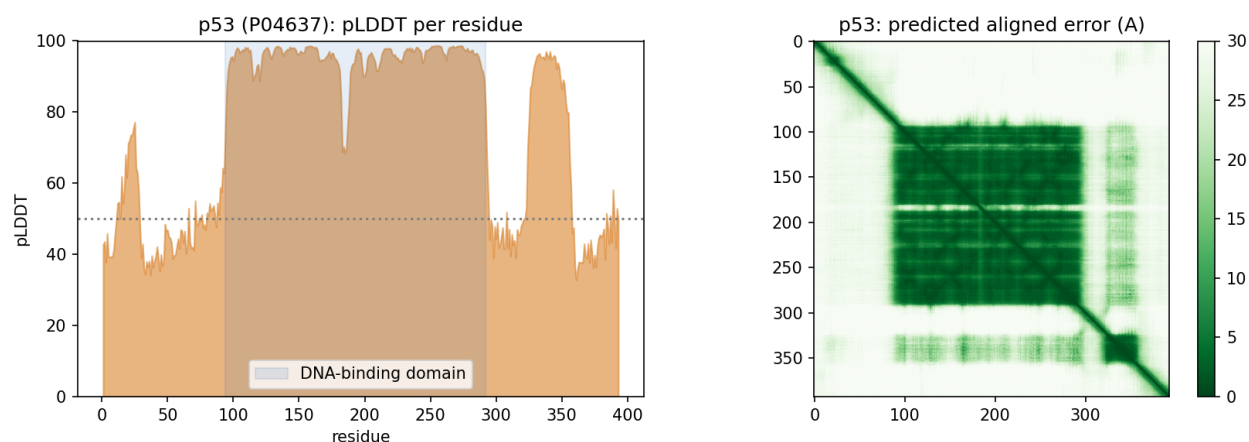


Figure 12.6: p53 (P04637) in the AlphaFold DB. Left: pLDDT per residue, with the DNA-binding domain shaded. Right: predicted aligned error. Only the DNA-binding domain forms a confident block.

**For fun:** in *Protein Fighter*, a game from Sergey Ovchinnikov’s lab, the health bar is the fighter’s pLDDT and the small map under it is its PAE.

**Scale and tools.** AlphaFold predicted almost the entire human proteome (Tunyasuvunakool et al. 2021, *Nature* 596:590), and the AlphaFold Database made predicted structures available for essentially every protein in UniProt (Varadi et al. 2024, *NAR* 52:D368). **ColabFold** (Mirdita et al. 2022, *Nat Methods* 19:679) replaced the slow alignment search with MMseqs2 and put AlphaFold in a Colab notebook. **ESMFold** predicts from a single sequence, using the ESM-2 language model (Session 11) instead of an alignment. In the notebook it predicts Ephrin-A5 in seconds, with TM-score 0.941 against the crystal structure (1SHW). In 2024, Demis Hassabis and John Jumper shared half of the [Nobel Prize in Chemistry](#) “for protein structure prediction”, and the other half went to David Baker “for computational protein design”.

**Known limits** (lecture). AlphaFold cannot tell whether a sequence folds at all, or how a mutation changes the structure. It may or may not find alternative conformational states. The arrangement of domains in large flexible or repeated proteins is often arbitrary. It “does not know what a membrane is”. In a protein-DNA complex it can place another protein where the DNA should be.

## 12.4 From single chains to interactions

### 12.4.1 Complexes are harder

AlphaFold2 was trained on single chains, yet it can predict complexes when the chains are given together (FoldDock; Bryant, Pozzati & Elofsson 2022, *Nat Commun* 13:1265). Later, AlphaFold-Multimer (Evans et al. 2021, [bioRxiv](#)) was retrained on complexes. Complexes are harder than single chains for four reasons:

1. **Paired evolutionary information helps interfaces.** Residues in contact across an interface co-evolve too, if the alignments of the two families can be paired (which sequence interacts with which).
2. **Interfaces need their own confidence.** AlphaFold-Multimer introduced **ipTM**, a confidence score for the interfaces, and FoldDock's **pDockQ** combines the interface pLDDT with the number of interface contacts. A model is compared with the true complex using **DockQ**, which runs from 0 to 1, with at least 0.23 counting as acceptable.
3. **Interface confidence is harder than chain confidence.** Each chain can be right while their relative placement is wrong, which is exactly what the PAE between chains shows.
4. **A plausible complex is not evidence of interaction.** A structure predictor can suggest **how** two supplied molecules could interact. It does not establish **whether** they interact in the cell, or when. That is one of the most important messages of this session.

Predicted complexes can still lead to biology when they are combined with other evidence. All-against-all modelling of 14 fertilisation proteins pointed to “a pentameric complex involving sperm IZUMO1, SPACA6, TMEM81 and egg JUNO, CD9” at the egg-sperm fusion synapse (Elofsson et al. 2024, [eLife 13:RP93131](#)).

### 12.4.2 The ranking problem

Generating a good model has become easier than *knowing* which of your models is good. In short, **sampling quality is not ranking quality**, as two examples show:

- **Antibody-antigen complexes** have no coevolution between the partners. On 110 complexes unseen in training, sampling 200 models per complex raised the average DockQ of the *best* model “from below 0.3 to above 0.5”. The models’ own confidence, however, often “cannot identify the best predicted structures”, so the top-ranked model is much worse than the best one (Fromm, Ludaic & Elofsson 2026, [Bioinformatics 42:btag136](#)).
- **Confidence depends on what you include.** Trimming a full-length construct to its interacting domains raises ipTM “even though the structure prediction of the interaction is unchanged”. **ipSAE** counts only residue pairs with good PAE, and it separates true from false complexes better (Dunbrack 2025, [bioRxiv](#)).

The common thread is Session 8’s lesson on a grand scale. Performance on examples similar to the training data says little about genuinely new ones, and confidence scores do not always know the difference.

### 12.4.3 When can a complex prediction be trusted? A practical heuristic

How, then, can we decide when to trust a complex prediction? The lecture sorts protein interactions into four regions. This is a heuristic for thinking about the question, not an established classification:

Table 12.2: Four regions of protein-protein interactions, by how well current methods handle them (a heuristic).

| Region                          | Character                                                                 | Example                                                | Current methods                                           |
|---------------------------------|---------------------------------------------------------------------------|--------------------------------------------------------|-----------------------------------------------------------|
| I, tractable                    | Stable interfaces, strong coevolution                                     | Obligate heterodimers (Ephrin-A5:EphB2)                | Work well                                                 |
| II, ambiguity and competition   | A plausible interface is easy; the right partner or stoichiometry is hard | Proteasome paralogs; Izumo1 and its competing partners | Struggle to rank the correct partner                      |
| III, missing coevolution        | No joint selective pressure                                               | Antibody-antigen, host-pathogen                        | AlphaFold3-class models help, partly through memorisation |
| IV, missing context or dynamics | Transient, conditional, disordered                                        | p53 transactivation domain binding MDM2                | Static structure prediction is the wrong tool             |

The lab tests this heuristic. Each group makes one AlphaFold3 Server prediction per region and checks whether the model's confidence tracks how hard each problem really is.

**What the scores do not tell you.** pLDDT is the confidence in the local structure, PAE the confidence in the relative position of two parts, and ipTM or ipSAE the confidence in an interface. All of them are the model's own estimates for the chains it was given. None of them shows that the proteins actually bind, that they bind each other rather than a competitor that was left out, or that the predicted complex is the one formed in the cell. A confident interface is a candidate hypothesis, to be tested with other evidence.

#### 12.4.3.1 Further reading

- Jumper et al. (2021), [Highly accurate protein structure prediction with AlphaFold](#)
- Bryant, Pozzati & Elofsson (2022), [Improved prediction of protein-protein interactions using AlphaFold2](#)
- Fromm, Ludaic & Elofsson (2026), [Evaluating deep learning based structure prediction methods on antibody-antigen complexes](#)

## 12.5 Generative structure modelling

### 12.5.1 Diffusion in one page

The newest structure predictors are generative models, and most of them are built on diffusion. A **diffusion model** learns to generate data by learning to remove noise, in three steps:

1. **Forward process.** Real data are noised by adding a little Gaussian noise step by step, until only noise is left. This step needs no learning.
2. **Training.** A network is shown a noisy example and its step number, and it is trained to predict the noise that was added.
3. **Generation.** Starting from pure noise, the network is applied many times, and each step removes a little noise.

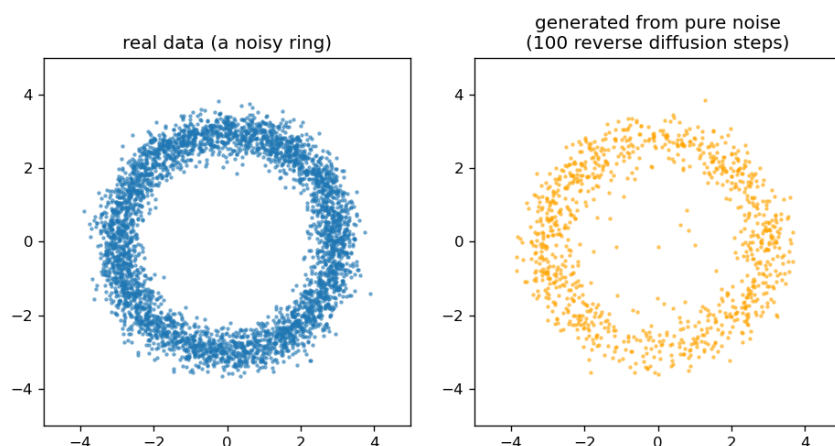


Figure 12.7: Left: points on a noisy ring. Right: 1,000 points generated from pure Gaussian noise by 100 learned denoising steps, in the notebook.

The idea goes back to Sohl-Dickstein et al. (2015). Ho, Jain & Abbeel ([arXiv:2006.11239](https://arxiv.org/abs/2006.11239), 2020) made it practical, and it is behind image generators such as DALL-E. Two relatives deserve one sentence each. **Flow matching** learns a smooth path from noise to data and needs fewer steps. A **variational autoencoder** generates data by decoding samples from a learned low-dimensional space. The third discussion problem asks whether a diffusion model learns a distribution or memorises its training examples.

### 12.5.2 AlphaFold3: what changed

AlphaFold3 (Abramson et al. 2024, [Nature 630:493](https://doi.org/10.1038/s41586-024-0476-4)) brings diffusion into structure prediction. It predicts “the joint structure of complexes including proteins, nucleic acids, small molecules, ions and modified residues”. Conceptually, four things changed:

- **Broader chemistry.** Its published model was designed to represent proteins, nucleic acids, ligands, ions and modified residues jointly, with residues, nucleotides and single ligand atoms all as tokens.
- **A pair-focused trunk.** The **Pairformer** replaces the Evoformer. It works mostly on the pair representation and much less on the alignment.
- **Generative coordinates.** A **diffusion module** replaces the structure module. It starts from random atom positions and denoises them, conditioned on the trunk.
- **All atoms at once.** The model works on all atoms at once, which is what lets it handle any molecule made of atoms.

Diffusion brings sampling into structure prediction, and this has two consequences. First, many seeds can be tried, which brings back the ranking problem above. Second, the model can **hallucinate** plausible-looking structure in regions that should be disordered, which the authors had to counter in training. At CASP16, “AlphaFold3 performs slightly better for protein complexes. However, when massive sampling is applied to AlphaFold2, the difference disappears” (Elofsson 2026, [Proteins 94:154](#)).

#### 12.5.2.1 Where the field is in 2026 (optional, and will age quickly)

AlphaFold3 was released without training code, so **open reproductions** followed quickly:

- **Boltz-1:** fully open (MIT licence).
- **Boltz-2:** adds binding-affinity prediction, approaching free-energy perturbation while being “at least 1000× more computationally efficient”.
- **Chai-1:** can run without alignments.
- **Protenix.**
- **OpenFold3:** preview release.
- **RoseTTAFold All-Atom** (Krishna et al. 2024, [Science 384:eadl2528](#)) is an independent architecture.
- **RNA.** Ten methods were benchmarked on RNAs and RNA complexes not seen in training. AlphaFold3 and Boltz-1 lead, but “accuracy depends greatly on similarity to the training set”, and the methods “recognize recurring motifs rather than generalize to novel RNA folds” (Ludaic & Elofsson 2026, [NAR 54:gkag813](#)).

The field moves fast, and three systems from 2026 show three different ways of moving beyond AlphaFold3.

- **ESMFold2** (Candido et al. 2026, [bioRxiv](#); Biohub; MIT licence, open weights). It joins a large protein language model, ESMC (the successor to Session 11’s ESM-2), to an AlphaFold3-style diffusion structure module, and the whole model has about 6.6 billion parameters. The original ESMFold was built for single chains, whereas ESMFold2 predicts complexes of proteins, nucleic acids and ligands, from single sequences or with an optional alignment. The authors report that it “exceeds the performance of established methods for biomolecular complex

prediction across benchmarks, including for the interactions of antibodies with their targets” (Candido et al. 2026). The notebook runs it on one complex from each of the four regions above, with five seeds each (figure). For every complex, the seed with the highest ipTM is a good model (DockQ 0.61-0.85). For the 1996 D1.3 antibody, however, three of five seeds put the antibody in the wrong place. Moreover, no single ipTM threshold separates right from wrong across complexes, since correct Fab : MHC-I models (ipTM 0.73) score below wrong D1.3 models (0.78).

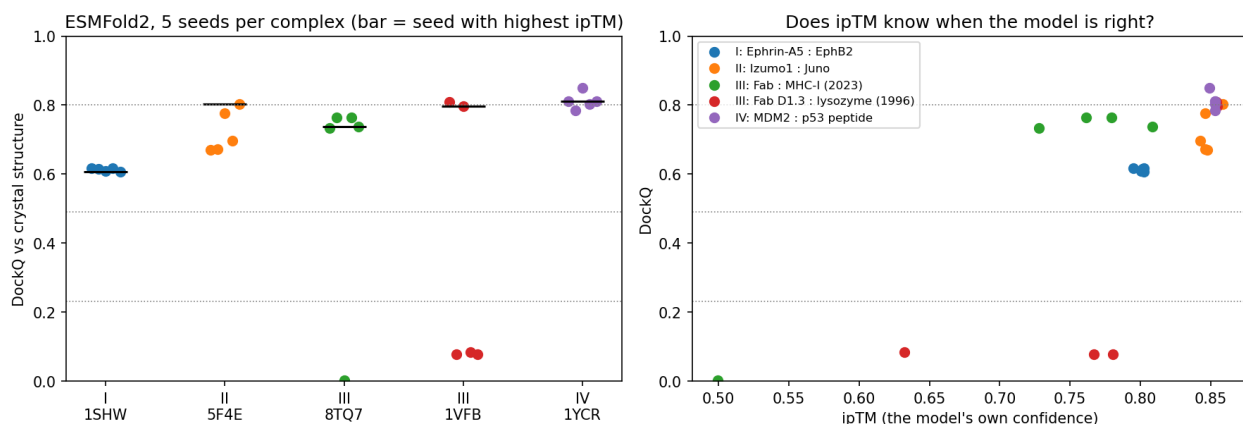


Figure 12.8: ESMFold2 on one complex per region, five seeds each, scored with DockQ against the crystal structure (0.23 acceptable, 0.49 medium, 0.80 high). Left: all seeds; the bar marks the seed with the highest ipTM. Right: ipTM against DockQ. These are well-known complexes; p53 : MDM2 has been in the PDB since 1996, so a good model here does not show that Region IV is solved.

- **IsoDDE**, the Isomorphic Labs Drug Design Engine (Isomorphic Labs 2026, [technical report](#)), is proprietary and comes from DeepMind’s drug-discovery sister company. The company claims:
  - on the Runs N’ Poses protein-ligand benchmark, more than double AlphaFold3’s accuracy for the ligands least similar to the training data;
  - on 334 new antibody-antigen complexes, 2.3 times AlphaFold3 and 19.8 times Boltz-2 at high accuracy (DockQ > 0.8);
  - binding-affinity predictions that can beat physics-based free-energy methods;
  - binding pockets found from sequence alone.

There is no code, no model and no independent test, so these are claims, not results. Notice where the claimed gains are: on the examples unlike the training set, where AlphaFold3 is weakest.

- **OpenDDE (company version)** (2026, [GitHub](#), Apache licence), developed by Aureka Research, is an open-source, all-atom biomolecular foundation model. It turns co-folding into a scalable engine for structure prediction, design and optimisation in drug discovery. Its developers report higher performance on antibody-antigen complexes.

- **OpenDDE (single-developer version)** (2026, [GitHub](#), MIT licence) is a community project by a single developer, not affiliated with Isomorphic Labs. It is not a new model. Instead, it chains existing open tools behind one web interface:

- AlphaFold3 for complexes;
- P2Rank for pockets;
- Boltz-2 for affinity;
- ImmuneBuilder for antibodies;
- ChEMBL and RDKit for known ligands;
- a language-model assistant.

It has no benchmarks of its own. What it shows is what an “engine” is: a pipeline of separate models, each of which can, and should, be evaluated on its own.

The contrast between open and closed systems matters for science. Open tools such as ESMFold2, Boltz-2 and OpenFold3 can be tested by anyone, on data they have never seen. A closed system, in contrast, can only be judged by what its owners choose to report.

### 12.5.3 Designing new proteins

The same generative machinery can also run in reverse, from a desired structure to a new sequence. A typical design pipeline has four steps:

1. **RFdiffusion** (Watson et al. 2023, [Nature 620:1089](#)) generates a new backbone, for example one that binds a chosen target surface;
2. **ProteinMPNN** (Dauparas et al. 2022, [Science 378:49](#)) designs a sequence for that backbone;
3. **AlphaFold2 or ESMFold** refolds the designed sequence, to check that it folds into the intended structure;
4. **experiments** test whether it does, and whether it works. Refolding is only a computational consistency check, not experimental validation. Hundreds of RFdiffusion designs were characterised in the lab, including an influenza-haemagglutinin binder whose cryo-EM structure was “nearly identical to the design model”.

#### 12.5.3.1 Further reading

- Abramson et al. (2024), [Accurate structure prediction of biomolecular interactions with AlphaFold 3](#)
- Watson et al. (2023), [De novo design of protein structure and function with RFdiffusion](#)
- Elofsson & Ben-Tal (2026), [What will be the future of computational biology for macromolecules in the era of AI?](#)

### 12.5.3.2 Further watching

*Flow-Matching vs Diffusion Models explained side by side* (AI Coffee Break with Letitia), used in the diffusion lecture:

Video: [Flow-Matching vs Diffusion Models explained side by side](#)

## 12.6 The course in one line

Modern structure prediction combines alignment and evolution with neural networks, attention, pair representations and geometry. The scientific warning that goes with it is that **a model is a hypothesis with confidence estimates, not an experiment.**

## 12.7 Check your understanding

*The interactive quiz for this section is in the online version: [course.bioinfo.se](https://course.bioinfo.se)*

## 12.8 The course notebook

`notebooks/session12-alphaFold-and-generative-models-in-code.ipynb` computes the numbers on this page:

1. Contact prediction for CheY by mutual information and DCA, computed from an alignment, and compared with ESM-2 and AlphaFold2;  $3 \times 3$  contact patterns in eight real structures.
2. AlphaFold DB models of  $\beta$ -globin and p53 with pLDDT and PAE; the  $\beta$ -globin model against its crystal structure; ESMFold on Ephrin-A5; ESMFold2 models of five complexes scored with DockQ.
3. The forward noising of a diffusion model, and a small diffusion model generating points from noise.

It runs on a CPU in a few minutes. Open the notebook in [Google Colab](#).

## 12.9 The lab

**Full lab:** the lab is adapted from the previous course's AlphaFold2 lab and the 2026 AlphaFold3 Server exercise. Your group runs one of four targets on the [AlphaFold Server](#), one for each region of the four-regions heuristic on this page:

- Region I: Ephrin-A5 : EphB2 (1SHW);
- Region II: Izumo1 : Juno (5F4E);
- Region III: antibody : antigen (8TQ7);

- Region IV: p53 peptide : MDM2 (1YCR).

Work through [notebooks/session12-lab.ipynb](#), replacing every `todo(...)` call with your own code:

1. fetch your target's chain sequences from the PDB and submit them to the AlphaFold Server (20 jobs per day and account);
2. read AlphaFold DB models: pLDDT, the PAE matrix, which models are disordered, and how closely a model matches its crystal structure;
3. load your AlphaFold Server result, plot its PAE heatmap, and compute pTM, ipTM and the interface score ipSAE;
4. pool the class's results and compare ipTM with ipSAE across the four regions;
5. superimpose your model on the crystal structure in PyMOL.

### 12.9.1 Submitting this lab

This lab is administered through Canvas, not through this book. See the course's Canvas page for the current quiz, the deadline and how to submit.

## 12.10 What's next

What remains is the capstone project. It returns to predicting secondary structure, the problem this course met in Session 7, now with every tool from Sessions 5-12 at hand.

# Capstone Project: Secondary Structure Prediction

## Note

**Sources for this page:** the project brief and grading rubric are adapted from slides/Secondary\_structure\_prediction\_project\_2025.pdf (Arne's own assignment text, reused directly since it's this course's own material, not an external source); the dataset structure below was verified by downloading and inspecting the real training data linked below (1,200 real entries, three file formats, checked directly — not assumed from a folder name).

## Tertiary structure

**Why are tertiary structures selfish?** Because the amino acids are all wrapped up in themselves.

## Framing

This project is not a new topic. It's the same problem posed descriptively on Session 6, using the tools built up from Session 5 (profiles) through Session 12 (AlphaFold): design, train, and evaluate your own secondary-structure predictor. There is no single correct architecture — part of the assignment is deciding how to build the model yourself, informed by what the course covered (a linear/profile score → a perceptron → a hidden layer → a proper PyTorch model → convolutions → recurrence → attention) and by your own reading of the literature.

## The task

You are given real protein data and asked to predict, for every residue, one of three secondary-structure classes: **H** (helix), **E** (strand), or **C** (coil) — exactly the three-class problem Session 7 and Session 9 introduced with Rost & Sander's PHD method as the historical example. Your model should be able to use **both** of the input representations provided (raw sequence and PSSM profile) — comparing what each one buys you is itself a natural thing to report on.

## Training data

A real, verified dataset of 1,200 protein domains, provided as three parallel folders sharing the same base filename per entry:

```
dataset/  
├─ fasta/   # 1,200 files – plain amino-acid sequence, FASTA format  
├─ dssp/    # 1,200 files – per-residue H/E/C label string (DSSP-derived)  
└─ pssm/    # 1,200 files – CSV, one row per residue, one column per  
            # amino acid (order: A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V),  
            # values are the position-specific scoring matrix's  
            # weighted observed percentages
```

Example entry (dlg2ra\_): a 96-residue domain, its FASTA sequence, a DSSP label string of matching length (CCCCC...HHHHHHHHHH...), and a 96-row  $\times$  20-column PSSM CSV — all three files verified to correspond to the same protein.

**Download the dataset** (Google Drive — dataset.zip, ~3 MB, 1,200 entries  $\times$  3 formats).

## Deliverables

1. **A public Colab notebook** containing the code that produces your results — reproducibility (does it actually rerun and reproduce your numbers, with a fixed random seed) is graded explicitly.
2. **A short written report as a single PDF**, structured as a research paper: introduction, materials/methods, results, conclusion/discussion, supplementary material if needed. **Maximum 1000 words** (including supplementary). The results section must include:
  - Your predictor's performance (accuracy, MCC, and similar metrics — the same evaluation toolkit Session 8 introduced), with training and validation loss curves.
  - At least one **ablation study** — how performance changes when you vary something about the model (regularization, dropout, which input features you use, number of layers, ...).
3. **An oral presentation of your code to a teaching assistant** (~15 minutes: 10 minutes walking through your model architecture and training procedure in your own notebook, 5 minutes of questions), focused on the deep-learning part of your work, not slides about the biology. You must be able to explain every part of the code you hand in.

This is not an exam — look up papers, discuss with peers, revisit earlier labs and notes. Just don't plagiarize; reports are checked with plagiarism software.

## Grading

### Report — 50%:

| Criterion                      | Weight | What it means                                                                                                            |
|--------------------------------|--------|--------------------------------------------------------------------------------------------------------------------------|
| Format and coherence           | 10%    | Clear intro/methods/results/discussion structure; stays within 1000 words                                                |
| Answered all required elements | 30%    | Every point above is addressed, with figures <i>and</i> accompanying written explanation                                 |
| Critical thinking / discussion | 30%    | Reasoning about your results, whatever they turned out to be — a honest, well-argued negative result can score well here |
| Figures                        | 15%    | Legible, referenced in the main text (no stand-alone figures), captions written as complete, self-contained sentences    |
| Overall impression             | 15%    | Understanding of the assignment and effort put in; answering beyond the minimum can help here                            |

### Code presentation — 50%:

| Criterion               | Weight | What it means                                                                                                            |
|-------------------------|--------|--------------------------------------------------------------------------------------------------------------------------|
| Reproducibility         | 30%    | We rerun your notebook and get your reported numbers back — fix your random seeds                                        |
| Documentation           | 20%    | Comments throughout, functions/methods explained                                                                         |
| Explanation of the code | 50%    | You walk through your own code live and answer follow-up questions about it — tests understanding, not just working code |

## Timeline

The project has three scheduled sessions; the dates and rooms are in the schedule on Canvas.

1. **Introduction**, in the week of Sessions 7-9, once you have seen secondary-structure prediction (Session 7) and how to evaluate a classifier (Session 8). We go through the task, the data and the grading, and you can start right away.
2. **Help with the project**, in the last week of teaching. Bring your notebook and your questions: the teachers and teaching assistants are there to help you get unstuck.
3. **Oral presentations of the code**, after the exam. Each of you presents to a teaching assistant, as described under Deliverables. The notebook and the report are due a few days later.

## Resources

- The dataset above, and the exam archive on the [course schedule page](#) — past exams are useful practice for the kind of reasoning this project also expects, even though this project itself isn't exam-style.
- Every day's own page and notebook, Session 5 through Session 12 in particular — this project deliberately doesn't introduce anything new.

## What's next

There isn't a Session 14 — this project is the course's closing synthesis, not a new topic to teach toward.

# References

Every page of this book names its sources where it uses them. This chapter collects them in one place: first the open textbooks and resources the book is built on, then all papers and books cited on any page (checked against PubMed or CrossRef), then databases, datasets and software, and finally the further reading and watching suggested for each session. “Cited in” links back to the pages that use each source.

## The open sources this book is built on

This book draws its narrative content from fully open sources, supplemented by original writing. None of the sources below restrict commercial use. Every page names which of them it used.

- **Dive into Deep Learning.** Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. <https://d2l.ai>. CC BY-SA 4.0. Primary source for every machine-learning session. Adapted material from this source is, per that license, shared here under the same terms.
- **Wikipedia.** <https://en.wikipedia.org>. CC BY-SA 4.0. Conceptual and algorithmic depth for the bioinformatics sessions, and occasional historical notes elsewhere. Never assigned as reading directly, but synthesized into this book’s own bounded pages (see the Welcome page for why).
- **EMBL-EBI Training.** <https://www.ebi.ac.uk/training/>. CC BY 4.0. The practical, tool-use layer for databases and sequence alignment.
- **biobook.** Käll, L. <https://github.com/statisticalbiotechnology/biobook>. CC BY 4.0. Notebook code implementing classical bioinformatics algorithms (pairwise alignment, MSA, and more), adapted with attribution.
- **NCBI Bookshelf.** <https://www.ncbi.nlm.nih.gov/books/>. Public domain (US government work). The BLAST algorithm and database practicalities, including the BLAST® Help manual.
- **RCSB PDB-101.** <https://pdb101.rcsb.org/>. CC BY 4.0 (the PDB-101 educational tier; general rcsb.org article text is separately copyrighted and is not used here).
- **A Byte of Python.** Swaroop, C. H. <https://python.swaroopch.com/>. CC BY-SA 4.0. Python fundamentals and object-oriented programming.

## Papers and books

- Abramson J, Adler J, Dunger J, et al. (2024). Accurate structure prediction of biomolecular interactions with AlphaFold 3. Nature 630:493-500.

- [doi:10.1038/s41586-024-07487-w](https://doi.org/10.1038/s41586-024-07487-w). PMID:38718835. *Cited in: Schedule, Session 12.*
- Alipanahi B, Delong A, Weirauch MT et al. (2015). Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nat Biotechnol* 33:831-8. [doi:10.1038/nbt.3300](https://doi.org/10.1038/nbt.3300). PMID:26213851. *Cited in: Session 10.*
  - Alley EC, Khimulya G, Biswas S et al. (2019). Unified rational protein engineering with sequence-based deep representation learning. *Nat Methods* 16:1315-1322. [doi:10.1038/s41592-019-0598-1](https://doi.org/10.1038/s41592-019-0598-1). PMID:31636460. *Cited in: Session 11.*
  - Almagro Armenteros JJ, Sønderby CK, Sønderby SK et al. (2017). DeepLoc: prediction of protein subcellular localization using deep learning. *Bioinformatics* 33:3387-3395. [doi:10.1093/bioinformatics/btx431](https://doi.org/10.1093/bioinformatics/btx431). PMID:29036616. *Cited in: Session 10.*
  - Almagro Armenteros JJ, Tsirigos KD, Sønderby CK et al. (2019). SignalP 5.0 improves signal peptide predictions using deep neural networks. *Nat Biotechnol* 37:420-423. [doi:10.1038/s41587-019-0036-z](https://doi.org/10.1038/s41587-019-0036-z). PMID:30778233. *Cited in: Session 11.*
  - Altschul SF, Gish W, Miller W, et al. (1990). Basic local alignment search tool. *J Mol Biol* 215(3):403-410. [doi:10.1016/S0022-2836\(05\)80360-2](https://doi.org/10.1016/S0022-2836(05)80360-2). PMID 2231712. *Cited in: Setup, Session 5.*
  - Altschul SF, Madden TL, Schäffer AA, et al. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res* 25(17):3389-3402. [doi:10.1093/nar/25.17.3389](https://doi.org/10.1093/nar/25.17.3389). PMID 9254694. *Cited in: Session 5.*
  - Anfinsen CB (1973). Principles that Govern the Folding of Protein Chains. *Science* 181:223-230. [doi:10.1126/science.181.4096.223](https://doi.org/10.1126/science.181.4096.223). PMID:4124164. *Cited in: Session 12.*
  - Baek M, DiMaio F, Anishchenko I et al. (2021). Accurate prediction of protein structures and interactions using a three-track neural network. *Science* 373:871-876. [doi:10.1126/science.abj8754](https://doi.org/10.1126/science.abj8754). PMID:34282049. *Cited in: Session 12.*
  - Bahdanau D, Cho K, Bengio Y (2014). Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473 (ICLR 2015). [doi:10.48550/arXiv.1409.0473](https://doi.org/10.48550/arXiv.1409.0473). *Cited in: Session 11.*
  - Baldwin J, Chothia C (1979). Haemoglobin: the structural changes related to ligand binding and its allosteric mechanism. *J Mol Biol* 129:175-220. [doi:10.1016/0022-2836\(79\)90277-8](https://doi.org/10.1016/0022-2836(79)90277-8). PMID:39173. *Cited in: Session 6.*
  - Basu S, Wallner B (2016). DockQ: A Quality Measure for Protein-Protein Docking Models. *PLoS One* 11:e0161879. [doi:10.1371/journal.pone.0161879](https://doi.org/10.1371/journal.pone.0161879). PMID:27560519. *Cited in: Session 12.*
  - Becares N, Gage MC, Voisin M, et al. (2019). Impaired LXR $\alpha$  phosphorylation attenuates progression of fatty liver disease. *Cell Reports* 26(4):984-995.e6. [doi:10.1016/j.celrep.2018.12.094](https://doi.org/10.1016/j.celrep.2018.12.094). PMID:30673619. *Cited in: Schedule, Session 2.*
  - Benjamini Y, Hochberg Y (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B* 57(1):289-300. [doi:10.1111/j.2517-6161.1995.tb02031.x](https://doi.org/10.1111/j.2517-6161.1995.tb02031.x). *Cited in: Session 1.*
  - Bommasani R, Hudson DA, Adeli E, et al. (2021). On the Opportunities and Risks of Foundation Models. arXiv:2108.07258. <https://arxiv.org/abs/2108.07258> *Cited*

in: [Session 11](#).

- Bryant P, Pozzati G, Elofsson A (2022). Improved prediction of protein-protein interactions using AlphaFold2. *Nat Commun* 13:1265. [doi:10.1038/s41467-022-28865-w](https://doi.org/10.1038/s41467-022-28865-w). PMID:35273146. *Cited in:* [Session 12](#).
- Bulyk ML, Johnson PLF, Church GM (2002). Nucleotides of transcription factor binding sites exert interdependent effects on the binding affinities of transcription factors. *Nucleic Acids Research* 30(5):1255-1261. [doi:10.1093/nar/30.5.1255](https://doi.org/10.1093/nar/30.5.1255). PMID:11861919. *Cited in:* [Schedule](#), [Session 7](#).
- Candido S, Hayes T, Derry A et al. (2026). Language Modeling Materializes a World Model of Protein Biology. *bioRxiv*, 2026.06.03.729735. [doi:10.64898/2026.06.03.729735](https://doi.org/10.64898/2026.06.03.729735). *Cited in:* [Session 12](#).
- Chen R, Li L, Weng Z (2003). ZDOCK: An initial-stage protein-docking algorithm. *Proteins* 52:80-87. [doi:10.1002/prot.10389](https://doi.org/10.1002/prot.10389). PMID:12784371. *Cited in:* [Session 12](#).
- Cheng H, Schaeffer RD, Liao Y, et al. (2014). ECOD: an evolutionary classification of protein domains. *PLoS Comput Biol* 10:e1003926. [doi:10.1371/journal.pcbi.1003926](https://doi.org/10.1371/journal.pcbi.1003926). PMID:25474468. *Cited in:* [Session 6](#).
- Chou PY, Fasman GD (1974). Prediction of protein conformation. *Biochemistry* 13(2):222-245. [doi:10.1021/bi00699a002](https://doi.org/10.1021/bi00699a002). PMID 4358940. *Cited in:* [Session 7](#).
- Dauparas J, Anishchenko I, Bennett N, et al. (2022). Robust deep learning-based protein sequence design using ProteinMPNN. *Science* 378(6615):49-56. [doi:10.1126/science.add2187](https://doi.org/10.1126/science.add2187). PMID:36108050. *Cited in:* [Schedule](#), [Session 12](#).
- Dayhoff MO, Schwartz RM, Orcutt BC (1978). A model of evolutionary change in proteins. In: Dayhoff MO (ed.), *Atlas of Protein Sequence and Structure*, Vol. 5, Suppl. 3, pp. 345-352. National Biomedical Research Foundation, Washington DC. *Cited in:* [Session 4](#).
- Devlin J, Chang M-W, Lee K et al. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805* (NAACL 2019). [doi:10.48550/arXiv.1810.04805](https://doi.org/10.48550/arXiv.1810.04805). *Cited in:* [Session 11](#).
- Dill KA, MacCallum JL (2012). The Protein-Folding Problem, 50 Years On. *Science* 338:1042-1046. [doi:10.1126/science.1219021](https://doi.org/10.1126/science.1219021). PMID:23180855. *Cited in:* [Session 12](#).
- Dominguez C, Boelens R, Bonvin AMJJ (2003). HADDOCK: A Protein-Protein Docking Approach Based on Biochemical or Biophysical Information. *J Am Chem Soc* 125:1731-1737. [doi:10.1021/ja026939x](https://doi.org/10.1021/ja026939x). PMID:12580598. *Cited in:* [Session 12](#).
- Dunbrack RL (2025). Rēs ipSAE loquuntur: What’s wrong with AlphaFold’s ipTM score and how to fix it. *bioRxiv*, 2025.02.10.637595. [doi:10.1101/2025.02.10.637595](https://doi.org/10.1101/2025.02.10.637595). PMID:39990437. *Cited in:* [Session 12](#).
- Ekeberg M, Lökvist C, Lan Y et al. (2013). Improved contact prediction in proteins: Using pseudolikelihoods to infer Potts models. *Phys Rev E* 87:012707. [doi:10.1103/physreve.87.012707](https://doi.org/10.1103/physreve.87.012707). PMID:23410359. *Cited in:* [Session 12](#).
- Ekman M (2021). *Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow*. Addison-Wesley. ISBN 978-0-13-747035-8. <https://www.informit.com/store/learning-deep-learning-theory-and-practice-of-neural-9780137470358> *Cited in:* [Welcome](#), [Session 7](#), [Session 8](#), [Session 9](#).
- Elnaggar A, Heinzinger M, Dallago C et al. (2022). ProtTrans: Toward Un-

- derstanding the Language of Life Through Self-Supervised Learning. *IEEE Trans Pattern Anal Mach Intell* 44:7112-7127. doi:10.1109/TPAMI.2021.3095381. PMID:34232869. *Cited in: Session 11.*
- Elofsson A (2026). AlphaFold3 at CASP16. *Proteins* 94:154-166. doi:10.1002/prot.70044. PMID:40851426. *Cited in: Session 12.*
  - Elofsson A, Ben-Tal N (2026). What will be the future of computational biology for macromolecules in the era of AI?. *PLoS Comput Biol* 22:e1014467. doi:10.1371/journal.pcbi.1014467. PMID:42418429. *Cited in: Session 12.*
  - Elofsson A, Han L, Bianchi E et al. (2024). Deep learning insights into the architecture of the mammalian egg-sperm fusion synapse. *eLife* 13:RP93131. doi:10.7554/elife.93131. PMID:38666763. *Cited in: Session 12.*
  - Esteva A, Kuprel B, Novoa RA et al. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature* 542:115-118. doi:10.1038/nature21056. PMID:28117445. *Cited in: Session 10.*
  - Evans R, O'Neill M, Pritzel A et al. (2021). Protein complex prediction with AlphaFold-Multimer. *bioRxiv*, 2021.10.04.463034. doi:10.1101/2021.10.04.463034. *Cited in: Session 12.*
  - Fermi G, Perutz MF, Shaanan B, et al. (1984). The crystal structure of human deoxyhaemoglobin at 1.74 Å resolution. *Journal of Molecular Biology* 175(2):159-174. doi:10.1016/0022-2836(84)90472-8. PMID:6726807. *Cited in: Session 3, Session 10.*
  - Fromm S, Ludaic M, Elofsson A (2026). Evaluating deep learning based structure prediction methods on antibody-antigen complexes. *Bioinformatics* 42:btg136. doi:10.1093/bioinformatics/btg136. PMID:41863324. *Cited in: Session 12.*
  - Garnier J, Osguthorpe DJ, Robson B (1978). Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins. *J Mol Biol* 120(1):97-120. doi:10.1016/0022-2836(78)90297-8. PMID 642007. *Cited in: Session 7.*
  - Goedhart J, Luijsterburg MS (2020). VolcanoNoseR is a web app for creating, exploring, labeling and sharing volcano plots. *Scientific Reports* 10:20560. doi:10.1038/s41598-020-76603-3. PMID:33239692. *Cited in: Session 2.*
  - Göbel U, Sander C, Schneider R et al. (1994). Correlated mutations and residue contacts in proteins. *Proteins* 18:309-317. doi:10.1002/prot.340180402. PMID:8208723. *Cited in: Session 12.*
  - He K, Zhang X, Ren S et al. (2016). Deep Residual Learning for Image Recognition. *Proc IEEE Conf Computer Vision and Pattern Recognition (CVPR) 2016*, 770-778. doi:10.1109/cvpr.2016.90. *Cited in: Session 10.*
  - Heinzinger M, Elnaggar A, Wang Y et al. (2019). Modeling aspects of the language of life through transfer-learning protein sequences. *BMC Bioinformatics* 20:723. doi:10.1186/s12859-019-3220-8. PMID:31847804. *Cited in: Session 11.*
  - Henikoff S, Henikoff JG (1992). Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci USA* 89(22):10915-10919. doi:10.1073/pnas.89.22.10915. PMID 1438297. *Cited in: Session 4, Session 5.*
  - Ho J, Jain A, Abbeel P (2020). Denoising Diffusion Probabilistic Models. *arXiv:2006.11239 (NeurIPS 33, 2020)*. doi:10.48550/arXiv.2006.11239. *Cited in: Session 12.*
  - Hochreiter S, Schmidhuber J (1997). Long Short-Term Memory. *Neural Computation* 9:1735-1780. doi:10.1162/neco.1997.9.8.1735. PMID:9377276. *Cited in:*

## Session 11.

- Holm L, Sander C (1993). Protein structure comparison by alignment of distance matrices. *J Mol Biol* 233:123-138. doi:10.1006/jmbi.1993.1489. PMID:8377180. Cited in: Session 6.
- Hopf TA, Ingraham JB, Poelwijk FJ, et al. (2017). Mutation effects predicted from sequence co-variation. *Nat Biotechnol* 35(2):128-135. doi:10.1038/nbt.3769. PMID 28092658. Cited in: Session 12.
- Hopf TA, Schärfe CPI, Rodrigues JPGLM et al. (2014). Sequence co-evolution gives 3D contacts and structures of protein complexes. *eLife* 3:e03430. doi:10.7554/elife.03430. PMID:25255213. Cited in: Session 12.
- Illergård K, Ardell DH, Elofsson A (2009). Structure is three to ten times more conserved than sequence—a study of structural response in protein cores. *Proteins* 77(3):499-508. doi:10.1002/prot.22458. PMID 19507241. Cited in: Session 7.
- Ioffe S, Szegedy C (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, PMLR 37:448-456. arXiv:1502.03167. <https://arxiv.org/abs/1502.03167> Cited in: Session 9.
- Isomorphic Labs Team (2026). Accurate predictions of novel biomolecular interactions with IsoDDE. Technical report, Zenodo. doi:10.5281/zenodo.19699685. Cited in: Session 12.
- Janin J, Henrick K, Moult J et al. (2003). CAPRI: A Critical Assessment of PRedicted Interactions. *Proteins* 52:2-9. doi:10.1002/prot.10381. PMID:12784359. Cited in: Session 12.
- Jumper J, Evans R, Pritzel A, et al. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature* 596:583-589. doi:10.1038/s41586-021-03819-2. PMID:34265844. Cited in: Schedule, Session 12.
- Kabsch W, Sander C (1984). On the use of sequence homologies to predict protein structure: identical pentapeptides can have completely different conformations. *Proc Natl Acad Sci USA* 81(4):1075-1078. doi:10.1073/pnas.81.4.1075. PMID 6422466. Cited in: Session 7.
- Karlin S, Altschul SF (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc Natl Acad Sci USA* 87(6):2264-2268. doi:10.1073/pnas.87.6.2264. PMID 2315319. Cited in: Session 5.
- Katchalski-Katzir E, Shariv I, Eisenstein M et al. (1992). Molecular surface recognition: determination of geometric fit between proteins and their ligands by correlation techniques. *Proc Natl Acad Sci USA* 89:2195-2199. doi:10.1073/pnas.89.6.2195. PMID:1549581. Cited in: Session 12.
- Kather JN, Krisam J, Charoentong P, et al. (2019). Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study. *PLoS Med* 16:e1002730. doi:10.1371/journal.pmed.1002730. Cited in: Session 10.
- Kelley DR, Snoek J, Rinn JL (2016). Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Res* 26:990-9. doi:10.1101/gr.200535.115. PMID:27197224. Cited in: Session 10.
- Kingma DP, Ba J (2015). Adam: A Method for Stochastic Optimization. 3rd International Conference on Learning Representations (ICLR). arXiv:1412.6980. <https://arxiv.org/abs/1412.6980> Cited in: Session 9.

- Korf I, Yandell M, Bedell J (2003). BLAST. O'Reilly Media, Sebastopol CA. ISBN 978-0-596-00299-2. <https://www.oreilly.com/library/view/blast/0596002998/> Cited in: Session 5.
- Krishna R, Wang J, Ahern W et al. (2024). Generalized biomolecular modeling and design with RoseTTAFold All-Atom. Science 384:eadl2528. doi:10.1126/science.adl2528. PMID:38452047. Cited in: Session 12.
- Krizhevsky A, Sutskever I, Hinton GE (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems 25:1097-1105. (Journal version: Commun ACM 60:84-90, 2017, doi:10.1145/3065386.) Cited in: Session 10.
- Kundrotas PJ, Zhu Z, Janin J et al. (2012). Templates are available to model nearly all complexes of structurally characterized proteins. Proc Natl Acad Sci USA 109:9438-9441. doi:10.1073/pnas.1200678109. PMID:22645367. Cited in: Session 12.
- Kyte J, Doolittle RF (1982). A simple method for displaying the hydropathic character of a protein. J Mol Biol 157:105-132. doi:10.1016/0022-2836(82)90515-0. PMID:7108955. Cited in: Session 10.
- Lau AM, Bordin N, Kandathil SM, et al. (2024). Exploring structural diversity across the protein universe with The Encyclopedia of Domains. Science 386:eadq4946. doi:10.1126/science.adq4946. PMID:39480926. Cited in: Session 6.
- LeCun Y, Boser B, Denker JS et al. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation 1:541-551. doi:10.1162/neco.1989.1.4.541. Cited in: Session 10.
- Lensink MF, Brysbaert G, Nadzirin N et al. (2019). Blind prediction of homo- and hetero-protein complexes: The CASP13-CAPRI experiment. Proteins 87:1200-1221. doi:10.1002/prot.25838. PMID:31612567. Cited in: Session 12.
- Levinthal C (1969). How to fold gracefully. In: Debrunner P, Tsibris JCM, Münck E (eds), Mössbauer Spectroscopy in Biological Systems: Proceedings of a meeting held at Allerton House, Monticello, Illinois. University of Illinois Press, pp. 22-24. Cited in: Session 12.
- Li W, Kinch LN, Karplus PA, Grishin NV (2015). ChSeq: a database of chameleon sequences. Protein Sci 24(7):1075-1086. doi:10.1002/pro.2689. PMID 25970262. Cited in: Session 7.
- Lin Z, Akin H, Rao R et al. (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. Science 379:1123-1130. doi:10.1126/science.ade2574. PMID:36927031. Cited in: Session 11, Session 12.
- Lipman Y, Chen RTQ, Ben-Hamu H et al. (2022). Flow Matching for Generative Modeling. arXiv:2210.02747 (ICLR 2023). doi:10.48550/arXiv.2210.02747. Cited in: Session 12.
- Ludaic M, Elofsson A (2026). Limits of deep-learning-based RNA prediction methods. Nucleic Acids Res 54:gkag813. doi:10.1093/nar/gkag813. PMID:42635124. Cited in: Session 12.
- Marks DS, Colwell LJ, Sheridan R et al. (2011). Protein 3D Structure Computed from Evolutionary Sequence Variation. PLoS One 6:e28766. doi:10.1371/journal.pone.0028766. PMID:22163331. Cited in: Session 12.
- Michel M, Hayat S, Skwark MJ et al. (2014). PconsFold: improved contact predictions improve protein models. Bioinformatics 30:i482-i488. doi:10.1093/bioinfor-

- [matics/btu458](#). PMID:25161237. *Cited in: Session 12.*
- Michel M, Menéndez Hurtado D, Elofsson A (2019). PconsC4: fast, accurate and hassle-free contact predictions. *Bioinformatics* 35:2677-2679. [doi:10.1093/bioinformatics/bty1036](#). PMID:30590407. *Cited in: Session 12.*
  - Michel M, Skwark MJ, Menéndez Hurtado D et al. (2017). Predicting accurate contacts in thousands of Pfam domain families using PconsC3. *Bioinformatics* 33:2859-2866. [doi:10.1093/bioinformatics/btx332](#). PMID:28535189. *Cited in: Session 12.*
  - Minsky M, Papert S (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge MA. (Expanded reissue 2017, [doi:10.7551/mitpress/11301.001.0001.](#)) *Cited in: Session 7.*
  - Mirabello C, Wallner B (2019). rawMSA: End-to-end Deep Learning using raw Multiple Sequence Alignments. *PLoS One* 14:e0220182. [doi:10.1371/journal.pone.0220182](#). PMID:31415569. *Cited in: Session 12.*
  - Mirdita M, Schütze K, Moriwaki Y, et al. (2022). ColabFold: making protein folding accessible to all. *Nature Methods* 19:679-682. [doi:10.1038/s41592-022-01488-1](#). PMID:35637307. *Cited in: Schedule, Session 12.*
  - Monastyrskyy B, D'Andrea D, Fidelis K et al. (2014). Evaluation of residue-residue contact prediction in CASP10. *Proteins* 82 Suppl 2:138-153. [doi:10.1002/prot.24340](#). PMID:23760879. *Cited in: Session 12.*
  - Monastyrskyy B, D'Andrea D, Fidelis K et al. (2016). New encouraging developments in contact prediction: Assessment of the CASP11 results. *Proteins* 84 Suppl 1:131-144. [doi:10.1002/prot.24943](#). PMID:26474083. *Cited in: Session 12.*
  - Morcos F, Pagnani A, Lunt B et al. (2011). Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proc Natl Acad Sci USA* 108:E1293-E1301. [doi:10.1073/pnas.1111471108](#). PMID:22106262. *Cited in: Session 12.*
  - Moult J, Pedersen JT, Judson R et al. (1995). A large-scale experiment to assess protein structure prediction methods. *Proteins* 23:ii-v. [doi:10.1002/prot.340230303](#). PMID:8710822. *Cited in: Session 12.*
  - Murzin AG, Brenner SE, Hubbard T, et al. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J Mol Biol* 247:536-540. [doi:10.1006/jmbi.1995.0159](#). PMID:7723011. *Cited in: Session 6.*
  - Needleman SB, Wunsch CD (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol* 48(3):443-453. [doi:10.1016/0022-2836\(70\)90057-4](#). PMID 5420325. *Cited in: Session 4.*
  - Nielsen H, Engelbrecht J, Brunak S et al. (1997). Identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites. *Protein Eng* 10:1-6. [doi:10.1093/protein/10.1.1](#). PMID:9051728. *Cited in: Session 11.*
  - Nielsen H, Teufel F, Brunak S et al. (2024). SignalP: The Evolution of a Web Server. *Methods Mol Biol* 2836:331-367. [doi:10.1007/978-1-0716-4007-4\\_17](#). PMID:38995548. *Cited in: Session 11.*
  - Oquab M, Darcet T, Moutakanni T, et al. (2023). DINOv2: Learning Robust Visual Features without Supervision. *arXiv:2304.07193* (Transactions on Machine Learning Research, 2024). <https://arxiv.org/abs/2304.07193> *Cited in: Session 10.*
  - Orengo CA, Michie AD, Jones S, et al. (1997). CATH—a hierarchic classification of protein domain structures. *Structure* 5:1093-1108. [doi:10.1016/S0969-2126\(97\)00260-8](#). PMID:9309224. *Cited in: Session 6.*

- Ovchinnikov S, Kamisetty H, Baker D (2014). Robust and accurate prediction of residue-residue interactions across protein interfaces using evolutionary information. *eLife* 3:e02030. doi:10.7554/elife.02030. PMID:24842992. Cited in: [Session 12](#).
- Pevsner J (2015). *Bioinformatics and Functional Genomics*, 3rd edition. Wiley-Blackwell. ISBN 978-1-118-58178-0. <https://www.wiley.com/en-us/Bioinformatics+and+Functional+Genomics%2C+3rd+Edition-p-9781118581780> Cited in: [Welcome](#).
- Pozzati G, Zhu W, Bassot C et al. (2022). Limits and potential of combined folding and docking. *Bioinformatics* 38:954-961. doi:10.1093/bioinformatics/btab760. PMID:34788800. Cited in: [Session 12](#).
- Qian N, Sejnowski TJ (1988). Predicting the secondary structure of globular proteins using neural network models. *J Mol Biol* 202(4):865-884. doi:10.1016/0022-2836(88)90564-5. PMID 3172241. Cited in: [Session 7](#).
- Radford A, Narasimhan K, Salimans T, Sutskever I (2018). Improving language understanding by generative pre-training. OpenAI technical report. [https://cdn.openai.com/research-covers/language-unsupervised/language\\_understanding\\_paper.pdf](https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf) Cited in: [Session 11](#).
- Rao R, Meier J, Sercu T, Ovchinnikov S, Rives A (2021). Transformer protein language models are unsupervised structure learners. *International Conference on Learning Representations (ICLR 2021)*. Preprint doi:10.1101/2020.12.15.422761. Cited in: [Session 11](#).
- Rao RM, Liu J, Verkuil R et al. (2021). MSA Transformer. *Proceedings of the 38th International Conference on Machine Learning (ICML)*, PMLR 139:8844-8856. Preprint doi:10.1101/2021.02.12.430858. Cited in: [Session 12](#).
- Ronneberger O, Fischer P, Brox T (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. *MICCAI 2015, Lecture Notes in Computer Science* 9351:234-241. doi:10.1007/978-3-319-24574-4\_28. Cited in: [Session 10](#).
- Rost B, Sander C (1993). Improved prediction of protein secondary structure by use of sequence profiles and neural networks. *Proc Natl Acad Sci USA* 90:7558-7562. doi:10.1073/pnas.90.16.7558. PMID:8356056. PMCID:PMC47181. Cited in: [Session 9](#).
- Rost B, Sander C (1993). Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology* 232(2):584-599. doi:10.1006/jmbi.1993.1413. PMID:8345525. Cited in: [Schedule](#), [Session 7](#), [Session 9](#), [Capstone](#).
- Ruiz-Serra V, Pontes C, Milanetti E et al. (2021). Assessing the accuracy of contact and distance predictions in CASP14. *Proteins* 89:1888-1900. doi:10.1002/prot.26248. PMID:34595772. Cited in: [Session 12](#).
- Rumelhart DE, Hinton GE, Williams RJ (1986). Learning representations by back-propagating errors. *Nature* 323(6088):533-536. doi:10.1038/323533a0. Cited in: [Session 7](#).
- Russakovsky O, Deng J, Su H et al. (2015). ImageNet Large Scale Visual Recognition Challenge. *Int J Comput Vis* 115:211-252. doi:10.1007/s11263-015-0816-y. Cited in: [Session 10](#).
- Saitou N, Nei M (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol Biol Evol* 4:406-425. doi:10.1093/oxfordjournals.molbev.a040454. Cited in: [Session 4](#).

- Schaarschmidt J, Monastyrskyy B, Kryshchak A et al. (2018). Assessment of contact predictions in CASP12: Co-evolution and deep learning coming of age. *Proteins* 86 Suppl 1:51-66. doi:10.1002/prot.25407. PMID:29071738. Cited in: Session 12.
- Schneider TD, Stephens RM (1990). Sequence logos: a new way to display consensus sequences. *Nucleic Acids Res* 18(20):6097-6100. doi:10.1093/nar/18.20.6097. PMID 2172928. Cited in: Session 4.
- Schuster M, Paliwal K (1997). Bidirectional recurrent neural networks. *IEEE Trans Signal Process* 45:2673-2681. doi:10.1109/78.650093. PMID:42670431. Cited in: Session 11.
- Senior AW, Evans R, Jumper J, et al. (2020). Improved protein structure prediction using potentials from deep learning. *Nature* 577:706-710. doi:10.1038/s41586-019-1923-7. PMID:31942072. Cited in: Schedule, Session 12.
- Shrestha R, Fajardo E, Gil N et al. (2019). Assessing the accuracy of contact predictions in CASP13. *Proteins* 87:1058-1068. doi:10.1002/prot.25819. PMID:31587357. Cited in: Session 12.
- Simons KT, Kooperberg C, Huang E et al. (1997). Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *J Mol Biol* 268:209-225. doi:10.1006/jmbi.1997.0959. PMID:9149153. Cited in: Session 12.
- Simonyan K, Zisserman A (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 (ICLR 2015). doi:10.48550/arXiv.1409.1556. Cited in: Session 10.
- Skwark MJ, Raimondi D, Michel M et al. (2014). Improved Contact Predictions Using the Recognition of Protein Like Contact Patterns. *PLoS Comput Biol* 10:e1003889. doi:10.1371/journal.pcbi.1003889. PMID:25375897. Cited in: Session 12.
- Smith TF, Waterman MS (1981). Identification of common molecular subsequences. *J Mol Biol* 147(1):195-197. doi:10.1016/0022-2836(81)90087-5. PMID 7265238. Cited in: Session 4.
- Sohl-Dickstein J, Weiss EA, Maheswaranathan N et al. (2015). Deep Unsupervised Learning using Nonequilibrium Thermodynamics. arXiv:1503.03585 (ICML 2015, PMLR 37:2256-2265). doi:10.48550/arXiv.1503.03585. Cited in: Session 12.
- Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J Mach Learn Res* 15:1929-1958. https://jmlr.org/papers/v15/srivastava14a.html Cited in: Session 9.
- Steinegger M, Söding J (2017). MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology* 35(11):1026-1028. doi:10.1038/nbt.3988. PMID:29035372. Cited in: Schedule, Setup, Session 5, Session 8, Session 11, Session 12.
- Swaroop CH. A Byte of Python. Online book, CC BY-SA 4.0. https://python.swaroopch.com/ Cited in: Session 2.
- Teufel F, Almagro Armenteros JJ, Johansen AR et al. (2022). SignalP 6.0 predicts all five types of signal peptides using protein language models. *Nat Biotechnol* 40:1023-1025. doi:10.1038/s41587-021-01156-3. PMID:34980915. Cited in: Session 11.
- Teufel F, Gíslason MH, Almagro Armenteros JJ et al. (2023). GraphPart: homology

- partitioning for biological sequence analysis. *NAR Genom Bioinform* 5:lqad088. doi:10.1093/nargab/lqad088. PMID:37850036. *Cited in: Session 11.*
- Tunyasuvunakool K, Adler J, Wu Z et al. (2021). Highly accurate protein structure prediction for the human proteome. *Nature* 596:590-596. doi:10.1038/s41586-021-03828-1. PMID:34293799. *Cited in: Session 12.*
  - Vakser IA (1995). Protein docking for low-resolution structures. *Protein Eng* 8:371-378. doi:10.1093/protein/8.4.371. PMID:7567922. *Cited in: Session 12.*
  - van Kempen M, Kim SS, Tumescheit C, et al. (2024). Fast and accurate protein structure search with Foldseek. *Nat Biotechnol* 42:243-246. doi:10.1038/s41587-023-01773-0. PMID:37156916. *Cited in: Session 6.*
  - Vaswani A, Shazeer N, Parmar N et al. (2017). Attention Is All You Need. arXiv:1706.03762 (NIPS 30, 2017). doi:10.48550/arXiv.1706.03762. *Cited in: Session 11.*
  - Veeling BS, Linmans J, Winkens J, Cohen T, Welling M (2018). Rotation Equivariant CNNs for Digital Pathology. MICCAI 2018, Lecture Notes in Computer Science 11071:210-218. doi:10.1007/978-3-030-00934-2\_24. *Cited in: Session 10.*
  - Wang S, Sun S, Li Z et al. (2017). Accurate De Novo Prediction of Protein Contact Map by Ultra-Deep Learning Model. *PLoS Comput Biol* 13:e1005324. doi:10.1371/journal.pcbi.1005324. PMID:28056090. *Cited in: Session 12.*
  - Watson JL, Juergens D, Bennett NR, et al. (2023). De novo design of protein structure and function with RFdiffusion. *Nature* 620:1089-1100. doi:10.1038/s41586-023-06415-8. PMID:37433327. *Cited in: Schedule, Session 12.*
  - Weigt M, White RA, Szurmant H et al. (2009). Identification of direct residue contacts in protein-protein interaction by message passing. *Proc Natl Acad Sci USA* 106:67-72. doi:10.1073/pnas.0805923106. PMID:19116270. *Cited in: Session 12.*
  - Wu NC, Dai L, Olson CA, Lloyd-Smith JO, Sun R (2016). Adaptation in protein fitness landscapes is facilitated by indirect paths. *eLife* 5:e16965. doi:10.7554/eLife.16965. PMID 27391790. *Cited in: Session 9.*
  - Yang J, Anishchenko I, Park H et al. (2020). Improved protein structure prediction using predicted interresidue orientations. *Proc Natl Acad Sci USA* 117:1496-1503. doi:10.1073/pnas.1914677117. PMID:31896580. *Cited in: Session 12.*
  - Zhang A, Lipton ZC, Li M, Smola AJ (2023). Dive into Deep Learning. Cambridge University Press (open version at <https://d2l.ai>; arXiv:2106.11342). *Cited in: Welcome, Session 7.*
  - Zhang Y, Skolnick J (2005). TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic Acids Res* 33:2302-2309. doi:10.1093/nar/gki524. PMID:15849316. *Cited in: Session 6.*
  - Zhou J, Troyanskaya OG (2015). Predicting effects of noncoding variants with deep learning-based sequence model. *Nat Methods* 12:931-4. doi:10.1038/nmeth.3547. PMID:26301843. *Cited in: Session 10.*

## Databases, datasets and software

- Abanades B, Wong WK, Boyles F et al. (2023). ImmuneBuilder: Deep-Learning models for predicting the structures of immune proteins. *Commun Biol* 6:575.

- [doi:10.1038/s42003-023-04927-7](https://doi.org/10.1038/s42003-023-04927-7). PMID:37248282. *Cited in: Session 12.*
- Acevedo A, Merino A, Alf  rez S, Molina   , Bold   L, Rodellar J (2020). A dataset of microscopic peripheral blood cell images for development of automatic recognition systems. *Data Brief* 30:105474. [doi:10.1016/j.dib.2020.105474](https://doi.org/10.1016/j.dib.2020.105474). *Cited in: Session 8.*
  - AlphaFold Server (Google DeepMind / Isomorphic Labs). <https://alphafoldserver.com> *Cited in: Session 12.*
  - Becares et al. (2019) differential-expression table (S196A vs WT LXR   mouse liver, HFHC diet; Fig. 4C), as shared in the VolcanoR repository: Becares-diffgenes\_HFHC.csv. [https://raw.githubusercontent.com/JoachimGoedhart/VolcanoR/master/Becares-diffgenes\\_HFHC.csv](https://raw.githubusercontent.com/JoachimGoedhart/VolcanoR/master/Becares-diffgenes_HFHC.csv) *Cited in: Session 2.*
  - Biopython: freely available Python tools for computational molecular biology. <https://biopython.org/> *Cited in: Session 6.*
  - ByteDance AML AI4Science Team, Chen X, Zhang Y et al. (2025). Protenix - Advancing Structure Prediction Through a Comprehensive AlphaFold3 Reproduction. *bioRxiv*, 2025.01.08.631967. [doi:10.1101/2025.01.08.631967](https://doi.org/10.1101/2025.01.08.631967). *Cited in: Session 12.*
  - CATH protein structure classification database. <https://www.cathdb.info/> *Cited in: Session 6.*
  - Chai Discovery, Boitreaud J, Dent J et al. (2024). Chai-1: Decoding the molecular interactions of life. *bioRxiv*, 2024.10.10.615955. [doi:10.1101/2024.10.10.615955](https://doi.org/10.1101/2024.10.10.615955). *Cited in: Session 12.*
  - Cock PJA, Antao T, Chang JT, et al. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* 25(11):1422-1423. [doi:10.1093/bioinformatics/btp163](https://doi.org/10.1093/bioinformatics/btp163). PMID:19304878. *Cited in: Session 2, Session 4.*
  - Complex Portal CPX-2158, Hemoglobin HbA complex. <https://www.ebi.ac.uk/complexportal/complex/CPX-2158> *Cited in: Session 3.*
  - contact-ajmal. OpenDDE (open drug-design engine), GitHub repository, MIT licence (2026). <https://github.com/contact-ajmal/opensdd> *Cited in: Session 12.*
  - Dallago C, Mou J, Johnston KE, et al. (2021). FLIP: Benchmark tasks in fitness landscape inference for proteins. *bioRxiv*. [doi:10.1101/2021.11.09.467890](https://doi.org/10.1101/2021.11.09.467890) (GB1 four-site data set). *Cited in: Session 9.*
  - DINOv2 ViT-S/14 weights (facebook/dinov2-small), Apache 2.0, via Hugging Face transformers. <https://huggingface.co/facebook/dinov2-small> *Cited in: Session 10.*
  - DTU Health Tech. SignalP 6.0 web server. <https://services.healthtech.dtu.dk/services/SignalP-6.0/> *Cited in: Session 11.*
  - ECOD: Evolutionary Classification of protein Domains database. <http://prodata.swmed.edu/ecod/> *Cited in: Session 6.*
  - Eddy SR (2011). Accelerated profile HMM searches. *PLoS Computational Biology* 7(10):e1002195. [doi:10.1371/journal.pcbi.1002195](https://doi.org/10.1371/journal.pcbi.1002195). PMID:22039361. *Cited in: Setup.*
  - Edgar RC (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res* 32(5):1792-1797. [doi:10.1093/nar/gkh340](https://doi.org/10.1093/nar/gkh340). PMID 15034147. *Cited in: Session 4.*
  - EMBL-EBI. AlphaFold DB. <https://alphafold.ebi.ac.uk/> *Cited in: Session 3, Session 6.*
  - EMBL-EBI. BioModels. <https://www.ebi.ac.uk/biomodels/> *Cited in: Session 3.*

- EMBL-EBI. Complex Portal. <https://www.ebi.ac.uk/complexportal/> Cited in: Session 3.
- EMBL-EBI. EVA. <https://www.ebi.ac.uk/eva/> Cited in: Session 3.
- EMBL-EBI. GWAS Catalog. <https://www.ebi.ac.uk/gwas/> Cited in: Session 3.
- EMBL-EBI. IntAct. <https://www.ebi.ac.uk/intact/> Cited in: Session 3.
- EMBL-EBI. InterPro. <https://www.ebi.ac.uk/interpro/> Cited in: Session 3, Session 5.
- EMBL-EBI. PDBe. <https://www.ebi.ac.uk/pdbe/> Cited in: Session 3.
- EMBL-EBI. PRIDE. <https://www.ebi.ac.uk/pride/> Cited in: Session 3.
- ESM Metagenomic Atlas / ESM Atlas API (Meta AI; ESMFold predictions). <https://esmatlas.com/> Cited in: Session 12.
- Fermi G, Perutz MF, Shaanan B, Fourme R (1984). PDB entry 2HHB: The crystal structure of human deoxyhaemoglobin at 1.74 Å resolution. RCSB Protein Data Bank. <https://www.rcsb.org/structure/2HHB> Cited in: Session 6.
- Foldseek software (Steinegger lab), GitHub repository and web server. <https://github.com/steineggerlab/foldseek> Cited in: Session 6.
- Foldseek web server. <https://search.foldseek.com/search> Cited in: Session 6.
- FunCoup. [funcoup.org](https://funcoup.org/). <https://funcoup.org/> Cited in: Session 3.
- GenBank (NCBI nucleotide sequence database). <https://www.ncbi.nlm.nih.gov/genbank/> Cited in: Session 5.
- Google Colaboratory (Colab). Google. <https://colab.research.google.com/> Cited in: Session 0.
- Harris CR, Millman KJ, van der Walt SJ, et al. (2020). Array programming with NumPy. Nature 585:357-362. doi:10.1038/s41586-020-2649-2. PMID:32939066. Cited in: Session 2.
- Human reviewed UniProtKB/Swiss-Prot entries with a single annotated subcellular location (5 classes, 700 proteins), compiled for this course. UniProt Consortium, CC BY 4.0. <https://www.uniprot.org/> Cited in: Session 8, Session 9, Session 10, Session 11.
- Hunter JD (2007). Matplotlib: a 2D graphics environment. Computing in Science & Engineering 9(3):90-95. doi:10.1109/MCSE.2007.55. Cited in: Session 2.
- JASPAR matrix MA0162.4 (EGR1, Homo sapiens, ChIP-seq, source ReMap), CC BY 4.0. Rauluseviciute I, Riudavets-Puig R, Blanc-Mathieu R, et al. (2024). JASPAR 2024: 20th anniversary of the open-access database of transcription factor binding profiles. Nucleic Acids Res 52(D1):D174-D182. doi:10.1093/nar/gkad1059. PMID 37962376. Cited in: Session 7.
- Kabsch W, Sander C (1983). Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. Biopolymers 22:2577-2637. doi:10.1002/bip.360221211. PMID:6667333. Cited in: Capstone.
- Kather JN, Halama N, Marx A (2018). 100,000 histological images of human colorectal cancer and healthy tissue (NCT-CRC-HE-100K and CRC-VAL-HE-7K). Zenodo. doi:10.5281/zenodo.1214456. CC BY 4.0. Cited in: Session 10.
- KB8029 secondary-structure dataset: 1,200 protein domains (FASTA, DSSP labels, PSSM CSV), dataset.zip (~3 MB), Google Drive. <https://drive.google.com/drive/folders/1gnxfXXYysCk4L4FYztzeejPVXJaMnxQw?usp=sharing> Cited in: Capstone.
- Krivák R, Hoksza D (2018). P2Rank: machine learning based tool for rapid and accurate prediction of ligand binding sites from protein structure. J Cheminform 10:39. doi:10.1186/s13321-018-0285-8. PMID:30109435. Cited in: Session 12.

- Käll L. bibook: Bioinformatics book (GitHub repository, statisticalbiotechnology/bibook), CC BY 4.0. <https://github.com/statisticalbiotechnology/bibook> Cited in: Session 4.
- Love MI, Huber W, Anders S (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology* 15:550. doi:10.1186/s13059-014-0550-8. PMID:25516281. Cited in: Session 2.
- Manim: Mathematical Animation Engine (Grant Sanderson / 3Blue1Brown; community edition by the Manim Community). <https://www.manim.community/> Cited in: Session 2.
- McKinney W (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference (SciPy 2010)*, 56-61. doi:10.25080/Majora-92bf1922-00a. Cited in: Session 2.
- Meta AI / FAIR. fair-esm: Evolutionary Scale Modeling (ESM) pretrained protein language models (Python package). <https://github.com/facebookresearch/esm> Cited in: Session 11.
- MMseqs2 software, GitHub repository (Söding/Steinegger labs). <https://github.com/soedinglab/MMseqs2> Cited in: Session 8.
- NCBI RefSeq NM\_000518.5, Homo sapiens hemoglobin subunit beta (HBB), mRNA. [https://www.ncbi.nlm.nih.gov/nuccore/NM\\_000518.5](https://www.ncbi.nlm.nih.gov/nuccore/NM_000518.5) Cited in: Session 3.
- NCBI. ClinVar. <https://www.ncbi.nlm.nih.gov/clinvar/> Cited in: Session 3.
- NCI. GDC Data Portal. <https://portal.gdc.cancer.gov/> Cited in: Session 3.
- Notredame C, Higgins DG, Heringa J (2000). T-Coffee: a novel method for fast and accurate multiple sequence alignment. *J Mol Biol* 302(1):205-217. doi:10.1006/jmbi.2000.4042. PMID 10964570. Cited in: Session 4.
- OpenFold Consortium. OpenFold3 (preview release). <https://github.com/aqlaboratory/openfold-3> Cited in: Session 12.
- Ormo M, Cubitt AB, Kallio K, et al. (1996). PDB entry 1EMA: Green fluorescent protein from *Aequorea victoria*. RCSB Protein Data Bank. <https://www.rcsb.org/structure/1EMA> Cited in: Session 6.
- Passaro S, Corso G, Wohlwend J et al. (2025). Boltz-2: Towards Accurate and Efficient Binding Affinity Prediction. *bioRxiv*, 2025.06.14.659707. doi:10.1101/2025.06.14.659707. PMID:40667369. Cited in: Session 12.
- Paszke A, Gross S, Massa F, et al. (2019). PyTorch: an imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* 32 (NeurIPS 2019); arXiv:1912.01703. <https://arxiv.org/abs/1912.01703> Cited in: Session 2, Session 7.
- PDB entry 1OJ6: human neuroglobin. RCSB Protein Data Bank. <https://www.rcsb.org/structure/1OJ6> Cited in: Session 6.
- PDB entry 1SHW (Ephrin-A5 / EphB2 complex). <https://www.rcsb.org/structure/1SHW> Cited in: Session 12.
- PDB entry 1VL6: Crystal structure of NAD-dependent malic enzyme (TM0542) from *Thermotoga maritima* at 2.61 Å resolution (JCSG, released 2004; unpublished). <https://www.rcsb.org/structure/1VL6> Cited in: Session 7.
- PDB entry 2HHB: Fermi G, Perutz MF, Shaanan B, Fourme R (1984). The crystal structure of human deoxyhaemoglobin at 1.74 Å resolution. *J Mol Biol* 175(2):159-174. doi:10.1016/0022-2836(84)90472-8. PMID 6726807. Cited in: Session 7.

- PDB entry 3CHY: Volz K, Matsumura P (1991). Crystal structure of Escherichia coli CheY refined at 1.7-Å resolution. J Biol Chem 266:15511-15519. <https://www.rcsb.org/structure/3CHY> Cited in: Session 12.
- PDB entry 4HHB (human deoxyhaemoglobin): Fermi G, Perutz MF, Shaanan B, Fourme R (1984). J Mol Biol 175:159-174. doi:10.1016/0022-2836(84)90472-8. Cited in: Session 12.
- PDB entry 4JB9: Crystal structure of antibody VRC06 in complex with HIV-1 gp120 core (2013; primary citation: Delineating antibody recognition in polyclonal sera from patterns of HIV-1 isolate neutralization, Science 2013). <https://www.rcsb.org/structure/4JB9> Cited in: Session 7.
- PDB entry 7LXM: HIV-1 envelope glycoprotein trimer in complex with antibody. RCSB Protein Data Bank. <https://www.rcsb.org/structure/7LXM> Cited in: Session 6.
- PDB entry 8C6Z. RCSB Protein Data Bank. <https://www.rcsb.org/structure/8C6Z> Cited in: Session 6.
- PDBe REST API (Protein Data Bank in Europe, EMBL-EBI). <https://www.ebi.ac.uk/pdbe/api/doc/> Cited in: Session 7, Session 8.
- Pedregosa F, Varoquaux G, Gramfort A, et al. (2011). Scikit-learn: machine learning in Python. Journal of Machine Learning Research 12:2825-2830. <https://jmlr.org/papers/v12/pedregosa11a.html> Cited in: Session 2, Session 7.
- Pfam family PF00042 (Globin), accessed via InterPro. Pfam: Mistry J, Chuguransky S, Williams L, et al. (2021). Pfam: the protein families database in 2021. Nucleic Acids Res 49(D1):D412-D419. doi:10.1093/nar/gkaa913. PMID 33125078. Cited in: Session 4, Session 5.
- Protein Data Bank entry 2HHB (human deoxyhaemoglobin). <https://www.rcsb.org/structure/2HHB> Cited in: Session 3.
- PyMOL Molecular Graphics System. Schrödinger, LLC. <https://pymol.org/> Cited in: Setup, Session 0, Session 6, Session 7.
- PyTorch deep-learning framework. <https://pytorch.org/> Cited in: Session 8, Session 9.
- RCSB PDB. Data API (GraphQL). <https://data.rcsb.org/graphql/index.html> Cited in: Session 3.
- RCSB PDB. Search API. <https://search.rcsb.org/> Cited in: Session 3.
- RCSB Protein Data Bank. <https://www.rcsb.org/> Cited in: Session 12.
- RCSB Protein Data Bank. Berman HM, Westbrook J, Feng Z, et al. (2000). The Protein Data Bank. Nucleic Acids Res 28(1):235-242. doi:10.1093/nar/28.1.235. PMID 10592235. Cited in: Session 5.
- RDKit: Open-source cheminformatics. <https://www.rdkit.org/> Cited in: Session 12.
- Reactome. Reactome. <https://reactome.org/> Cited in: Session 3.
- Rfam. RF00005 (tRNA). <https://rfam.org/family/RF00005> Cited in: Session 3.
- Rosetta Commons. Rosetta software suite. <https://rosettacommons.org/> Cited in: Session 6.
- SCOP database, hosted at EMBL-EBI PDBe. <https://www.ebi.ac.uk/pdbe/scop/> Cited in: Session 6.
- Shaanan B (1983). PDB entry 1HHO: Structure of human oxyhaemoglobin at 2.1 Å resolution. RCSB Protein Data Bank. <https://www.rcsb.org/structure/1HHO> Cited in: Session 6.

- STRING. string-db.org. <https://string-db.org/> Cited in: Session 3.
- Tareen A, Kinney JB (2020). Logomaker: beautiful sequence logos in Python. *Bioinformatics* 36(7):2272-2274. doi:10.1093/bioinformatics/btz921. PMID 31821414. Cited in: Session 4.
- TED: The Encyclopedia of Domains database. <https://ted.cathdb.info/> Cited in: Session 6.
- Thompson JD, Higgins DG, Gibson TJ (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res* 22(22):4673-4680. doi:10.1093/nar/22.22.4673. PMID 7984417. Cited in: Session 4, Session 5.
- TorchVision maintainers and contributors. torchvision: PyTorch computer-vision library (ImageNet-pretrained ResNet-18 weights). <https://pytorch.org/vision/> Cited in: Session 10.
- UniProt entry P04637 (human p53) and its AlphaFold DB model. <https://alphafold.ebi.ac.uk/entry/P04637> Cited in: Session 12.
- UniProtKB entry P68871 (HBB\_HUMAN). <https://www.uniprot.org/uniprotkb/P68871/entry> Cited in: Session 3, Session 5.
- Varadi M, Bertoni D, Magana P et al. (2024). AlphaFold Protein Structure Database in 2024: providing structure coverage for over 214 million protein sequences. *Nucleic Acids Res* 52:D368-D375. doi:10.1093/nar/gkad1011. PMID:37933859. Cited in: Session 12.
- Virtanen P, Gommers R, Oliphant TE, et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods* 17:261-272. doi:10.1038/s41592-019-0686-2. PMID:32015543. Cited in: Session 2.
- Visual Studio Code, with the Google Colab extension. Microsoft and Google. <https://code.visualstudio.com/> Cited in: Setup.
- VolcaNoseR web app (Goedhart & Luijsterburg). <https://huygens.science.uva.nl/VolcaNoseR/> Cited in: Session 2.
- Waskom ML (2021). seaborn: statistical data visualization. *Journal of Open Source Software* 6(60):3021. doi:10.21105/joss.03021. Cited in: Session 2.
- Wellcome Sanger Institute. COSMIC. <https://cancer.sanger.ac.uk/cosmic> Cited in: Session 3.
- Wohlwend J, Corso G, Passaro S et al. (2024). Boltz-1 Democratizing Biomolecular Interaction Modeling. *bioRxiv*, 2024.11.19.624167. doi:10.1101/2024.11.19.624167. PMID:39605745. Cited in: Session 12.
- Wolberg WH, Mangasarian OL, Street N, Street W (1995). Breast Cancer Wisconsin (Diagnostic) [dataset]. UCI Machine Learning Repository (CC BY 4.0). doi:10.24432/C5DW2B. Cited in: Session 9.
- Yang J, Shi R, Wei D, Liu Z, Zhao L, Ke B, Pfister H, Ni B (2023). MedMNIST v2 - A large-scale lightweight benchmark for 2D and 3D biomedical image classification. *Sci Data* 10:41. doi:10.1038/s41597-022-01721-8. BloodMNIST subset, CC BY 4.0. Cited in: Session 8, Session 9, Session 10.
- Zdrazil B, Felix E, Hunter F et al. (2024). The ChEMBL Database in 2023: a drug discovery platform spanning multiple bioactivity data types and time periods. *Nucleic Acids Res* 52:D1180-D1192. doi:10.1093/nar/gkad1004. PMID:37933841. Cited in: Session 12.
- Škrinjar P, Eberhardt J, Tauriello G et al. (2025). Have protein-ligand co-

folding methods moved beyond memorisation?. bioRxiv, 2025.02.03.636309. doi:10.1101/2025.02.03.636309. Cited in: Session 12.

## Further reading and watching, by session

Wikipedia articles, textbook chapters, tutorials and videos suggested on each page.

### Welcome

- EMBL-EBI Training. EMBL-EBI Training. <https://www.ebi.ac.uk/training/>
- Käll L. bibook. <https://github.com/statisticalbiotechnology/bibook>
- NCBI. NCBI Bookshelf. <https://www.ncbi.nlm.nih.gov/books/>
- RCSB PDB. RCSB PDB-101. <https://pdb101.rcsb.org/>
- Swaroop CH. A Byte of Python. <https://python.swaroopch.com/>
- Wikipedia. Wikipedia. <https://en.wikipedia.org>

### Setup

- Google Developers Blog (2025). Google Colab is coming to VS Code. <https://developers.googleblog.com/google-colab-is-coming-to-vs-code/>
- Microsoft, VS Code documentation. Remote development tips and tricks. <https://code.visualstudio.com/docs/remote/troubleshooting>
- Microsoft, VS Code documentation. Remote development using SSH. <https://code.visualstudio.com/docs/remote/ssh>
- Microsoft. Get started with OpenSSH for Windows. [https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh\\_install\\_firstuse](https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse)

### Session 1

- Wikipedia. Bayes' theorem. [https://en.wikipedia.org/wiki/Bayes%27\\_theorem](https://en.wikipedia.org/wiki/Bayes%27_theorem)
- Wikipedia. Chain rule. [https://en.wikipedia.org/wiki/Chain\\_rule](https://en.wikipedia.org/wiki/Chain_rule)
- Wikipedia. Conditional probability. [https://en.wikipedia.org/wiki/Conditional\\_probability](https://en.wikipedia.org/wiki/Conditional_probability)
- Wikipedia. Cross product. [https://en.wikipedia.org/wiki/Cross\\_product](https://en.wikipedia.org/wiki/Cross_product)
- Wikipedia. Dot product. [https://en.wikipedia.org/wiki/Dot\\_product](https://en.wikipedia.org/wiki/Dot_product)
- Wikipedia. Eigenvalues and eigenvectors. [https://en.wikipedia.org/wiki/Eigenvalues\\_and\\_eigenvectors](https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors)
- Wikipedia. Exponential function. [https://en.wikipedia.org/wiki/Exponential\\_function](https://en.wikipedia.org/wiki/Exponential_function)
- Wikipedia. False discovery rate. [https://en.wikipedia.org/wiki/False\\_discovery\\_rate](https://en.wikipedia.org/wiki/False_discovery_rate)
- Wikipedia. Gradient. <https://en.wikipedia.org/wiki/Gradient>
- Wikipedia. Integral. <https://en.wikipedia.org/wiki/Integral>

- Wikipedia. Likelihood function. [https://en.wikipedia.org/wiki/Likelihood\\_function](https://en.wikipedia.org/wiki/Likelihood_function)
- Wikipedia. Log probability. [https://en.wikipedia.org/wiki/Log\\_probability](https://en.wikipedia.org/wiki/Log_probability)
- Wikipedia. Logarithm. <https://en.wikipedia.org/wiki/Logarithm>
- Wikipedia. Matrix multiplication. [https://en.wikipedia.org/wiki/Matrix\\_multiplication](https://en.wikipedia.org/wiki/Matrix_multiplication)
- Wikipedia. Matrix norm. [https://en.wikipedia.org/wiki/Matrix\\_norm](https://en.wikipedia.org/wiki/Matrix_norm)
- Wikipedia. Multiple comparisons problem. [https://en.wikipedia.org/wiki/Multiple\\_comparisons\\_problem](https://en.wikipedia.org/wiki/Multiple_comparisons_problem)
- Wikipedia. Norm (mathematics). [https://en.wikipedia.org/wiki/Norm\\_\(mathematics\)](https://en.wikipedia.org/wiki/Norm_(mathematics))
- Wikipedia. Normal distribution. [https://en.wikipedia.org/wiki/Normal\\_distribution](https://en.wikipedia.org/wiki/Normal_distribution)
- Wikipedia. P-value. <https://en.wikipedia.org/wiki/P-value>
- Wikipedia. Pearson correlation coefficient. [https://en.wikipedia.org/wiki/Pearson\\_correlation\\_coefficient](https://en.wikipedia.org/wiki/Pearson_correlation_coefficient)
- Wikipedia. Principal component analysis. [https://en.wikipedia.org/wiki/Principal\\_component\\_analysis](https://en.wikipedia.org/wiki/Principal_component_analysis)
- Wikipedia. Sigmoid function. [https://en.wikipedia.org/wiki/Sigmoid\\_function](https://en.wikipedia.org/wiki/Sigmoid_function)
- Wikipedia. Spearman's rank correlation coefficient. [https://en.wikipedia.org/wiki/Spearman%27s\\_rank\\_correlation\\_coefficient](https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient)
- Wikipedia. Standard deviation. [https://en.wikipedia.org/wiki/Standard\\_deviation](https://en.wikipedia.org/wiki/Standard_deviation)
- Wikipedia. Student's t-test. [https://en.wikipedia.org/wiki/Student%27s\\_t-test](https://en.wikipedia.org/wiki/Student%27s_t-test)
- Wikipedia. Tensor. <https://en.wikipedia.org/wiki/Tensor>
- Wikipedia. Transpose. <https://en.wikipedia.org/wiki/Transpose>
- Wikipedia. Unit vector. [https://en.wikipedia.org/wiki/Unit\\_vector](https://en.wikipedia.org/wiki/Unit_vector)
- Wikipedia. Variance. <https://en.wikipedia.org/wiki/Variance>
- Wikipedia. Volcano plot (statistics). [https://en.wikipedia.org/wiki/Volcano\\_plot\\_\(statistics\)](https://en.wikipedia.org/wiki/Volcano_plot_(statistics))
- Wikipedia. Welch's t-test. [https://en.wikipedia.org/wiki/Welch%27s\\_t-test](https://en.wikipedia.org/wiki/Welch%27s_t-test)
- Zhang, Lipton, Li & Smola, Dive into Deep Learning (d2l.ai). Calculus. [https://d2l.ai/chapter\\_preliminaries/calculus.html](https://d2l.ai/chapter_preliminaries/calculus.html)
- Zhang, Lipton, Li & Smola, Dive into Deep Learning (d2l.ai). Linear Algebra. [https://d2l.ai/chapter\\_preliminaries/linear-algebra.html](https://d2l.ai/chapter_preliminaries/linear-algebra.html)
- Zhang, Lipton, Li & Smola, Dive into Deep Learning (d2l.ai). Preliminaries. [https://d2l.ai/chapter\\_preliminaries/index.html](https://d2l.ai/chapter_preliminaries/index.html)
- Zhang, Lipton, Li & Smola, Dive into Deep Learning (d2l.ai). Probability and Statistics. [https://d2l.ai/chapter\\_preliminaries/probability.html](https://d2l.ai/chapter_preliminaries/probability.html)
- Video: 3Blue1Brown, Essence of Calculus, ch. 2. The paradox of the derivative [video]. YouTube. <https://www.youtube.com/watch?v=9vKqVkMQHKk>
- Video: 3Blue1Brown, Essence of Calculus, ch. 3. Derivative formulas through geometry [video]. YouTube. [https://www.youtube.com/watch?v=S0\\_qX4VJhMQ](https://www.youtube.com/watch?v=S0_qX4VJhMQ)
- Video: 3Blue1Brown, Essence of Calculus, ch. 4. Visualizing the chain rule and product rule [video]. YouTube. <https://www.youtube.com/watch?v=YG15m2VwSjA>
- Video: 3Blue1Brown, Essence of Calculus, ch. 8. Integration and the fundamental theorem of calculus [video]. YouTube. <https://www.youtube.com/watch?v=rfG8ce4nNh0>
- Video: 3Blue1Brown, Essence of Calculus, ch. 9. What does area have to do with

- slope? [video]. YouTube. <https://www.youtube.com/watch?v=FnJqalESC2s>
- Video: 3Blue1Brown, Essence of Linear Algebra, ch. 10. Cross products [video]. YouTube. <https://www.youtube.com/watch?v=eu6i7WJeinw>
- Video: 3Blue1Brown, Essence of Linear Algebra, ch. 14. Eigenvectors and eigenvalues [video]. YouTube. <https://www.youtube.com/watch?v=PFDu9oVAE-g>
- Video: 3Blue1Brown, Essence of Linear Algebra, ch. 9. Dot products and duality [video]. YouTube. <https://www.youtube.com/watch?v=LyGKycYT2v0>
- Video: 3Blue1Brown. Bayes theorem, the geometry of changing beliefs [video]. YouTube. <https://www.youtube.com/watch?v=HZGCoVF3YvM>
- Video: 3Blue1Brown. Binomial distributions [video]. YouTube. <https://www.youtube.com/watch?v=8idr1WZ1A7Q>
- Video: 3Blue1Brown. But what is the Central Limit Theorem? [video]. YouTube. <https://www.youtube.com/watch?v=zeJD6dqJ5lo>
- Video: 3Blue1Brown. Why “probability of 0” does not mean “impossible” [video]. YouTube. <https://www.youtube.com/watch?v=ZA4jkHKZM50>
- Video: Despicable Me (2010). Film, dir. P. Coffin & C. Renaud. Illumination Entertainment / Universal Pictures. [https://en.wikipedia.org/wiki/Despicable\\_Me](https://en.wikipedia.org/wiki/Despicable_Me)
- Video: StatQuest with Josh Starmer. False Discovery Rates, FDR, clearly explained [video]. YouTube. <https://www.youtube.com/watch?v=L3nlGfSyHV0>
- Video: Steven Bradburn. What Is A P-Value? - Clearly Explained [video]. YouTube. <https://www.youtube.com/watch?v=ukcFrzt6cHk>

## Session 2

- Video: 3Blue1Brown. How I animate 3Blue1Brown | A Manim demo with Ben Sparks [video]. YouTube. <https://www.youtube.com/watch?v=rbu7Zu5X1zI>
- Video: Sycamore. Animate Like 3Blue1Brown - No Coding Required (Manim Animation Using A.I.) [video]. YouTube. <https://www.youtube.com/watch?v=InHu0GkOb2s>

## Session 3

- EMBL-EBI Training. The UniProt databases. <https://www.ebi.ac.uk/training/online/courses/uniprot-quick-tour/the-uniprot-databases/>
- EMBL-EBI Training. What is ENA?. <https://www.ebi.ac.uk/training/online/course/s/ena-quick-tour/what-is-ena/>
- NCBI. About RefSeq. <https://www.ncbi.nlm.nih.gov/refseq/about/>
- NCBI. GenBank overview. <https://www.ncbi.nlm.nih.gov/genbank/>
- NCBI. RefSeq FAQ. <https://www.ncbi.nlm.nih.gov/books/NBK50679/>
- UniProt help: Protein existence (the five evidence levels used in every UniProtKB entry). CC BY 4.0. [https://www.uniprot.org/help/protein\\_existence](https://www.uniprot.org/help/protein_existence)
- Wikipedia. Accession number (bioinformatics). [https://en.wikipedia.org/wiki/Accession\\_number\\_\(bioinformatics\)](https://en.wikipedia.org/wiki/Accession_number_(bioinformatics))
- Wikipedia. Biological database. [https://en.wikipedia.org/wiki/Biological\\_database](https://en.wikipedia.org/wiki/Biological_database)
- Wikipedia. BioRxiv. <https://en.wikipedia.org/wiki/BioRxiv>

- Wikipedia. Boolean search. [https://en.wikipedia.org/wiki/Boolean\\_search](https://en.wikipedia.org/wiki/Boolean_search)
- Wikipedia. Ensembl genome database project. [https://en.wikipedia.org/wiki/Ensembl\\_genome\\_database\\_project](https://en.wikipedia.org/wiki/Ensembl_genome_database_project)
- Wikipedia. GenBank. <https://en.wikipedia.org/wiki/GenBank>
- Wikipedia. Genome browser. [https://en.wikipedia.org/wiki/Genome\\_browser](https://en.wikipedia.org/wiki/Genome_browser)
- Wikipedia. International Nucleotide Sequence Database Collaboration. [https://en.wikipedia.org/wiki/International\\_Nucleotide\\_Sequence\\_Database\\_Collaboration](https://en.wikipedia.org/wiki/International_Nucleotide_Sequence_Database_Collaboration)
- Wikipedia. MEDLINE. <https://en.wikipedia.org/wiki/MEDLINE>
- Wikipedia. PubMed Central. [https://en.wikipedia.org/wiki/PubMed\\_Central](https://en.wikipedia.org/wiki/PubMed_Central)
- Wikipedia. PubMed. <https://en.wikipedia.org/wiki/PubMed>
- Wikipedia. RefSeq. <https://en.wikipedia.org/wiki/RefSeq>
- Wikipedia. Sequence database. [https://en.wikipedia.org/wiki/Sequence\\_database](https://en.wikipedia.org/wiki/Sequence_database)
- Wikipedia. UCSC Genome Browser. [https://en.wikipedia.org/wiki/UCSC\\_Genome\\_Browser](https://en.wikipedia.org/wiki/UCSC_Genome_Browser)
- Wikipedia. UniProt. <https://en.wikipedia.org/wiki/UniProt>
- Video: Johns Hopkins Welch Medical Library. PubMed Overview [video]. YouTube. <https://www.youtube.com/watch?v=1otz5qrUbxo>
- Video: UniProt. Welcome to UniProt [video]. YouTube. <https://www.youtube.com/watch?v=OwOJmKmc7VM>

## Session 4

- Wikipedia contributors. "BLOSUM". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/BLOSUM>
- Wikipedia contributors. "Dot plot (bioinformatics)". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Dot\\_plot\\_\(bioinformatics\)](https://en.wikipedia.org/wiki/Dot_plot_(bioinformatics))
- Wikipedia contributors. "Gap penalty". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Gap\\_penalty](https://en.wikipedia.org/wiki/Gap_penalty)
- Wikipedia contributors. "Multiple sequence alignment". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Multiple\\_sequence\\_alignment](https://en.wikipedia.org/wiki/Multiple_sequence_alignment)
- Wikipedia contributors. "Needleman-Wunsch algorithm". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Needleman%E2%80%90Wunsch\\_algorithm](https://en.wikipedia.org/wiki/Needleman%E2%80%90Wunsch_algorithm)
- Wikipedia contributors. "Point accepted mutation". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Point\\_accepted\\_mutation](https://en.wikipedia.org/wiki/Point_accepted_mutation)
- Wikipedia contributors. "Sequence homology". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Sequence\\_homology](https://en.wikipedia.org/wiki/Sequence_homology)
- Wikipedia contributors. "Sequence logo". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Sequence\\_logo](https://en.wikipedia.org/wiki/Sequence_logo)
- Wikipedia contributors. "Smith-Waterman algorithm". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Smith%E2%80%90Waterman\\_algorithm](https://en.wikipedia.org/wiki/Smith%E2%80%90Waterman_algorithm)
- Wikipedia. Extreme value theory. CC BY-SA 4.0. [https://en.wikipedia.org/wiki/Extreme\\_value\\_theory](https://en.wikipedia.org/wiki/Extreme_value_theory)
- Wikipedia. Phylogenetic tree; Neighbor joining; UPGMA. CC BY-SA 4.0. [https://en.wikipedia.org/wiki/Phylogenetic\\_tree](https://en.wikipedia.org/wiki/Phylogenetic_tree)

## Session 5

- NCBI. BLAST Help (BLAST documentation, NCBI Bookshelf NBK1762). National Center for Biotechnology Information, Bethesda MD. <https://www.ncbi.nlm.nih.gov/books/NBK1762/>
- Wikipedia contributors. "BLAST (biotechnology)". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/BLAST\\_\(biotechnology\)](https://en.wikipedia.org/wiki/BLAST_(biotechnology))
- Wikipedia contributors. "E-value". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/E-value>
- Wikipedia contributors. "Hidden Markov model". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Hidden\\_Markov\\_model](https://en.wikipedia.org/wiki/Hidden_Markov_model)
- Wikipedia contributors. "InterPro". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/InterPro>
- Wikipedia contributors. "Pfam". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/Pfam>
- Wikipedia contributors. "Position weight matrix". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Position\\_weight\\_matrix](https://en.wikipedia.org/wiki/Position_weight_matrix)
- Wikipedia contributors. "Pseudocount". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/Pseudocount>
- Wikipedia contributors. "PSI-BLAST". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/PSI-BLAST>
- Video: Edwards R. BLAST 2: seed and extend [video]. San Diego State University, YouTube. <https://www.youtube.com/watch?v=kAAme1fBanc>
- Video: Serrano L. A friendly introduction to Bayes Theorem and Hidden Markov Models [video]. YouTube. <https://www.youtube.com/watch?v=kqSzLo9fenk>

## Session 6

- RCSB Protein Data Bank. PDB-101: educational portal (CC BY 4.0). <https://pdb101.rcsb.org/>
- Wikipedia contributors. Alpha helix. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Alpha\\_helix](https://en.wikipedia.org/wiki/Alpha_helix)
- Wikipedia contributors. Anfinsen's dogma. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Anfinsen%27s\\_dogma](https://en.wikipedia.org/wiki/Anfinsen%27s_dogma)
- Wikipedia contributors. Beta sheet. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Beta\\_sheet](https://en.wikipedia.org/wiki/Beta_sheet)
- Wikipedia contributors. Homology modeling. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Homology\\_modeling](https://en.wikipedia.org/wiki/Homology_modeling)
- Wikipedia contributors. Protein structure. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Protein\\_structure](https://en.wikipedia.org/wiki/Protein_structure)
- Wikipedia contributors. Root-mean-square deviation of atomic positions. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Root-mean-square\\_deviation\\_of\\_atomic\\_positions](https://en.wikipedia.org/wiki/Root-mean-square_deviation_of_atomic_positions)
- Wikipedia contributors. Template modeling score. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Template\\_modeling\\_score](https://en.wikipedia.org/wiki/Template_modeling_score)

- Wikipedia contributors. Threading (protein sequence). Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Threading\\_\(protein\\_sequence\)](https://en.wikipedia.org/wiki/Threading_(protein_sequence))

## Session 7

- Rosenblatt F (1958). The design of an intelligent automaton. Research Reviews, Office of Naval Research, October 1958, pp. 5-13. Figure via Wikimedia Commons (public domain). [https://commons.wikimedia.org/wiki/File:Organization\\_of\\_a\\_biological\\_brain\\_and\\_a\\_perceptron.png](https://commons.wikimedia.org/wiki/File:Organization_of_a_biological_brain_and_a_perceptron.png)
- Wikipedia contributors. "AI winter". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/AI\\_winter](https://en.wikipedia.org/wiki/AI_winter)
- Wikipedia contributors. "Alpha helix". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Alpha\\_helix](https://en.wikipedia.org/wiki/Alpha_helix)
- Wikipedia contributors. "Backpropagation". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/Backpropagation>
- Wikipedia contributors. "Beta sheet". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Beta\\_sheet](https://en.wikipedia.org/wiki/Beta_sheet)
- Wikipedia contributors. "Gradient descent". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Gradient\\_descent](https://en.wikipedia.org/wiki/Gradient_descent)
- Wikipedia contributors. "Linear separability". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Linear\\_separability](https://en.wikipedia.org/wiki/Linear_separability)
- Wikipedia contributors. "Perceptron". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/Perceptron>
- Wikipedia contributors. "Perceptrons (book)". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Perceptrons\\_\(book\)](https://en.wikipedia.org/wiki/Perceptrons_(book))
- Wikipedia contributors. "Protein secondary structure". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Protein\\_secondary\\_structure](https://en.wikipedia.org/wiki/Protein_secondary_structure)
- Wikipedia contributors. "Protein structure prediction". Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Protein\\_structure\\_prediction](https://en.wikipedia.org/wiki/Protein_structure_prediction)
- Video: 3Blue1Brown (Sanderson G) (2017). But what is a neural network? Deep Learning, chapter 1 [video]. YouTube. [https://www.youtube.com/watch?v=aircAr\\_uvnKk](https://www.youtube.com/watch?v=aircAr_uvnKk)
- Video: 3Blue1Brown (Sanderson G) (2017). Gradient descent, how neural networks learn. Deep Learning, chapter 2 [video]. YouTube. <https://www.youtube.com/watch?v=IHZwWFHwa-w>

## Session 8

- UniProt Consortium. UniProt help: Subcellular location (CC BY 4.0). [https://www.uniprot.org/help/subcellular\\_location](https://www.uniprot.org/help/subcellular_location)
- Wikipedia contributors. Activation function. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Activation\\_function](https://en.wikipedia.org/wiki/Activation_function)

- Wikipedia contributors. Batch effect. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Batch\\_effect](https://en.wikipedia.org/wiki/Batch_effect)
- Wikipedia contributors. CASP. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/CASP>
- Wikipedia contributors. Chou-Fasman method. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Chou%E2%80%93Fasman\\_method](https://en.wikipedia.org/wiki/Chou%E2%80%93Fasman_method)
- Wikipedia contributors. Coefficient of determination. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Coefficient\\_of\\_determination](https://en.wikipedia.org/wiki/Coefficient_of_determination)
- Wikipedia contributors. Confusion matrix. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Confusion\\_matrix](https://en.wikipedia.org/wiki/Confusion_matrix)
- Wikipedia contributors. Cross-validation (statistics). Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Cross-validation\\_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))
- Wikipedia contributors. Hydrophilicity plot. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Hydrophilicity\\_plot](https://en.wikipedia.org/wiki/Hydrophilicity_plot)
- Wikipedia contributors. Leakage (machine learning). Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Leakage\\_\(machine\\_learning\)](https://en.wikipedia.org/wiki/Leakage_(machine_learning))
- Wikipedia contributors. Logistic regression. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Logistic\\_regression](https://en.wikipedia.org/wiki/Logistic_regression)
- Wikipedia contributors. Matthews correlation coefficient (Phi coefficient). Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Phi\\_coefficient](https://en.wikipedia.org/wiki/Phi_coefficient)
- Wikipedia contributors. Neural network (machine learning). Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Neural\\_network\\_\(machine\\_learning\)](https://en.wikipedia.org/wiki/Neural_network_(machine_learning))
- Wikipedia contributors. Neuron. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/Neuron>
- Wikipedia contributors. Precision and recall. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)
- Wikipedia contributors. Receiver operating characteristic. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](https://en.wikipedia.org/wiki/Receiver_operating_characteristic)
- Wikipedia contributors. Regression analysis. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Regression\\_analysis](https://en.wikipedia.org/wiki/Regression_analysis)
- Wikipedia contributors. Softmax function. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Softmax\\_function](https://en.wikipedia.org/wiki/Softmax_function)
- Wikipedia contributors. Statistical classification. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Statistical\\_classification](https://en.wikipedia.org/wiki/Statistical_classification)
- Wikipedia contributors. Supervised learning. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Supervised\\_learning](https://en.wikipedia.org/wiki/Supervised_learning)
- Wikipedia contributors. Training, validation, and test data sets. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Training,\\_validation,\\_and\\_test\\_data\\_sets](https://en.wikipedia.org/wiki/Training,_validation,_and_test_data_sets)
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, “Builders’ Guide”. d2l.ai (CC BY-SA 4.0). [https://d2l.ai/chapter\\_builders-guide/index.html](https://d2l.ai/chapter_builders-guide/index.html)
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, “Generalization in Classification”. d2l.ai (CC BY-SA 4.0). [https://d2l.ai/chapter\\_linear-classification/generalization-classification.html](https://d2l.ai/chapter_linear-classification/generalization-classification.html)

- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, “Linear Regression”. d2l.ai (CC BY-SA 4.0). [https://d2l.ai/chapter\\_linear-regression/linear-regression.html](https://d2l.ai/chapter_linear-regression/linear-regression.html)
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, “Softmax Regression”. d2l.ai (CC BY-SA 4.0). [https://d2l.ai/chapter\\_linear-classification/softmax-regression.html](https://d2l.ai/chapter_linear-classification/softmax-regression.html)

## Session 9

- Wikipedia contributors. Backpropagation. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/Backpropagation>
- Wikipedia contributors. Dropout (neural networks). Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Dropout\\_\(neural\\_networks\)](https://en.wikipedia.org/wiki/Dropout_(neural_networks))
- Wikipedia contributors. Early stopping. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Early\\_stopping](https://en.wikipedia.org/wiki/Early_stopping)
- Wikipedia contributors. Overfitting. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). <https://en.wikipedia.org/wiki/Overfitting>
- Wikipedia contributors. Regularization (mathematics). Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Regularization\\_\(mathematics\)](https://en.wikipedia.org/wiki/Regularization_(mathematics))
- Wikipedia contributors. Stochastic gradient descent. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Stochastic\\_gradient\\_descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent)
- Wikipedia contributors. Vanishing gradient problem. Wikipedia, The Free Encyclopedia (CC BY-SA 4.0). [https://en.wikipedia.org/wiki/Vanishing\\_gradient\\_problem](https://en.wikipedia.org/wiki/Vanishing_gradient_problem)
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, “Linear Regression”. d2l.ai (CC BY-SA 4.0). [https://d2l.ai/chapter\\_linear-regression/linear-regression.html](https://d2l.ai/chapter_linear-regression/linear-regression.html)
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, “Multilayer Perceptrons”. d2l.ai (CC BY-SA 4.0). [https://d2l.ai/chapter\\_multilayer-perceptrons/index.html](https://d2l.ai/chapter_multilayer-perceptrons/index.html)
- Video: 3Blue1Brown (Sanderson G). Backpropagation calculus. Deep Learning, chapter 4. YouTube. <https://www.youtube.com/watch?v=tIeHLnjs5U8>
- Video: 3Blue1Brown (Sanderson G). Backpropagation, intuitively. Deep Learning, chapter 3. YouTube. <https://www.youtube.com/watch?v=Ilg3gGewQ5U>
- Video: 3Blue1Brown (Sanderson G). But what is a neural network? Deep Learning, chapter 1. YouTube. <https://www.youtube.com/watch?v=aircAruvnKk>
- Video: 3Blue1Brown (Sanderson G). But what is cross-entropy? Compression is Intelligence, part 2. YouTube. <https://www.youtube.com/watch?v=GLYgs6v2YfU>

## Session 10

- Wikipedia contributors. “U-Net”. Wikipedia, The Free Encyclopedia. <https://en.wikipedia.org/wiki/U-Net>
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, ch. “Convolutional Neural Networks” (CC BY-SA 4.0). [https://d2l.ai/chapter\\_convolutional-neural-networks/index.html](https://d2l.ai/chapter_convolutional-neural-networks/index.html)

- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, ch. “Modern Convolutional Neural Networks” (CC BY-SA 4.0). [https://d2l.ai/chapter\\_convolutional-modern/index.html](https://d2l.ai/chapter_convolutional-modern/index.html)
- Video: 3Blue1Brown (Grant Sanderson). “But what is a convolution?” YouTube. <https://www.youtube.com/watch?v=KuXjwB4LzSA>

## Session 11

- The Nobel Prize in Physiology or Medicine 1999: Günter Blobel. NobelPrize.org. <https://www.nobelprize.org/prizes/medicine/1999/summary/>
- Wikipedia contributors. “Long short-term memory”. Wikipedia, The Free Encyclopedia. [https://en.wikipedia.org/wiki/Long\\_short-term\\_memory](https://en.wikipedia.org/wiki/Long_short-term_memory)
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, ch. “Attention Mechanisms and Transformers” (CC BY-SA 4.0). [https://d2l.ai/chapter\\_attention-mechanisms-and-transformers/index.html](https://d2l.ai/chapter_attention-mechanisms-and-transformers/index.html)
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, ch. “Modern Recurrent Neural Networks” (CC BY-SA 4.0). [https://d2l.ai/chapter\\_recurrent-modern/index.html](https://d2l.ai/chapter_recurrent-modern/index.html)
- Zhang A, Lipton ZC, Li M, Smola AJ. Dive into Deep Learning, ch. “Recurrent Neural Networks” (CC BY-SA 4.0). [https://d2l.ai/chapter\\_recurrent-neural-networks/index.html](https://d2l.ai/chapter_recurrent-neural-networks/index.html)
- Video: 3Blue1Brown (Grant Sanderson). “Attention in transformers, step-by-step” (Deep Learning ch. 6). YouTube. <https://www.youtube.com/watch?v=eMlx5fFN0Yc>
- Video: 3Blue1Brown (Grant Sanderson). “How might LLMs store facts” (Deep Learning ch. 7). YouTube. <https://www.youtube.com/watch?v=9-Jl0dxWQs8>
- Video: 3Blue1Brown (Grant Sanderson). “Large Language Models explained briefly”. YouTube. <https://www.youtube.com/watch?v=LPZh9BOjkQs>
- Video: 3Blue1Brown (Grant Sanderson). “Transformers, the tech behind LLMs” (Deep Learning ch. 5). YouTube. <https://www.youtube.com/watch?v=wjZofjX0v4M>

## Session 12

- The Nobel Prize in Chemistry 2024 (Baker; Hassabis & Jumper). NobelPrize.org. <https://www.nobelprize.org/prizes/chemistry/2024/summary/>
- Video: AI Coffee Break with Letitia (Parcalabescu L). “Flow-Matching vs Diffusion Models explained side by side”. YouTube. [https://www.youtube.com/watch?v=fiRjwZ\\_6KI](https://www.youtube.com/watch?v=fiRjwZ_6KI)
- Video: Kohl S (DeepMind). “Highly accurate protein structure prediction with AlphaFold”. YouTube. <https://www.youtube.com/watch?v=92j-7b-sRPw>

## What isn't reused, and why

- **Magnus Ekman's *Learning Deep Learning*** and **Jonathan Pevsner's *Bioinformatics and Functional Genomics*** were this course's previously assigned textbooks. Both are under conventional copyright. Nothing from either is reproduced here; some pages credit Ekman's book (and the teachers' own earlier lecture slides) as an internal source of ideas, and any resemblance in topic ordering is because these books teach the same standard material.
- **Goodfellow, Bengio & Courville's *Deep Learning*** is free to read online but not licensed for reuse or redistribution, so it is treated like the two textbooks above.
- **Michael Nielsen's *Neural Networks and Deep Learning*** is CC BY-NC: freely reusable for non-commercial teaching, but an earlier draft that used it was switched to *Dive into Deep Learning* throughout, to keep one license without a non-commercial restriction across the whole book.