Structured Learning for Structural Bioinformatics

Applications of Deep Learning to Protein Structure Prediction

David Menéndez Hurtado
Structured Learning for Structural Bioinformatics
Applications of Deep Learning to Protein Structure Prediction
David Menéndez Hurtado

Abstract
Proteins are the basic molecular machines of the cell, performing a broad range of tasks, from structural support to catalysis of chemical reactions. Their function is determined by their 3D structure, which in turn is dictated by the order of their components, the amino acids.

This thesis is dedicated to applications of machine learning to the problems of contact prediction, \textit{ab-initio}, and model quality assessment. In particular, my research has been focused on developing methods that are both effective, and easy to use.

In the first paper, we improved the already state-of-the-art model quality assessment (MQA) program ProQ3 replacing the underlying machine learning algorithm from svm to Deep Learning, baptised ProQ3D. The correlation between predicted and true scores was improved from 0.85 to 0.90, using the same training data and features.

The second paper joined several programs into a single pipeline for \textit{ab-initio} structure prediction: contact prediction, folding, and model selection. We attempted to predict the structures of all 6379 PFAM families with unknown structure, of which 558 we believe to be accurate. Of these, 415 had not been reported before.

The third paper uses advances in machine learning to build a contact predictor, PconsC4, that is fast and easy to deploy in large-scale studies, since it requires a single Multiple Sequence Alignment (MSA), and no external dependencies. The predictions are state-of-the-art, yielding a 12% improvement in precision over PconsC3, and 244 times faster.

With ProQ4, in the fourth paper, we introduce a novel way of training deep networks for MQA in a way that minimises the bias of the training data, and emphasises model ranking, and demonstrate its viability with a minimal description of the protein. The ranking correlation was improved with respect to ProQ3D from 0.82 to 0.90.

Lastly, in the fifth paper, we show the results of ProQ3D and ProQ4 in a completely blind test: CASP13.
STRUCTURED LEARNING FOR STRUCTURAL BIOINFORMATICS

David Menéndez Hurtado
Structured Learning for Structural Bioinformatics
Applications of Deep Learning to Protein Structure Prediction

David Menéndez Hurtado
ABSTRACT

Proteins are the basic molecular machines of the cell, performing a broad range of tasks, from structural support to catalysis of chemical reactions. Their function is determined by their 3D structure, which in turn is dictated by the order of their components, the amino acids.

This thesis is dedicated to applications of machine learning to the problems of contact prediction, ab-initio, and model quality assessment. In particular, my research has been focused on developing methods that are both effective, and easy to use.

In the first paper, we improved the already state-of-the-art model quality assessment (MQA) program ProQ3 replacing the underlying machine learning algorithm from SVM to Deep Learning, baptised ProQ3D. The correlation between predicted and true scores was improved from 0.85 to 0.90, using the same training data and features.

The second paper joined several programs into a single pipeline for ab-initio structure prediction: contact prediction, folding, and model selection. We attempted to predict the structures of all 6379 Pfam families with unknown structure, of which 558 we believe to be accurate. Of these, 415 had not been reported before.

The third paper uses advances in machine learning to build a contact predictor, PconsC4, that is fast and easy to deploy in large-scale studies, since it requires a single Multiple Sequence Alignment (MSA), and no external dependencies. The predictions are state-of-the-art, yielding a 12% improvement in precision over PconsC3, and 244 times faster.

With ProQ4, in the fourth paper, we introduce a novel way of training deep networks for MQA in a way that minimises the bias of the training data, and emphasises model ranking, and demonstrate its viability with a minimal description of the protein. The ranking correlation was improved with respect to ProQ3D from 0.82 to 0.90. Lastly, in the fifth paper, we show the results of ProQ3D and ProQ4 in a completely blind test: CASP13.
Proteiner är grundläggande molekylära maskiner i cellen. Experimentella studier är oftast arbetskraftsintensiva och dyra; och många av dem misslyckas utan att ge resultat. Ett av målen av bioinformatik är att, om möjligt, ersätta dessa experiment med beräkningsmetoder.

Den mänskliga impulsen är att kategorisera och modellera allt baserat på hårda och enkla regler. Men naturen är rölig och motstår att följa dessa regler. En av lösningarna som lades fram var användningen av maskininlärning: mjuka och komplicerade regler, som datorn når efter att ha tittat på data. Reglerna är ”mjuka” eftersom de är probabilistiska och ”komplicerade” eftersom de är resultatet av att analysera stora datasätt.

Under det senaste decenniet har djupinlärning revolutionerat området maskininlärning, särskilt inom datorsyn och taligenkänning. Det främsta skälet till dess framgång är förmågan att träna mycket flexibla modeller, som kan fånga den verkliga världen, på stora mängder data. Men nyckeln som gör det möjligt är förmågan att utnyttja strukturen i data.

Detta arbete presenterar tillämpningen av maskininlärning i allmänhet och djupinlärning i synnerhet på flera uppgifter inom området för förutsägelse av proteinstrukturen: kontaktprognos, \textit{ab-initio} modellering, och modellkvalitetsbedömning.

Fokus för min forskning har varit att undersöka hur man kan representera proteindata på ett sätt som är lämpligt för djupinlärning. Mer konkret har jag arbetat med att utveckla metoder som är effektiva men ändå enkla att installera och använda.
PAPERS INCLUDED IN THIS THESIS

PAPER I


PAPER II


PAPER III


PAPER IV

David Menéndez Hurtado, Karolis Uziela, and Arne Elofsson, A novel training procedure to train deep networks in the assessment of the quality of protein models. *Manuscript*

PAPER V

Jianlin Cheng†, Myong-Ho Choe†, Arne Elofsson†, Kun-Sop Han†, Jie Hou†, Ali H. A. Maghrabi†, Liam J. McGuffin†, David Menéndez Hurtado†, Kliment Olechnovic†, Torsten Schwede†, Gabriel Studer†, Karolis Uziela†, Ceslovas Venclovas†, and Björn Wallner†. Estimation of model accuracy in casp13. *Proteins*, 2019; 1–17. URL: https://doi.org/10.1002/prot.25767.

† indicates shared first author.
OTHER PAPERS NOT INCLUDED


CONTENTS

Introduction viii

I THE INFORMATICS

1 MACHINE LEARNING 3

  1.1 Classification and typology 4
      1.1.1 Do we have labels? 4
      1.1.2 Categorical or continuous? 4

  1.2 The machine learning spectrum 4

  1.3 Training and test sets 5

  1.4 Evaluating predictions: figures of merit 5
      1.4.1 Classification problems 5
      1.4.2 Regression problems 6

  1.5 Traditional machine learning 6
      1.5.1 Linear regression 6
      1.5.2 Logistic regression 7
      1.5.3 Multi-Layer Perceptron 8
      1.5.4 Support Vector Machines 9
      1.5.5 $k$-Nearest Neighbour 10
      1.5.6 Decision tree 10
      1.5.7 Random Forest 12
      1.5.8 Gaussian Processes 12
      1.5.9 Isotonic regression 13

  1.6 On the wrongness of machine learning 13

2 DEEP LEARNING 15

  2.1 The basic blocks 15
      2.1.1 Fully connected layers 15
      2.1.2 Convolutions 16
      2.1.3 Recurrent 16
      2.1.4 Non-linearities 17

  2.2 Gradient descent 18
      2.2.1 Back-propagation 19
      2.2.2 Stochastic gradient descent 20
      2.2.3 Optimisers 20

  2.3 Taming the complexity: regularisation 22

  2.4 Tensors and gradients 24

  2.5 The quest for depth 24

  2.6 Deep transfer learning 26
      2.6.1 Multiple tasks 27
      2.6.2 Representation learning 27
II THE BIOLOGY
3 PROTEINS  31
  3.1 What are proteins?  31
  3.2 The many levels of protein structures  31
    3.2.1 The amino acids  31
    3.2.2 Primary structure or sequence  33
    3.2.3 Secondary structure  33
    3.2.4 Tertiary structure  33
    3.2.5 Quaternary structure  33
  3.3 Protein biochemistry  33
    3.3.1 The hydrophobic effect  36
    3.3.2 From molten globule to structure  37
    3.3.3 Energy terms  37
  3.4 Experimental methods  38
    3.4.1 Partial restraints  39
4 THE INFORMATICS OF BIOLOGY  41
  4.1 Predictors  41
  4.2 Multiple Sequence Alignments  41
    4.2.1 Pairwise alignment  42
    4.2.2 Scaling up: BLAST  42
    4.2.3 PSSMs and PSI-BLAST  42
    4.2.4 Hidden Markov Models  43
  4.3 Protein structure prediction  44
    4.3.1 Physics-based modelling  44
    4.3.2 Homology modelling  44
    4.3.3 Ab-initio folding  44
    4.3.4 Evaluation  45
    4.3.5 Model Quality Assessment  46
  4.4 Contact prediction  47
    4.4.1 Mutual Information  47
    4.4.2 Direct Coupling Analysis  48
    4.4.3 Phylogenetic bias  49
    4.4.4 Pattern recognition  50
  4.5 Assessing the field: CASP  50

III MY WORK
5 PAPER CONTRIBUTIONS  53
INTRODUCTION

THE PRESENT THESIS is an introduction to help understand the work that I have done during my PhD studies. The guiding principle that I used to compose it are all the things I wish I had known when I started, in the hope that it would serve as a starting point for future generations of deep learning bioinformaticians.

This thesis is divided into three parts:

Part I introduces machine learning, the workhorse of my work.

Part II explains the biological and biochemical underpinnings.

Part III summarizes the content of each paper in this thesis.

Following are the papers as published in the journals. The different sections are loosely coupled, so the reader is allowed and encouraged to skim or skip the parts that they are familiar with.

I have made extensive use of margin notes to guide the reader, but leaving more than enough space for notes, comments, doodles, or theorems.
Part I
THE INFORMATICS
MACHINE LEARNING

The Scientific Revolution during the Renaissance was fuelled by the realisation that Nature can be parametrised using only a handful of equations. With these tools and sufficient measurements, natural philosophers could use the universal equations to fully model the mechanical universe. In the words of de Carrança (1582): “The arts that belong to the common use of mankind have a generality link because Nature is mother of all things, and common to all”.

As mathematical tools improved, increasingly more complex systems could be analysed, such as orbits of astrophysical objects or complex optical designs. Furthermore, the later development of computers allowed for an explosion in the size and breadth of tractable problems, such as dynamics of atomic nuclei, or weather forecast.

These applications were either developed from first principles or as effective theories; but contained at their core, relatively simple mathematical models. For example, the dynamics of \( n \)-bodies gravitationally interacting are very complicated to solve, but it all spawn from two simple equations, Newton’s second law of motion for an object of mass \( m \): \( \vec{F} = m \cdot \vec{a} \) and the force of gravity between two masses \( m_1 \) and \( m_2 \): \( F_g = G \frac{m_1 m_2}{r^2} \).

The advent of computers not only gave us mathematical muscle, but it also allowed to connect them to sensors; providing us with large amounts of data about the real world. Data that can be used to fit statistical models, even in the absence of underlying mathematical theories, such as recognising objects in images, identifying where in the cell a protein is going to end up, or translating natural language. The Scientific Revolution of the 16th century brought the concept “if it can be measured, it can be modelled”, but the Data Revolution of the 20th century expanded it “if it can be represented, it can be modelled”. We went from measuring the variables present in our equations to represent the world in numbers on a computer.

Machine learning is the study of statistical models that can create inferences from collections of examples. In other words, a machine learning model is capable of producing generalised rules from a set of particular cases, an example of induction. The models can be as concrete as relating the voltage and measured current intensity in a circuit, as complex as relating the sensory input of a rocket with its control, or as abstract as mapping natural images to the text describing its contents.

Instead of designing algorithms that simulate reality from first principles, that can be insurmountably complex and time-consuming; a machine learning model uses data: the algorithm depends on a series of free parameters that are deduced from examples. This flexibility comes with a price: the programmer has relinquished the control to the dataset, and sacrificed the possibility of a mechanistic interpretation. More on this in Section 1.6.
1.1 CLASSIFICATION AND TYPOLOGY

Machine learning tasks can be classified according to several criteria. Here are, in broad strokes, some of the main types that cover the majority of the machine learning problems according to different criteria.

1.1.1 Do we have labels?

- Unsupervised: we do not have data with annotated target values. Examples: clustering of DNA sequences, dimensionality reduction.

- Supervised: our training data has assigned labels, and we want to predict them to new data. Examples: protein secondary structure prediction, linear regression.

  The focus of this thesis will be on supervised tasks.

1.1.2 Are they categorical or continuous?

The supervised tasks can be again divided depending on the nature of the labels:

- Classification: our labels are categorical variables. Examples: image recognition, automated transcription of speech, presence or absence of tumours, protein subcellular localisation.

- Regression: we are interested in the value of continuous variables. Examples: curve fitting, counting.

1.2 THE MACHINE LEARNING SPECTRUM

We can design machine learning models with different degrees of restrictions, or parametric assumptions. A more restricted model needs less data to converge, and if the underlying assumptions are correct, they will not hinder its performance. On the other hand, if these restrictions are not accurate, the model will be biased and its performance, limited.

If we instead relax the parametric assumptions we obtain a more flexible model, capable of tackling more complex problems. But this versatility comes with a cost: they require more data to train.

A theoretical result

Can we take it to the extreme? Can we train a model completely free of assumptions in the case of infinite data? The No Free Lunch Theorem (Wolpert, 1996) says, averaging over all problems, all algorithms are equally good. In other words, without inputting domain knowledge that restricts the space of possibilities, we cannot do better than random. For a given training dataset, an infinite number of models can fit it equally well, while giving completely different answers for any point outside of that. Domain knowledge can provide us with constraints – for example, certain physical models must be continuous – or point at the relevant inputs – weather prediction would depend on temperature, but not on the day of the week – and give us a measure of
simplicity – for models of equal quality, Ockam’s razor would slash away the more complex ones.

1.3 training and test sets

To train a supervised model, we need a set of labelled examples, called the \textit{training set}, and a separate and independent set – the \textit{test set} – to evaluate its performance. Why do we need a separate test set? It is easy to come up with rules that are perfectly accurate on the training set, but do not generalise well, a phenomenon called \textit{overfitting}. Thus, we will use the performance on the test set as an estimation of the performance on real-world data. When presented with a collection of models, we will prefer the ones that have the best performance on the test set.

1.4 evaluating predictions: figures of merit

How can we compare them? Figures of merit allow us to evaluate how well the model performs. Here I will describe some of the most commonly used:

1.4.1 Classification problems

The most obvious measure is the \textit{accuracy}, or the proportion of predictions that are correct. This is easy to implement and interpret, but it is of limited use on unbalanced datasets: when the number of positive and negative classes are very different. For example, if 99\% of our data are negative examples, a model that \textit{always} predicts the negative class has an accuracy of 99\%! Yet, it is useless in practice because it does not add any new information.

In order to get a more nuanced view we should consider the \textit{precision} – the fraction of correctly predicted positive examples – and \textit{recall} – the fraction of positive examples correctly identified.

\[
\text{precision} = \frac{\text{True Positives}}{\text{Predicted Positives}} \\
\text{recall} = \frac{\text{True Positives}}{\text{All Positives}}
\]

Precision is also called \textit{Positive Predictive Value (ppv)}, specially in the context of contact prediction.

A combination of both is the \textit{F}_1 \textit{score}: the harmonic mean of precision and recall:

\[
\text{F}_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}
\]
1.4.2 Regression problems

When predicting a continuous variable, the simplest measure is the Mean Squared Error, or mse. Often, we want the measure to be in the same units as the input, so we take the square root. We may also be interested in the correlation between predicted – \( p \) – and true – \( t \) – which can be expressed by the Pearson correlation \( R \):

\[
R = \frac{\sum_{i=1}^{N}(p_i - \bar{p})(t_i - \bar{t})}{\sqrt{\sum_{i=1}^{N}(p_i - \bar{p})^2 \sqrt{\sum_{i=1}^{N}(t_i - \bar{t})^2}}}
\]

This measures the linear correlation between two variables, and can be affected by outliers or skewed distributions. An alternative is the Spearman correlation, which computes the Pearson R on the indexes, thus measuring the correlation between rankings.

1.5 A point of comparison: Traditional Machine Learning

In this section, I will give an overview of some of the most popular supervised machine learning algorithms to illustrate how they work, and the type of underlying assumptions they operate under. I will describe them in their simplest form, be it for regression or classification, but both can be easily generalised: a regression algorithm can be turned to a binary classifier mapping the positive and negative labels to \( (1, -1) \) or vice-versa. A binary classifier can be used in a problem with \( N \) classes by either training \( N \) binary classifiers for each class versus the rest, or all the \( \binom{N}{2} \) pairwise binary classifications.

1.5.1 Linear regression: Ordinary Least Squares, Ridge, and Lasso

Ordinary Least Squares (\( ols \)) is the simplest linear regression model: the linear combination of the inputs that minimises the squared error. For a matrix \( \tilde{X} \) of observations and a vector of target \( \tilde{y} \), \( ols \) finds the vector \( \tilde{w} = (w_0, w_1, ..., w_d) \) that minimises the loss:

\[
L_{ols} = ||\tilde{X}\tilde{w} - \tilde{y}||_2^2,
\]

where \( d \) is the number of dimensions of the inputs, ie., the number of input features.

This method is simple and can be solved efficiently by linear algebra libraries through, for example, an \( LU \) decomposition\(^1\). But if there is co-linearity between input features, the matrix \( \tilde{X} \) is close to singular, so its inverse can become numerically unstable.

A simple solution is to add a term that tends to shrink the coefficients of \( \tilde{w} \), and makes the solution unique even in the singular case:

\(^1\) A standard procedure to solve systems of equations, where the matrix \( \tilde{X} \) is factored into the product of a lower and an upper triangular matrices, without needing to explicitly invert the matrix. Then, the components of \( \tilde{w} \) can be obtained iteratively.
1.5 Traditional Machine Learning

\[ L_{\text{Ridge}} = ||\tilde{X}\vec{w} - \vec{y}||_2^2 + \lambda||\vec{w}||_2^2, \]

where \( \lambda \) is a number that regulates the strength of this shrinkage. Since the penalisation depends on the \( L^2 \) norm of the vector \( \vec{w} \), the weight of co-linear features will be “distributed” amongst them.

Sometimes, we want sparse weights, for example, if we know some features are irrelevant, but we do not know which ones. We can then use \( L^1 \) regularisation:

\[ L_{\text{Lasso}} = ||\tilde{X}\vec{w} - \vec{y}||_2^2 + \alpha||\vec{w}||_1, \]

where \( \alpha \) is our new regularisation strength parameter. The higher it is, the more weights will be close to 0.

This is a good point to introduce some important vocabulary in machine learning:

- **Parameters** are a set of numbers involved in the algorithm inferred from the data. In this case, the components of the vector \( \vec{w} \) are the parameters learned.

- **Hyperparameters** are the settings decided on by the human. In this case, the strength of the regularisations, \( \alpha \) and \( \lambda \), are hyperparameters, but also the decision to use one or the other.

1.5.2 Logistic regression

Logistic regression, despite its name, is an adaptation of linear models for classification. The output of the linear model is wrapped by a logistic function:

\[ f(\tilde{x}) = \frac{1}{1 + e^{-\tilde{\vec{w}} \cdot \tilde{x}}} \]

It can also be generalised to \( N \) multiple mutually exclusive classes. Each class has one linear model given by the vector of weights \( \{\tilde{\vec{w}}_i\}_{i=1}^N \):

\[
\begin{bmatrix}
z_1 \\
z_2 \\
\vdots \\
z_N
\end{bmatrix} =
\begin{bmatrix}
\tilde{\vec{w}}_{1\tilde{x}} \\
\tilde{\vec{w}}_{2\tilde{x}} \\
\vdots \\
\tilde{\vec{w}}_{N\tilde{x}}
\end{bmatrix} = \tilde{\vec{W}} \cdot \tilde{x},
\]

where \( \tilde{\vec{W}} \) is an \( (N \times d) \) matrix. The components of the vector \( \vec{z} \) are called the **logits**. To obtain probabilities we use the exponential to make them all positive and normalise the sum. This is called the softmax function:

\[ \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \]

If the logits are scaled up, the softmax will be more concentrated around the maximum, suggesting a more confident prediction. As before, we can apply \( L^1 \) or \( L^2 \) regularisations, which imply lower coefficients on \( \vec{w} \), ergo lower values for the logits. A stronger regularisation returns a less confident predictor, as shown in Figure 1.1. In particular, 1.1b, has a much narrower region of uncertain predictions, hence it is more confident.
1.5.3 Multi-Layer Perceptron

A linear model on $d$ features is limited to $d$ free parameters, or at most $d \cdot N$ for $N$ classes, and is limited to linear functions. When we have more complicated tasks, and more data for them, we would like to be able to learn more parameters. A simple way to do so is to build upon linear models, and have an intermediate hidden state $\hat{h}$ of arbitrary dimension. The vector of probabilities for every class is then:

$$\hat{h} = \hat{W}_1 \cdot \hat{x}$$
$$\hat{z} = \hat{W}_2 \cdot \hat{h}$$
$$p(\hat{x} \in i) = \sigma(\hat{z}_i)$$

where the last line is only necessary for classification.

Although we now have more free parameters – the components of the matrices $\hat{W}_1$ and $\hat{W}_2$ – the model is exactly equivalent to a linear model, since the two matrices $\hat{W}_1$ and $\hat{W}_2$ act right one after the other, so can be multiplied together to recover the equivalent of our old matrix $\hat{W}$. This can be fixed by introducing an arbitrary non-linear function $f$ to the hidden layer:

$$\hat{h} = \hat{W}_1 \cdot \hat{x}$$
$$\hat{z} = \hat{W}_2 \cdot f(\hat{h})$$
$$p(\hat{x} \in i) = \sigma(\hat{z}_i)$$

Since we have no constraints over the properties of $f$, common choices are simple point-wise functions, such as $\tanh(x)$; the logistic, $\frac{1}{1+e^{-x}}$; or the Rectified Linear Unit, $\text{ReLU}(x) = [x \text{ if } x > 0 \text{ else } 0]$.

More layers! We can repeat the same procedure and chain two hidden layers:
1.5.4 Support Vector Machines

A Support Vector Machine, or SVM, is in its simplest form, a supervised, binary classification algorithm that tries to find the hyperplane that maximises the separation of the two groups. To account for noise, a slack parameter, that will ignore points that are too close to the boundary can be included. Figure 1.2 illustrates an example of classifying two species from the classic Iris dataset (Fisher, 1936). It consists on measurements of the length and width of petals and sepals of three species of Iris flowers, *Iris setosa*, *Iris versicolor*, and *Iris virginica*, taken from the same field, on the same day, by the same person using the same instrument.

It can be generalised to non-flat boundaries using the so-called kernel trick, where Euclidean distances are replaced with an arbitrary measure of similarity given by positive-definite kernel function:

$$
||\tilde{x}_1 - \tilde{x}_2|| \rightarrow k(\tilde{x}_1, \tilde{x}_2)
$$

For example, in Figure 1.2b we have used a Radial Basis Function (rbf):
which implicitly projects the data into an infinite-dimensional space.
Another way of interpreting the kernel trick is to think of it as learning a topological transformation of the space that makes the data separable by the final hyperplane, a linear function.

1.5.5 k-Nearest Neighbour

The Nearest Neighbour classifier takes the \( k \) closest points in the training set and predicts the most common label. The choice of \( k \) is a balance between noise and flexibility: smaller values give more distinct frontiers but are more susceptible to noise. See it illustrated in the Figure 1.3.

1.5.6 Decision tree

Decision trees are based on a measure of impurity of a sample: the more homogeneous, the less impure it is. The most common is the Gini impurity:

\[
G(\vec{x}) = \sum_k p_k(1 - p_k),
\]

where \( p_k \) is the fraction of labels equal to \( k \) in the group.

A decision tree splits recursively the training based on the feature that gives the highest decrease in Gini impurity, as illustrated in Figure 1.4a. The final result is a series of simple boolean rules that can be interpreted by humans. Figure 1.5 is an example: the root of the tree – shown in white – contains all the training points, and each branch splits it according to a single feature. The process is repeated until no more cuts can improve the performance, or until the tree reaches a maximum depth.
1.5 Traditional Machine Learning

(a) Gini impurity as a function of the first split. The ideal cut is selected.

(b) Regions predicted by the tree.

Figure 1.4: Classifications made by a decision tree on the Iris species dataset.

Figure 1.5: The rules of a decision tree are interpretable. The saturation of the background corresponds to the purity of the node. While a human can read them, they do not make much sense to us, it is just splitting the space, one feature at a time.
1.5.7 Random Forest

A Random Forest is an ensemble of decision trees, as described in Section 1.5.6. Each tree is trained on a random subset of the data, and the final score is a vote across the trees. Furthermore, for every split, we only consider a new random subset of the features, to increase diversity. The advantage over a single decision tree is that now we have an ensemble of trees, each trained on slightly different data. Since every data point is only considered by a fraction of the trees, the random forest is more robust against noise; but for the same reason, it will not be able to model so well outliers.

1.5.8 Gaussian Processes

A Gaussian Process (gp) takes as an input a set of data points \((x, y)\) that are assumed to be generated by a latent, unknown function \(f(x)\) that we wish to infer, plus Gaussian noise. The output is a probability distribution over functions, that should be interpreted as the likelihood for each given function to have produced the observed data.

![Gaussian Processes](image1)

(a) Complete data

![Gaussian Processes, gapped](image2)

(b) Gapped data.

Figure 1.6: Reconstruction of the \(\sin(x)\) function using Gaussian Processes. Note the larger uncertainty when there is a gap in the training set.

Consider a vector space over functions, called Hilbert space \(\mathcal{H}\), and define a complete, orthonormal basis \(\{\phi_i(x)\}_{i=0}^n\). Any function \(f\) in this space can be decomposed as a linear combination of the basis:

\[
f(x) = \sum_{i=0}^{n} c_i \phi_i(x)
\]

In \(\text{gp}\), we define our Hilbert space through a positive semi-definite covariance function, \(k(x, x')\). This induces a metric defined through the distance:

\[
d(f, g) = \int k(x)k(x')g(x')dx dx',
\]

and a series of eigenfunctions of the covariance function.

For example, the previously seen radial basis covariance function defines the Hilbert space of \(C^\infty \cap L^2\) functions: all the smooth functions of square-integrable.
The GP will thus project our \( n \) data points into this \( N \)-dimensional space – in general, \( N \gg n \); usually \( N = \infty \). Note that, due to geometry, our \( n \) data points must live in an \( n \)-dimensional subspace of \( H \). The output is an estimation \( \hat{f} \) of our latent function \( f \), that can be interpreted as a decomposition in the eigenfunctions of our covariance function.

\[
\hat{f}(x) = \sum_{i=0}^{n} c_i \phi_i(x)
\]

But, unlike other procedures, \( c_i \) are not just numbers, but have probability distributions over them: Gaussian Processes gives a probabilistic view of the answer. Thus, we can use them to sample likely candidates for the latent function, as shown in Figure 1.7.

![Gaussian Process sampling](image)

Figure 1.7: Posterior samples from Gaussian Processes.

### 1.5.9 Isotonic regression

Isotonic regression minimises the squared errors of a function that is piece-wise constant, and non-decreasing. Given enough data points, it can fit arbitrarily complex curves, as long as they are monotonous. This method is particularly useful in calibration to turn predicted scores into actual probabilities, as shown in the Figure 1.8.

### 1.6 On the wrongness of machine learning

The results of machine learning can be impressive, which explains the general excitement surrounding any possible application. Their strength is that they are able to extract a signal from data without needing human guidance, but there lies their weakness: machine learning algorithms are incapable of reasoning. Since they lack a mechanistic model, they can find correlations, but not causations. That is, they can answer the what, but cannot say why.
Figure 1.8: Example of Isotonic regression applied for calibration. On the x-axis is the score given by Gaussdca, a statistical method that correlates with the probability of two residues being in contact (more information on Section 4.4). The black dots are the data points, the dotted line is a simple binning, and the solid line is the isotonic regression. The isotonic regressor does not require to define a number of bins up front which would make it susceptible to noise in too small bins.

Furthermore, they cannot extrapolate. Training a machine learning model means finding the set of parameters that “best” fit the training points, but we have no constraints outside of that region. If we were to feed it input data that are sufficiently far away from what was used to train, our outputs will be, essentially, unconstrained. For example, rare diseases are a problem for Artificial Intelligence in healthcare. Since they are rare, they do not have many training examples, and even less when several rare conditions appear on the same patient.

Moreover, input data can, on occasions, be wrong, but the machine will accept it unless it has explicitly been trained to recognise these failure modes. In general, a machine learning model can only get to be as good as the data it was used to train it, and the data it is being fed on, and will always incorporate all of its biases and limitations.

Attempts at mitigate these problems gave rise to explainable machine learning models, where the reasoning can be studied by a human. In general, these models are either limited in their expressibility, or their interpretability is hardly related to the way humans reason (see Figure 1.5). But yet, unexplainable models are often more accurate, and good enough when the cost of errors is tolerable, or there are alternative safeguards in place.

One of the last frontiers of machine learning is dealing with the situations when they are so widely used that their decisions directly affects the landscape on which they act, such as high-frequency trading or recommending systems.
We saw in the previous chapter how to turn simple linear models into more general multi-layered perceptrons via stacking. Deep learning is the logical continuation to a higher number of layers. This chapter discusses why we would want to do it and how to train them.

SUCCESS STORIES

We start this section introducing the two fields that have been spearheading the Deep Learning revolution: computer vision and speech recognition. The first consists in identifying objects in digital images; while the second is being able to transcribe and process audio recordings of human speech.

It is striking that both cases correspond to tasks that humans are very good at. Indeed, neural networks are very good at perceptual learning, or tasks where the computer replaces the human senses. This is perhaps because neural networks are inspired by the biological nervous system, but also because humans can understand the process, label more data, and, when it fails, guess at why it is wrong. Furthermore, since our brain can do it, we know they are solvable problems.

But we are not only limited to perceptual problems. Deep neural networks have shown to be good at tasks that humans cannot perform well, like contact prediction (Wang et al., 2017), estimation of protein model quality (Cheng et al., 2019), and other tasks.

2.1 THE BASIC BLOCKS

A deep learning model is composed of a series of layers, or simple transformations – usually linear – plus a point-wise non-linearity.

2.1.1 Fully connected layers

Also known as “dense” layers, they are the simplest building block of deep learning: a simple multiplication of the input \( \vec{i} \) with a matrix \( \tilde{W} \) whose entries we want to determine, as described in Section 1.5.3. To make them more general, they can include a bias term, \( \vec{b} \), that is added after the multiplication.

\[
\vec{o} = \tilde{W} \cdot \vec{i} + \vec{b}
\]

Every element of the output vector \( \vec{o} \) is a function of every input, so this layer partially destroys the structure of the data. On the other hand, since the intermediate layers can be arbitrarily large, it can have a lot of parameters, ie, it can be very expressive. It is often used as the last layer of the model.
2.1.2 Convolutions

A convolution is a linear operation over a small patch, that is repeated across the whole input, to capture spatial relationships in the data via keeping translational invariance. Consider the case of 1D convolutions. The input is a 2D matrix, where we have a number of input channels at each position. A convolution looks at a small but fixed window and applies a linear operation; then it moves one unit and repeats the same operation, with the same weights as before. The output is another 2D vector, with an arbitrary number of channels, which outputs are mutually independent. The set of weights corresponding to each output is called a filter.

Each convolution has a limited field of view, called the receptive field. The size of the input region that is visible to a layer depends on the width of the filters and number of layers before it. We can obtain a bigger receptive field using wider filters, strided convolutions – introducing gaps in the filter –, or most commonly, stacking layers.

2.1.3 Recurrent

A recurrent layer tries to capture temporal dependencies on arbitrary time steps, such as those of natural language, or repeating motifs in proteins. They work by taking two inputs: the vector at the current position ($\vec{x}_t$), and the hidden state of the previous cell ($\vec{h}_{t-1}$). A simple cell has the form:

$$
\begin{align*}
\vec{h}_t &= \sigma (\vec{W}_{hh}\vec{h}_{t-1} + \vec{W}_{xh}\vec{x}_t) \\
\vec{y}_t &= \vec{W}_{hy}\vec{h}_t,
\end{align*}
$$

where $\vec{W}$ are three matrices and $\sigma$ a non-linear function, usually the hyperbolic tangent. Note that this structure is very similar to a fully connected layer with two inputs, where the weights are shared across steps. The series of outputs at every time step $\vec{y}_t$ can be kept, to get a different prediction at every time step, or only considered the last one if we want a global one.

This layer has practical problems. In particular – as we will see in the next section – if the non-linearity is saturating and any of the intermediate steps reaches the saturating region, the gradients will vanish, preventing the flow of information. If instead, we use a non-saturating non-linearity we risk having an unstable dynamical system, where the values explode. For these reasons, no one uses the RNN as explained here, but use more complicated variations that address some of these problems. The most popular choice is the Long-Short Term Memory (LSTM) layer, that will be explained in Section 2.5.

Karpathy (2015b) has a thorough introduction to Recurrent networks, and Olah (2015) delves deeper into the convergence issues in long-range, and the solutions.
2.1 The Basic Blocks

2.1.4 Non-linearities

The workhorses of deep learning layers are linear transformations. The composition of linear transformations is also linear, so to reap the benefits of multiple layers, we need non-linearities. For the most part, they are point-wise functions that act on the output of a layer. Historically, the first non-linearity used was the sigmoid:

\[ f(x) = \frac{1}{1 + e^{-x}} \]

It has the advantage of clipping the range of values between 0 and 1, which makes the network stable with respect to large intermediate values (Figure 2.1a). An improvement was the hyperbolic tangent:

\[ f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \]

with a similar shape to the sigmoid, but now outputting values between -1 and 1 (Figure 2.1b). Being anti-symmetric, it can achieve a stable distribution of outputs with normally distributed weights in the intermediate layers.

Both functions saturate on extreme values, i.e. their derivative approaches zero. While this is useful to control the stability of the outputs, it is a problem for the propagation of gradients. The alternative is using non-saturating functions, that allow the gradients to flow for a wider range of values.

The simplest is the Rectified Linear Unit, or ReLU, introduced by Glorot et al. (2011) and plotted in Figure 2.2a:

\[ f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \]

In this thesis I have almost exclusively used the Exponential Linear Unit, or elu, shown in Figure 2.2b:

\[ f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \frac{1}{1 + e^{-x}} & \text{otherwise} \end{cases} \]

Figure 2.1: Saturating non-linearities
The choice of this non-linearity incurs in a slight increase in training time, since it needs to compute the exponential; but it usually gives a moderate improvement in performance as well.

Output

The last layer is an exception. The function applied depends on the properties of the target:

- If the output is a binary classification or multiple mutually compatible classes, use a sigmoid, as described in Section 1.5.2 on logistic regression.

- If the output is a series of mutually exclusive classes, use a softmax, as explained in Section 1.5.3 on multi-layered perceptrons.

- If the output is a regression, use a linear layer to minimise the squeezing of the outputs. This is valid even if the output is bounded between 0 and 1, when the values at the extrema are likely to appear because the network would need to learn very high values for the logits.

- If the outputs are angles, split the labels into sines and cosines and use the hyperbolic tangent. This representation preserves angular distances – i.e., angles that differ by almost \(2\pi\) are close – and are not biased towards an arbitrary middle point. The original angle can be recovered using the \text{arctan2} function.

2.2 Training procedure: gradient descent

A deep learning network can have millions of parameters, but how do we find the optimal values?

First, we need to define a loss function, that is, a measurement of how wrong our predictions are on known data. For example, the mean squared error for \(N\) data points \(x_i\) with labels \(y_i\) and predictions \(o_i\) is:

\[
L_{\text{MSE}} = \sum_{i=1}^{N} (y_i - o_i)^2
\]
Another popular choice is the cross-entropy loss, that measures the information loss between true and predicted values:

\[ L_H = \sum_{i}^{-} y_i \cdot \log(o_i) \]

We initialise the network with random numbers and measure how wrong we are. Then, we compute the gradients of the loss with respect to each of the parameters \( w_j \) of the network:

\[ g_j = \frac{1}{N} \sum_{i}^{N} \frac{\partial L (o_i, y_i)}{\partial w_j} \]

The gradients tell us in which direction we have to “nudge” the network to improve its performance for the next iteration:

\[ w_j|_{t=k+1} = w_j|_{t=k} - \eta \cdot g_j|_{t=k} \]

where \( \eta \) is the step size, a small number that reflects our belief of the size of the region where the gradients still point in the same direction.

### 2.2.1 Back-propagation

To train the network, we need to compute the gradients of the loss with respect to all the parameters of the network. This can be done automatically using the back-propagation algorithm, or backprop (Rumelhart et al., 1986), the machine version of the chain rule.

We start computing the derivative of the loss with respect to the output of the network \( \partial L/\partial o_i \). Then we can apply the chain rule to compute the derivatives with respect to the output \( y_i \) before the non-linearity:

\[ \frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial o_i} \cdot \frac{\partial o_i}{\partial y_i} \]

For example, if the last layer is a fully connected layer without a bias, the gradients for each weight are:

\[ \frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}} = \frac{\partial L}{\partial y_j} \cdot w_{ij} \]

The procedure can be applied recursively to an arbitrary number of layers using a computational graph: a directed graph where the nodes are inputs, outputs, and intermediate values; and the edges connect each node with the values that are immediately needed to compute it. Figure 2.3 contains a simple example. Nowadays, the differentiation can be done automatically using frameworks like Tensorflow (Abadi et al., 2015) or PyTorch (Paszke et al., 2017), so the deep learning practitioner can write models without needing to explicitly write down the gradients, which is time-consuming and error-prone.
Figure 2.3: An example of a computational graph computing the simple formula above. The square nodes are the three inputs, $a$, $b$, and $c$; while the diamond indicates the output value, $O$. The node in italics, $d$ is a temporary variable.

Computing the outputs of the network is called **forward pass**, since the output of each layer depends only on the previous values, closer to the inputs. To compute the gradients, we need to keep the outputs of each layer between the weight and the labels. Since we transverse the graph in the opposite direction, it is called the **backward pass**.

### 2.2.2 Stochastic gradient descent

Computing the backward pass takes roughly 3 times longer than the forward pass (Vanhoucke, 2016). Since deep learning models require large amounts of data, computing the gradients on the whole dataset is impractical. Instead, we can approximate the gradients of the loss w.r.t. the whole data by a random sample:

$$g_j = \frac{1}{N} \sum_i \partial L(o_i, y_i) \approx \frac{1}{n} \sum_i \partial L(o_i, y_i)$$

where $n \ll N$. Each subsample is called a **mini-batch**, and it is re-sampled after every iteration. Once we have sampled the whole data, we have completed an **epoch**.

The stochasticity of the gradient descent introduces noise in the gradients. While this may slow down convergence, it also helps the network explore the parameter space, allowing it to break from possible local minima.

### 2.2.3 Optimisers

The optimisers are the algorithms responsible for transforming the gradient into weights updates. The simplest form is the already introduced Stochastic Gradient Descent (**sgd**), where the weight updates are proportional to the gradient computed at the previous iteration:

$$\Delta w_j |_{t=k+1} = -\eta g_j |_{t=k} = -\eta \frac{\partial L}{\partial w_j} |_{t=k}$$

This is the basis for all optimisers used in Deep Learning.

To reduce the noise introduced by the stochasticity we can introduce a momentum term, regulated by the parameter $\mu$:
The “velocity” vector $v_j$ keeps a memory of the latest updates. The “mass” $\mu$ controls the balance between the directions pointed at by new updates and keeping the inertia.

The momentum term may be pointing in the wrong direction, for example, because we have overshot the minimum. The Nesterov momentum update (Nesterov, 1983) performs the momentum update before computing the gradients, giving the gradient the possibility of correcting mistakes if the momentum was misleading. If the momentum term was instead a good estimation, the new gradient will be computed closer to the minimum, yielding a better estimation.

In similarity to the temperature in Simulated Annealing (see, for example, the work of Tsallis and Stariolo (1996)), we can reduce the learning rate as the training progresses. A common choice is to reproduce cooling down the system as reducing the learning rate. As in thermodynamics, a smaller temperature means the system has to move more slowly, giving smaller steps, and exploring the space closer to the current position. On the $k$–th epoch, the learning rate is:

$$\eta = \frac{\eta_0}{1 + \delta \cdot k} \quad (2.1)$$

Another option is to not set the schedule upfront, but adapt it to the specific training. i.e. reduce it by a fixed fraction when the loss has not improved for a certain number of epochs. We believe the system to be close to the minimum, and we need to take smaller steps to approach it.

Instead of setting a learning rate schedule from the beginning, adaptive optimisers try to adjust it to the data. The general idea is that the closer we are to the minimum in each dimension, the smaller the learning rate should be. AdaGrad (Adaptive Gradients (Duchi et al., 2011)) uses the $L^2$ norm of the gradients to regulate the learning rate. We drop the index subscripts, as every term corresponds to one dimension:

$$a|_{k+1} = a|_k + s^2|_{\mu=k}$$
$$\Delta w|_{\mu=k+1} = -\eta \frac{s}{\sqrt{a + \epsilon}} = -\eta \frac{s}{\sqrt{\sum_{\tau=0}^{k} s_{\tau}^2 + \epsilon}}$$

where $\epsilon$ is a small number to prevent division by 0. The effective learning rate of AdaGrad is monotonously decreasing, and it decays faster in the directions where the gradients are higher. This makes it sensitive to initial conditions: if any dimension is given a bad initial condition, the gradients will be large, and the effective learning rate small, so AdaGrad will be limited to only small steps in that direction. adadelta (Zeiler, 2012) solves this by accumulating only the updates over a window with an exponential decay:
allowing the algorithm to overcome past large gradients. It also considers the dimensionality of the quantities: in AdaGrad the learning rate has dimensions. They replaced it for a value of the right dimensionality:

\[ \Delta w|_{t=k+1} = -\frac{\sqrt{\Delta w|_k^2}}{\sqrt{a + \epsilon}} g \]

ADADELTA depends only on one parameter, \( \rho \), and it is less sensitive to starting points.

Adaptive Moment Estimation (Adam) (Kingma and Ba, 2015) is the most modern optimiser, and it combines several of the ideas of previous algorithms. In the first place, it keeps track of the two moments of the distribution of the gradient on a sliding window with decay rates \( \beta_1 \) and \( \beta_2 \):

\[
\begin{align*}
    m|_{k+1} &= \beta_1 m|_k + (1 - \beta_1) g|_{k+1} \\
    v|_{k+1} &= \beta_1 v|_k + (1 - \beta_1) g^2|_{k+1}
\end{align*}
\]

Since they are initialised with zeroes, they are biased, especially at the beginning. We can correct them:

\[
\begin{align*}
    \hat{m}|_{k+1} &= \frac{m|_{k+1}}{1 - \beta_1^{k+1}} \\
    \hat{v}|_{k+1} &= \frac{v|_{k+1}}{1 - \beta_2^{k+1}}
\end{align*}
\]

The final update rule is:

\[ \Delta w|_{k+1} = -\eta \frac{\hat{m}|_{k+1}}{\sqrt{\hat{v}|_{k+1}} + \epsilon} \]

This is the optimiser that will be used in the rest of the thesis. For a thorough review consult the work of Ruder (2016).

2.3 TAMING THE COMPLEXITY: REGULARISATION

Neural networks can have millions of parameters, so they are susceptible to over-fitting. In order to converge to generalisable models, we can apply a variety of regularisation techniques. Most of them act as a barrier that hinders the training, in a way that only enough data can overcome. Others encode invariances of the real world into our model, such as translation invariance given by convolutions on images. Here are some:

Weight decay

To prevent any single activation from dominating the predictions, we can add a term to the loss penalising large weights. If this term is proportional to the square of the weights, it is called L\(^2\) regularisation and can be interpreted
as a Gaussian prior over the weights centred around 0, as seen with linear models in Section 1.5.1.

The most popular technique specifically developed for deep learning is Dropout, (Srivastava et al., 2014). During training, a random fraction $0 < \rho < 1$ of intermediate inputs is set to 0 — dropped out — while the rest of values are scaled by a factor of $\frac{1}{1-\rho}$ to compensate. Since the network cannot trust any particular neuron activation to be present, it must distribute the information across different parts. The network is effectively different for every batch, so we are training an ensemble of models, most of which have not seen any data, but are heavily regularised to the average. At prediction time, dropout is usually turned off, but if we leave it on and sample with the same input, we can estimate the uncertainty of the prediction like in Gaussian Processes.

The next breakthrough in regularisation is Batch Normalisation, or Batchnorm (Ioffe and Szegedy, 2015). This has the unusual properties of both increasing generalisation capabilities — thus serving as a regulariser — and helping training performance — ergo, without hindering.

It works as a layer that learns the mean and standard deviation of the distribution of its inputs and corrects them to keep them close to 0 and 1, respectively. As was seen with some optimisers, it can keep a momentum term to smooth out the changes in averages. In the original paper, it was placed before the activation, but further research suggests it is better to place it after it.

The reason why Batch Normalisation actually works as a regulariser was not well understood until the work of Santurkar et al. (2018), who showed how it smooths the loss function landscape, both helping the training and converging to more generalised minima. Keskar et al. (2016) had shown that smoother landscapes lead to better generalisation.

On a higher level, we can design our network to be robust with respect to invariances in our data. For example, convolutions are translational invariant, so a convolution-based secondary structure predictor will be able to recognise the sequence signature of an α-helix regardless of its relative position. We can also design our training and architecture to fit the problem, as we did on Paper IV.

Sometimes, we wish our network to learn an invariance that cannot be embedded in the network architecture or the representation, such as scales of natural images in computer vision, or multiple sequence alignment quality in protein bioinformatics Since deep learning models can leverage arbitrarily large datasets, we can use data augmentation, a procedure where data points at training time are modified in the way we want our network to learn invariance. So, in computer vision by showing the image of the same cat at multiple sizes and orientations, we hope our network can recognise a cat regardless of the two transformations. In bioinformatics, we may want to train on alignments generated with different settings to provide more robust networks.

Special mention deserves the concept of Differential Privacy, or how to learn from sensitive data without leaking private information (Dwork et al., 2006)

Dropout

Batchnorm

It was previously thought that Batchnorm was preventing the weights to covariate together, by forcing $\sigma \approx 1$.

Architectural

Data augmentation
**Differential Privacy**

This framework allows users to train models with the confidence that an adversary with full access to the trained model cannot recover any single identifiable information because the model is guaranteed not to memorise any of its training data.

One example procedure implemented by McMahan and Andrew (2018) starts by splitting each mini-batch used by the Stochastic Gradient Descent into micro-batches and computing the gradient for each of them. Then, the gradients are clipped, so that any micro-batch can provide only so much change in the weights, and independent random Gaussian noise is added to each of them, to dilute any private signal they may have. At the end of the iteration, the mini-batch is averaged, and the weights updated.

### 2.4 Conjunctive Tissue: Tensors and Gradients

As we have seen before, each layer is composed of mostly tensors and linear operations. Every intermediate layer in a deep network is learning a transformation of its inputs, that we can interpret as a representation that brings it closer to the target. These representations are, in general, difficult for humans to interpret, but in some cases, like computer vision, we can *visualise* them. A convolutional network trained to identify objects in images will learn to recognise edges and simple textures in the first layers, that are combined into more complex representations as the information travels through the network.

One important consequence is that all data, regardless of its source or nature, is represented as tensors. We can then seamlessly connect through these tensors different branches working with dissimilar kinds of data, and train end-to-end having them talk to each other through gradients. Karpathy (2015a) has a pedagogical explanation on this idea applied to automatic description of images in natural images. A more extreme example is the work of Kaiser et al. (2017), where they combine eight different tasks into a single network, including translation, captioning, and image recognition.

In our case in bioinformatics, this means that we can seamlessly combine multiple sources of information about a protein into the same network, be it sequence, structure, phylogenetic, or even the text of scientific articles.

### 2.5 The Quest for Depth

Deeper networks have advantages over shallower ones, even for the same number of parameters. For example, stacking convolutions with a few filters each means the final layer has a bigger receptive field than a single convolution with many filters. Furthermore, most layers are “stupid” linear transformations with non-linearities between them. Stacking more layers means we obtain a more non-linear function, and hence, richer. The main practical obstacle to train deeper networks is to be able to transmit the gradient information down to every layer. Indeed, in the first attempts with multi-layered perceptrons, more than two hidden layers did not offer an advantage. Glorot and Bengio
(2010) studied in detail the difficulties of training multi-layered perceptrons known at the time.

Their first contribution was non-saturating non-linearities, like ReLU. Since they do not saturate, neurons are never stuck in ranges where the gradients are almost zero, and thus prevented from learning. ReLU has a null gradient for negative numbers, but we expect that to happen only half of the examples, while half of the others will give a positive signal. This is not guaranteed, and there is a chance of “dead neurons”, where the activation is always zero, but that has not been found to be a significant problem. If dead neurons are a problem, LeakyReLU can be the solution. This activation is like ReLU, but the negative side has a small, fixed slope, so the gradient is non-zero everywhere.

Tightly tied with the non-linearity is the initialisation of the weights. Glorot et al. recommended initialising the layers with zero mean and standard deviation $\sigma = 6/[n_{in} + n_{out}]$, where $n_{in}$ and $n_{out}$ are, respectively, the number of input and output features of the layer. In this case, small initialisations are critical because saturating neurons are very slow to train. With non-saturating initialisations like ReLU, we can use larger weights without fearing the plateaus. He et al. (2015) recommends $2/[\sqrt{n_{in}}]$ as a stable choice: each output will be a linear combination of $n_{in}$ input features. Since the standard deviation of the sum of $n$ normal random variables is proportional to $\sqrt{n}$, and half the activations are zero, this factor keeps the distribution of activation stable across layers.

Adapting the initialisation to the non-linearity uses the expected dynamics of a random network to keep stable distributions. Batch normalisation forces them, ensuring that the information can flow regardless of the dynamical state of the network, and adapting to the particular features of the dataset.

If the first layers in a deep model are too far from the labels, can we bring them closer, without sacrificing depth? Szegedy et al. (2015) attempted that in GoogleLeNet, where the network had several output layers at different depths. While only the last one was actually used in production, the intermediate ones inject gradients and help train the lower layers.

The last fundamental idea is Residual Networks (He et al., 2016), which allows to train networks of arbitrary depth. ResNets are composed of blocks of two layers – convolutions in the original paper, but applicable to any other type – where the output is directly added to the inputs using a so-called skip connection, as illustrated in Figure 2.4. This extra path opens an information highway from the inputs to the labels.

Recurrent networks, albeit formally “shallow” – they typically have one or two layers – face similar problems to feed-forward networks, now because the information has to travel along the time dimension. Long-Short Term Memory (LSTM) layers are a variation of the simple RNN that includes some internal skip connections, allowing the network to transfer long-range information, but also “forget” gates that can learn to ignore certain parts of the input that may be irrelevant.
Pascanu et al. (2012) explored the influence and solutions of initialisations and non-linearities to the problems of vanishing and exploding gradients, and provide recipes for training recurrent networks minimising these problems.

Practical advice

2.6 DEEP TRANSFER LEARNING

The success of machine learning is tightly coupled to the quality and quantity of both the training data and the labels. The concept of transfer learning can help us mitigate to some extent these limitations by leveraging related datasets and extracting knowledge. It comes in two flavours: multi-task, and representation learning.
2.6.1 Multiple tasks

Some tasks are related to each other, so a model that learns them simultaneously will perform better than when done independently. Figure 2.5 shows the Ramachandran plot, $\phi$ and $\psi$ dihedral angles, for a set of proteins. The points are clustered in distinct regions, suggesting that knowledge of one of the angles also conditions the other. An algorithm that learns them together has a better chance of recovering this relationship than two distinct models without information from each other. Furthermore, the dihedral angles are conditioned by secondary structure, so for example, knowing a residue is in an $\alpha$-helix means the $\psi$ angle is close to zero. Lastly, we can observe that the surface accessibility is not completely conditioned by the dihedral angles, but they are not independent. Indeed, $\alpha$-helices are more likely to be exposed than $\beta$-sheets, but rarely more than 60%.

The ways multi-task learning can help our predictions are:

- Implicitly learn of the joint distributions of dependent outputs.
- Some tasks are difficult. Having labels for an easier task can help bring the network to a state where it is easier to predict the harder one.
- Even if the tasks are equally hard, combining gradients from several tasks can help reduce noise and improve convergence.
- Labels from different annotations mean we have more data, which means we can train a larger and more expressive model.

This was the idea behind the secondary structure predictor used in Papers III and IV.

2.6.2 Representation learning

As explained in Section 2.4, each intermediate layer in a deep network is learning a representation of its inputs, a transformation that brings it closer to the target. If the dataset for our task of interest is small or biased, our network will not be able to learn a good representation, ergo the performance will be poor. In this situation, we can apply transfer learning, which is done in two steps:

1. Pre-training: train a network on a related task for which we have a high-quality dataset. This step learns a new representation of the data.

2. Training: replace the top layers with new ones, and retrain on the dataset and task of interest while keeping the weights of the re-used layers unchanged. We keep our previous representation but use it for a new task. This step is training fewer parameters, so it needs less data, and the architecture can be more constrained.

3. Fine-tuning (optional): we can refine the predictions by allowing more layers to change. In this stage, we unfreeze some of the pre-trained layers, and train again. Our network is already trained, so we only want to take small steps.
This can be done keeping the learning rate small. Otherwise, we will ruin the pre-training.

Paper IV, makes use of this technique, where we pre-trained a section of the network on a large subset of the PDB to predict secondary structure from sequence. These predictions were then used by a second network, combined with the secondary structure of models, to predict their quality.
Part II

THE BIOLOGY
3.1 WHAT ARE PROTEINS AND WHY DO WE CARE?

Proteins are the fundamental machines in biology, performing tasks such as catalysis, transporting molecules, and providing the structural backbone of the cell. They have crucial roles in the biochemical pathways, so understanding them can lead us to create new and refined drugs or bio-engineered organisms.

But, how are they created? The information used by the cell to build proteins is encoded in the DNA. When the biosynthesis begins, the double helix unfolds, and the genetic contact is translated into RNA. This new molecule is then transported to the ribosome, the machinery that translates the RNA into functional proteins.

The DNA is in itself a polymer of four blocks: the bases A, T, G, and C, grouped in non-overlapping triplets called codons. A protein is encoded between a start (usually ATG) and a stop (TAG, TGA, and TAA) codons, while each of the codons in between codifies for one amino acid.

3.2 THE MANY LEVELS OF PROTEIN STRUCTURES

Proteins are polymers, long chains of amino acids, structured at several levels, depending on which scale we look at them. In this section, I will present the main descriptions of proteins at different scopes.

3.2.1 The amino acids

Amino acids are the building blocks of proteins. They are composed of a carboxyl (-COOH) and an amine (-NH$_2$) groups, forming the backbone, and a side-chain. Two examples are illustrated in Figure 3.1. DNA usually codifies up to twenty different species, but at least 500 are known to occur naturally (Wagner and Musso, 1983). The differences between most of them are only on the side chain.

The side chains are the group responsible for the specific physico-chemical properties of the compound, such as hydrophobicity, pH, or electrostatic charge. The backbone can polymerise, bonding with other amino acids and forming a long chain. Having a common backbone means that in principle, any amino acid can be connected to any other, which gives proteins a lot of flexibility: for a protein of length $L$ there are $20^L$ possible combinations.
Figure 3.1: Two amino acids as appear in proteins. The maroon, solid rectangle indicates the backbone, common to all amino acids; and the blue dashed the side chain, that determines the specific chemical properties.

Figure 3.2: The peptide bond, shown in blue. The different shades correspond to different elements.
3.2.2 Primary structure or sequence

The amino acids can connect to each other through the peptide bond, forming a chain. The primary structure is the sequence of amino acids as encoded in the DNA. For example:

\[
\text{ALA ARG ILE ASN GLY ARG GLU ILE ASN VAL THR LYS LYS}
\]

This is the easiest to obtain experimentally since the advent of sequencing techniques. In their modern rendition, these experiments read the DNA of the organism, that can be translated into proteins. The collection of all protein sequences of an organism is the proteome.

3.2.3 Secondary structure

The polypeptide chains are locally organised in motifs stabilised by hydrogen bonds. The most common is the $\alpha$-helix, shown in Figure 3.3a, where the $\alpha$ hydrogen bonds are formed between the backbone oxygen of one residue, and the hydrogen of the amine group, four residues beyond, and continued in a regular pattern, forcing the backbone to twist into a helix. The same bond is possible with residues that are closer – two or three – or further – five residues, the $\pi$ helix – but are less common.

The second most common arrangement is the $\beta$-sheet depicted in Figure 3.3b, where approximately extended chains are placed next to each other and bonds are formed between juxtaposed residues.

3.2.4 Tertiary structure

Once the chain is locally stabilised by the hydrogen bonds it folds into a compact structure. The spatial arrangement of the secondary structure elements is the tertiary structure. One example is the chain A of the protein 4V0B, in Figure 3.4.

3.2.5 Quaternary structure

Proteins do not always work alone, but they form complexes composed of several chains. The relative arrangement of each chain is the quaternary structure.

3.3 Protein biochemistry

Experiments by Levinthal (1969) show that proteins placed in the right medium of pH and temperature will unfold into a random coil, with no trace of tertiary nor secondary structure. When restored to physiological conditions, they will refold into its natural shape, recovering its original biological and chemical properties. As long as the protein is chemically unaltered, it will recover the tertiary structure independently of the folding machinery of
Figure 3.3: The two most frequent secondary structure elements.
the cell. This was codified as Anfinsen’s dogma: under physiological conditions, the primary structure of globular proteins determines the secondary and tertiary structures, and thus, its function, independently of the cell’s machinery (Anfinsen, 1973). Typically, the fully folded state is reached in the order of milliseconds to seconds. The hypothesis has a few exceptions, namely: fibrous proteins, prions – which are proteins that have an alternative but non-functional stable tertiary structure – aggregating proteins – where the individual proteins bind to each other forming large and non-functional structures – and large, multi-domain proteins. A reasonable hypothesis is that a priori, the native conformation seems to be the state in the lowest energy, and the random coil is just rolling down the energy landscape.

Cyrus Levinthal noted that, for every amino acid, a protein has two main degrees of freedom corresponding to the torsion of the backbone, plus one more for the rotation of the side chain. This gives us a configuration space of the order of $10^{2L}$ for a protein of $L$ residues, but the kinematics suggest that it only has time to sample $10^8$ conformations, an exponentially tiny fraction for proteins of typical length.

Levinthal suggested two corrections:

- The native conformation is not necessarily the one of minimum energy, but a local meta-stable minimum with a deep enough well that it is stable. For most proteins, this is not very deep, around 10-15 kcal/mol.
The native conformation must be kinematically accessible, possibly guided by local interactions that partially fold the protein. In other words, the energy landscape of protein folding presents a wide funnel from the unfolded state to its near-native conformation, which allows for a rapid transition.

3.3.1 The hydrophobic effect

Water molecules are electric dipoles with a partial negative charge on the oxygen, and a partial positive charge on the hydrogens. This introduces an electrostatic interaction between molecules in bulk of water, that self-organises to form a network of hydrogen bonds. Any non-polar molecule submerged in water will disrupt this network of bonds. To preserve as many energy-favourable hydrogen bonds the water molecules near the boundary can reorient themselves, forming a pocket around the hydrophobic substance in the process. The degrees of freedom for the boundary is restricted, so the entropy – and consequently the free energy – is reduced. This is the hydrophobic effect, responsible for the immiscibility of fats and water.

A protein chain will be composed of both hydrophobic and hydrophilic amino acids. When placed, unfolded, in the cellular environment, the hydrophobic ones will not be able to form hydrogen bonds with the water. The water molecules will then reorient themselves to form hydrogen bonds with their neighbours, which creates tiny pockets around them. Quickly, these pockets will join together, expelling most of the water and forming a compact core of hydrophobic residues, surrounded by hydrophilic ones, the molten globule. A partial secondary structure, close to native, is observed.
3.3.2 From molten globule to structure

The core of the molten globule is not completely devoid of solvent, which slightly increases the distance between adjacent elements, greatly reducing the van der Waals interactions. Long-range contacts are not preserved, which indicates a relative flexibility between its components. Our globule is “soft” and “wet”.

The transition to the solid, native state, involves the repacking of the side chains into a tight and compact core. This process finalises the packing of secondary structure and freezes most of the long-range contacts.

It is noteworthy that not all proteins present a molten globule, and can transition directly between coil and native (Finkelstein and Ptitsyn, 2002).

Further discussion

3.3.3 Energy terms

A relatively generic form of the energy dictating the dynamics of a protein is:

\[
H(\{\vec{r}_i\}_{i=0}^N) = \sum_{\text{bonds}} k_b (d - d_0)^2 + \sum_{\text{angles}} k_\alpha (\theta - \theta_0) +
\]

\[
+ \sum_{\text{torsions}} f(\omega) + \sum_{\text{free pairs}} k_{ij} \left[ \left( \frac{r_{0ij}}{r_{ij}} \right)^{12} - 2 \left( \frac{r_{0ij}}{r_{ij}} \right)^6 \right] +
\]

\[
+ \sum_{ij} \frac{q_i q_j}{4\pi\epsilon r_{ij}}
\]

The terms correspond to:

\(\alpha\) A harmonic potential on the bond lengths \(d\) for every pair of covalently bonded atoms, where \(d_0\) is the ideal bond length and \(k_b\) is the strength of the potential for the atom types.

\(\beta\) A harmonic potential on the angles between adjacent bonds \(\theta\).

\(\gamma\) An arbitrary function \(f\) on every torsion angle \(\omega\).

\(\delta\) The Lennard-Jones potential between all pairs of atoms not covalent bonded, where \(r_{0ij}\) denotes the equilibrium distance and \(k_{ij}\) the depth of the energy well. This term corresponds to the van der Waals forces.

\(\epsilon\) The electrostatic energy between charges, assuming they are spherically distributed.

Note that the collection \(\vec{r}_i\) includes the water molecules and other atoms in the environment of the protein itself.

Of these terms, \(\alpha\) and \(\beta\) are strictly valid only near their respective minima, while \(\delta\) is valid for distances around and beyond the minimum. In practice, this is not a significant problem because the deviations from the ideal values are small. From here we can see that most of the dynamics in proteins is given...
by the flexibility of the torsion angles. Roughly speaking the torsions of the backbone determine the secondary structure and those of the side chains, the packing.

The final thermodynamical component is the entropy, which measures the number of accessible states. An important contribution comes from the interaction between water and hydrophobic areas. In the presence of a hydrophobic surface, water has to reorient to preserve the hydrogen bonds – they are very energetically favourable, around 5 kcal/mol, so there is a high cost to its breakage. This puts a constraint to their orientation, partially freezing them around the surface, which reduces their entropy.

### 3.4 Experimental Methods

How can we know the structure of a protein? The most commonly used methods are X-ray crystallography and NMR spectroscopy.

**X-ray**

X-ray crystallography starts by purifying and crystallising the protein, and observing the diffraction pattern of an X-ray beam. The X-rays interact with the protein, diffract, and form a pattern of dots that is proportional to the intensity of the Fourier transform of the electron density. By taking the inverse transformation, the crystallographers can reconstruct the electron density, and then fit an atomic model. The X-ray detector – a photographic plate or a CCD sensor – can only measure intensities, not phase information. Experimentalists have to apply an iterative process where they combine estimated phases from models with the real intensities.

One criticism of X-ray crystallography is that the proteins are in crystals, not in physiological conditions. The alternative method is Nuclear Magnetic Resonance (NMR) (Wüthrich, 2001), which applies strong oscillating magnetic fields and measures resonances. From the spectrum, one can identify distances between atoms with non-zero nuclear spin. The method cannot provide sufficient constraints to determine the whole system, so it is given as an ensemble of models instead. The differences between individual structures can be caused by natural movements of the protein, or by uncertainties in the experiment itself.

In the last decades, a new method has become practical for protein structures: cryogenic electron microscopy. In this method, the protein samples are placed in solution on a thin sheet of solution, flash-frozen to cryogenic temperatures, and put under an electron microscope. Since proteins are diluted, each one is frozen in a random orientation, so each image is a 2D projection along a different axis. The various images are clustered and combined to reduce noise, and a 3D model is inferred from it.

This technique allows us to study large complexes in solution, without the necessity to crystallise them. It can reveal the presence of unknown proteins, and even provide information on the thermodynamics: The Boltzmann distribution says that the probability of a state \( x \) depends on the energy of the system:

\[
p(x) \propto e^{-\frac{H(x)}{k_B T}}
\]
So, when we have a sample with different conformations of the same protein, counting the number of images that correspond to each state gives us a measure of the difference of energies between the two states. The development of Cryo-EM was recognised in 2017 with the award of the Nobel Prize in Chemistry “for developing cryo-electron microscopy for the high-resolution structure determination of biomolecules in solution” to Jacques Dubochet, Joachim Frank, and Richard Henderson (Cressey and Callaway, 2017).

3.4.1 Partial restraints

Sometimes, an experiment to obtain the full structure is not possible. There are simplified methods that, while they cannot give a full structure, they may be able to provide enough constraints for computational methods.

The first is Small Angle X-ray Scattering, or saxes. The process is similar to X-ray crystallography, but now the protein is in solution. Each unit will produce its own interference pattern, but since they are not crystallised, the orientation of each pattern is randomised. From this we can infer a rough shape of the protein, and the presence or absence of differences between conformations.

A related method is Small Angle Neutron Scattering, or sans. It replaces X-rays with neutrons, which changes the sensitivity to different nuclear species. In particular, hydrogen has one of the largest cross-sections. One application is the study of membrane proteins in an appropriately chosen detergent, which can be almost invisible to the neutron beam. More information on both methods can be found in the work of Feigin and Svergun (1987).

Mass-spectrometry cross-linking (Sinz, 2018) can be used to measure distances within proteins. The protein is mixed with a reagent that connects to specific functional groups of amino acid side chains. Then, an enzyme is added that cleaves the protein, and the fragments are sent to a liquid chromatography tandem mass spectrometer that analyses the composition of each fragment. The linkers keep the fragments split by the digestion together, from which we can infer they were in contact. Furthermore, since we know the size of the linker, we can have an accurate upper bound on the distance.
Bioinformatics is the application of computational methods to biological problems, either by analysing large amounts of data or by replacing experiments with computer programs.

4.1 Predictors

A significant fraction of the protein bioinformatics work is the development of predictors: statistical or machine learning methods that can, given limited information, predict properties of proteins. A common choice is to make predictions taking only the amino acid sequence because this is the easiest information we can acquire of a protein. Other methods may require extra annotation, such as Gene Ontology – an annotation condensing all our knowledge of the gene that produced the protein – or 3D structures – experimental or models.

4.2 Increasing Statistics: Multiple Sequence Alignments

Proteins, just like organisms, are products of evolution that accumulate random modifications over time. As mutations accumulate, species diverge, but they retain certain similarity due to their common origin, thus, we can find similar versions of the same protein in different organisms. Even when the function is novel, it usually evolves as a minor modification of pre-existing proteins. This means that, in most cases, we can obtain a better prediction when working at the family level, i.e., considering not only our protein of interest but also any related sequence in our database.

To have coherent statistics, every sequence must be aligned to the original query, as in the following example:

```
   1 10 20 30 40 50
FCLEPPYTGPCKARMRVYFYNARSGCETFIYGGCKAKRNNFKSEECMRTCG
FCREPPYTGCASHVFYFYNATTGLQSFVYGGCGKQNNFMDEKECLHTCD
FCREPPYTGCRAHFIYFYNATTGLQOTFYVGGCGKQNNFMDEKECLHTCD
ICSMKNTGPKCAYMPRFNSQTKQCEFEEIFIYGGCGNNNNFMTRDCCNCS
-CTLPKVPGCPNAYFVRWYDQOEKEICISSFIYGGCGNNNNFMQSEVCD-----
------DPGPCKAYMPRFYEIEEIKKCEQFIEYGGCGNENRFFTKRECQICK
ECLMAPODPGNGCVERWFFSPEAEKQRLFYGSGQCNANAENFJEDCMASK-LCHLAMESGCPRAAKPRWYFDPGOKTCVEEYIYGGCRCGSSNNSFETAKAECMRTCS
-CEQANDPGPCXAEPRWYFNSQTGCEQFIEYGGCLGNDNFFVTEQEQODTTG
```

The collection of sequences is called a Multiple Sequence Alignment, or msa. Here, the dash - indicates a gap, a residue that is not present at that particular position. This section explains how to build it.
4.2.1 **Pairwise alignment**

The building block of an **MSA** is the alignment of each related sequence to our query, and the first step is to measure how similar they are. The most basic scheme simply counts the number of matching amino acids, assigning a positive score for matches, and a penalty for mismatches.

Proteins have insertions and deletions, which means the matching regions in different proteins may be discontinuous. We can allow for gaps in the scores including a term that considers both opening – our belief of how common insertions and deletions are – and extending the gap – considering our expectation of their length.

Furthermore, not all mismatches are the same. For example, leucine (L) and isoleucine (I) are functionally very similar, and can often be replaced without loss of function. On the other hand, a single mutation from glutamic acid (E) to valine (V), in humans is enough to cause sickle cell anaemia.

Neither are all matches alike. Alanine (A) forms over 7% of the proteome, so we are more likely to find a hit at random than with tryptophan (W), which forms less than 2%.

We can codify the difference between different amino acids in a substitution matrix. The most common choice to detect distantly related proteins is the **BLOSUM** family. Henikoff and Henikoff (1992) derived them from a set of 500 multiple sequence alignments.

4.2.2 **Scaling up: BLAST**

Performing a pairwise alignment on a large sequence database would be very slow, but for a given query, most entries in the database are not related. Programs like **BLAST** (Basic Local Alignment Search Tool, Altschul et al., 1990) introduce a set of heuristics to discard negatives as quickly as possible. They do not guarantee an optimal alignment according to the scoring matrix, but they compensate it by being able to produce a much higher amount of data.

4.2.3 **Position Specific Scoring Matrices and psi-BLAST**

Substitution matrices assume a fixed evolution model, and that every residue is as important as every other. But this is not true, we can observe that some protein regions are highly conserved because they are, in one way or another, important for the function of the protein. In the earlier example, we can see how around column 40, there are fewer changes. We want our search to reflect this fact, and, for instance, only accept changes in these regions if they provide enough similarity in the rest of the protein to support the fact that they are related.

**PSI-BLAST** (Altschul, 1997) does this using an iterated search, and building **Position Specific Scoring Matrices (PSSM)**. The first iteration is a normal **BLAST** search, using the default substitution matrix, but in the subsequent iterations, it uses the statistics of each column to update the substitution matrix. Since
each position is treated independently, it can model better the importance of each residue.

### 4.2.4 Increasing sensitivity with Hidden Markov Models

**PSSMs** can model the differences in probabilities for each amino acid at different positions, but gaps are still modelled with a flat penalty across the sequence. We know this is not a good representation of proteins, since they are unlikely to happen in the core due to the lack of space. Insertions are easy to introduce in short, external loops; and deletions are most likely in long loops. A representation that allows us to model both mutations and insertions and deletions with different rates at different positions are Hidden Markov Models (**HMM**).

An **HMM** is a probabilistic model, where the system travels across a series of hidden states following a Markov chain, a model with discrete states where the probability of transition depends only on the current state – ignoring the history. We do not have access to these states – hence hidden. Instead, each of them has a certain probability of emitting a symbol: the observed amino acids. Fitting a Hidden Markov Model consists on estimating the emission probability for each state – the analogue to the **PSSMs** – and also the transition probability between states. Like in **PSI-BLAST**, this is done iteratively, passing several times through the data and updating our belief. Since **HMM** is a fully probabilistic model, we can apply heuristics with more rigorous underpinnings, which also helps to increase sensitivity while keeping the number of **CPU** cycles low.

The two most popular programs to perform these searches are **JACKHMER** (Johnson et al., 2010), and **HHBLITS** (Remmert et al., 2011):

- **JACKHMER** compares every sequence in the database with its **HMM**. It performs an **HMM-sequence comparison**.

- **HHBLITS** uses a database of pre-computed **HMMs** based on sequence clusters. It compares **HMM** to **HMM**, and in case of a match, includes all the members of the cluster in the **MSA**.

**JACKHMER** is slower because it must consider each sequence; but it can search against any database without the need to pre-prepare it. **JACKHMER** can easily keep an up-to-date sequence database, or work on databases tailored for the search at hand.

On the other hand, each **HHBLITS** search is faster since it only needs to compare once against each cluster, but it requires a lot of computational resources to perform the initial clustering. This means the databases are not updated as often, and only a handful of versions are available. Furthermore, since all the members of each cluster are added, the **MSA** has sharp boundaries at the edges of each one.
One of the main goals of protein bioinformatics is predicting the 3D structure of a protein given only its sequence. In this section, I will present the main strategies.

4.3.1 *Physics-based modelling*

Unfolded natural proteins in physiological conditions are known to fold into their native state (Levinthal, 1969). In principle, the same procedure can be replicated *in silico* with molecular dynamics, like the work done by Freddolino et al. (2008). This was possible because the domain is very short – 35 residues – and fast folding; but in general is not a practical solution, as it requires a lot of computational work.

4.3.2 *Homology modelling*

Proteins that are close in sequence are close in structure, so if we can find a protein of known structure that has a sequence close to our protein of interest, we can use it as a *template*. When a good hit is found, this is the most accurate and reliable modelling strategy.

*modeller* (Šali and Blundell, 1993), for example, uses the templates to infer distance restraints, uses geometrical algorithms to create an initial model, and then relaxes it with a short molecular dynamics run.

4.3.3 *Ab-initio folding*

In the absence of available templates, we can use *ab-initio* or *template-free* methods, in which we try to find the minimum energy of the system, including additional terms derived from predictors. The two most widely-used programs are Rosetta and *confold*.

*Rosetta*  Rosetta’s (Leaver-Fay et al., 2011) *ab initio* protocol implements a simulated annealing scheme that starts with an extended chain and randomly replaces fragments with ones derived from the pdb. Since the fragments are taken from real proteins, the models tend to have good chemical properties, even when completely wrong. In order to speed up and improve convergence, we can add additional energy terms to the force field, such as contact restraints.

*CONFOLD*  *CONFOLD* (Adhikari et al., 2015) is the other protocol used in this thesis. It is build upon *cns*, and works by transforming contacts and secondary structure to a large set of distance restraints between atoms. Then, it runs a geometric solver, followed by a fast relaxation. This method is much faster than Rosetta, but since it lacks the chemical information of the fragments, it does not work well on regions without contacts.
4.3.4 Evaluation

Once we have a model, how can we compare it with the native structure? How can we measure how accurate it is?

One option is to superimpose model in such a way that the metric is optimised. Given two structures with \( L \) residues in common, and being \( d_i \) the distance between the \( i \)-th residues of both structures, we define the distances as:

\[
\text{RMSD}, \text{ or the Root Mean Squared Deviation:}
\]

\[
\text{RMSD} = \min \left( \sqrt{\frac{\sum_{i=1}^{L} d_i^2}{L}} \right),
\]

The S-score (Superposition score) is:

\[
S = \max \left( \frac{1}{L} \sum_{i=1}^{L} \frac{1}{1 + \left( \frac{d_i}{d_0} \right)^2} \right),
\]

where \( d_0 \) is a parameter to be decided, usually 3 Å.

The \( \text{tm} \) (Template Modelling score) is the same as \( S \), but \( d_0 = 1.24 \sqrt{L - 15} - 1.8 \), to keep TM roughly independent of the length.

Both \( S \) and \( \text{tm} \) scores give values between 0 and 1. For TM, values below 0.3 correspond to random structures, and scores over 0.5 indicate structures about in the same fold (Xu and Zhang, 2010).

The problem with superposition scores is that they are sensitive to conformational changes. For example, consider a protein composed of two subunits connected by a small, flexible hinge; and a model where each of the subunits is perfect, but the relative orientation is not the same. The superposition scores would align one of the subunits, with perfect scoring, and give bad values to the other.

A solution is to compare structures locally. \( \text{LDDT} \) (Local Distance Difference Test) (Mariani et al., 2013) measures, for every atom, the fraction of preserved distances, up to a given tolerance, within an inclusion radius. \( \text{LDDT} \) considers all inter-atomic distances that are within 0.5 Å, 1 Å, 2 Å, and 4 Å. The final scores is obtained averaging over all the atoms in the residue and inclusion thresholds.

Olechnovič et al. (2012) proposed a similar method, using surface contacts instead of distances. They represent each residue as the Voronoi volume of its atoms. The surface area for the contact between residues \( i \) and \( j \) of the model is \( S_m(i, j) \), and for the reference – native structure – \( S_r(i, j) \). \( \text{CAD} \) score is proportional to the difference between the two, with some normalisation terms:

\[
\text{CAD score} = 1 - \frac{\sum_{i,j} \min (|S_r(i, j) - S_m(i, j)|, S_r(i, j))}{\sum_{i,j} S_r(i, j)}
\]
4.3.5 Model Quality Assessment, or Estimation of Model Accuracy

The previous section dealt with the case where we know the native structure, but what can we do when we do not know it? After all, if we have the experimental data, there is no need for modelling. The task of predicting the quality of a model is called Model Quality Assessment (MQA), or in some modern papers, Estimation of Model Accuracy (EMA). The purpose of MQA is two-fold:

- **Model selection**, or picking the best model from an ensemble generated by one or different methods.

- **Model evaluation**, or estimating the overall quality, or how much it can be trusted. Some applications may require high-quality models, while for others, knowing the right fold is sufficient.

**Energy-based MQA?**

In principle, we could compute the energy of each model, assuming that the lower the energy, the closer we are to our target. This has two flaws: it does not provide information about how far – since in principle we do not know the energy of the native state – and it is not a reliable way of comparing models from different methods – because energy functions are very sensitive to small details. Consider the case of a perfect model, except for two atoms that are in the same position. The scores presented on the previous section would all be close to 1, but the energy would be infinite!

We can partially remediate this problem by repacking the models using a common energy function, but this method would still favour models that were optimised with the same energy function we are using.

The alternative solution is to use machine learning and try to predict scores directly. We can use unsupervised programs, like Pcons (Lundström et al., 2008), that cluster the models, and select the one that is closer to the centre. The underlying assumption is that, if several independent methods agree, they are probably correct. If all our models come from a single method, but the set of restraints is good, they will all be similar; but if the restraints are insufficient or contradictory, the models will be diverse. They are called consensus methods.

The alternative is to use supervised machine learning, describe the model with a set of features, and try to predict in isolation its quality. The features we can use fall into two categories:

- Description of physico-chemical properties, such as energy terms and torsion angles.

- Agreement with predictors, such as secondary structure. A good model should have a good agreement with predictors, but not necessarily perfect.

This approach is called single method, and in this thesis, we have developed two supervised single methods (Papers I and IV).
To be able to predict a structure without a template, we need an crucial bit of additional information: the contact map, or the collection of residues that are in contact. We usually define contact as closer than 8Å between the C-β atoms, but other authors may use 10Å between C-α, 4Å between any pair of heavy atoms, or combinations.

Folded proteins are tightly packed structures, as illustrated in Figure 4.1. Since there is very little to no space between neighbouring residues, not every mutation is allowed. When a protein gains a mutation, it usually needs to be compensated by others somewhere else in the protein to keep it stable. For example, if an amino acid is replaced by a bigger one, a nearby residue would have to be changed for a smaller one; or if a neutral amino acid gains a positive charge, it would be energetically favourable to have a negatively charged next to it. Contacts in our protein, or residues nearby in space, will appear as sets of correlated, compensating mutations in an msa.

(a) Cartoon and sticks
(b) Atom-sized spheres

Figure 4.1: Fragment of a β-sheet in two representations: cartoon and sticks, and space filling spheres, showing the tightness of the packing. They are both seen from the same angle at the same scale.

4.4.1 Mutual Information

How can we formalise mathematically correlated mutations? Mutual Information codifies the information gain that we can obtain for one distribution knowing the other, and considering the relative distributions. The formula is:

\[
MI(i,j) = \sum_{x,y}^{q_{\text{max}}} f(x_i,y_j) \log \left( \frac{f(x_i,y_j)}{f(x_i)f(y_j)} \right),
\]

where \(f(x_i,y_j)\) is the joint distribution of amino acids \(x_i\) and \(y_j\) at positions \(i\) and \(j\), \(f(x_i)\) and \(f(y_j)\) are the marginal distributions, and \(q_{\text{max}}\) is the maximum number of amino acids types in the alignment, including gaps. Note that if \(x\) and \(y\) are independent, \(f(x_i,y_j) = f(x_i)f(y_j)\), the logarithm vanishes, and thus \(M(i,j) = 0\). If the msa is of enough quality, the contacts will present high values of MI.
4.4.2 Direct Coupling Analysis

Mutual Information has a problem with the transitivity property: consider three residues, A, B, and C; where B is close to both A and C, but A and C are far away. Mutual Information will likely detect a correlation between A and B, and between B and C; but it will also show a correlation between A and C!

To discern true from spurious correlations, we need to fit a statistical model to the whole data at once, in this way, we hope to recover the true (direct) relationships. This can be accomplished with Direct Coupling Analysis (DCA).

Several variations exist, but they are all based on a Potts model of statistical mechanics. This is a model where each position (in our case, residue), can take a number of discrete, well-defined, spin states (amino acid types), and the model depends only on the intrinsic properties at each location, and pairwise interactions.

The energy of a sequence of amino acids \( \vec{\sigma} \) takes the form:

\[
H(\vec{\sigma}) = \sum_{i,j=1}^{N} J_{ij}(\sigma_i, \sigma_j) + \sum_{i=1}^{N} h_i(\sigma_i),
\]

where \( h_i(\sigma_i) \) is the chemical potential of having a given amino acid at position \( i \), and \( J_{ij}(\sigma_i, \sigma_j) \) is the pairwise interaction between residues \( i \) and \( j \) given their amino acid species.

We can assume the proteins in our msa were generated by a similar model, so their frequency should follow the Boltzmann distribution:

\[
p(\vec{\sigma}|h, J) = \frac{1}{Z} e^{-\beta H(\vec{\sigma})}
\]

\( \beta \) is a scaling factor, the inverse of the temperature, and \( Z \) is the partition function, a normalisation term to ensure all the probabilities sum up to 1:

\[
Z = \sum_{\vec{\sigma}} e^{-\beta H(\vec{\sigma})}
\]

Fitting a Potts model means to estimate the values of \( J \) and \( h \) that best explain the observed distribution of sequences in the msa. The problem as such is intractable because computing \( Z \) implies a sum over all the possible sequences of length \( N \), \( 21^N \) (the 20 natural amino acids plus the gap state).

One solution is to replace the partition function with an approximation, such as done by plmDCA (Ekeberg et al., 2014); or side-step it all together, like GaussDCA (Baldassi et al., 2014).

Once the values of \( J \) are obtained, the scores of the contacts can be estimated by taking the Frobenius norm of each of the \( J \) matrices. That is, the square root of the sum of the squares of the couplings between each pair of amino acids:

\[
C(i, j) = \sqrt{\sum_{\sigma_i, \sigma_j=1}^{21} J_{ij}(\sigma_i, \sigma_j)^2}. 
\]
where $C(i,j)$ is the contact score between residues $i$ and $j$. This number cannot be readily interpreted as a probability.

This model approximations ignore the possibility of multiple rotamers for a given amino acid and consider that any interaction between three or more residues can be decomposed to the sum of each pair. Finally, DCA attempts to reconstruct the evolutionary couplings, not necessarily the contacts. Anishchenko et al. (2017) showed that most of the top-scoring pairs indicated by DCA are true contacts, others correspond to contacts between different subunits in a homodimer or pairs of separated residues involved in the function. The coupling is real, but its nature is different from the model explained at the beginning of this section.

### 4.4.3 Phylogenetic bias: the APC correction

Natural proteins are not randomly drawn from a generative model, like Potts. Proteins evolve alongside a tree, where a mutation in an organism is likely to be shared among all the descendants. This appears as a pattern of vertical and horizontal lines on the contact map, as seen on the upper triangle of Figure 4.2.

Dunn et al. (2008) proposed a simple method to reduce this bias: for each column in the contact map, subtract the average of the row times the average of the column, normalised by the total average.

![Figure 4.2: Contact scores without (upper) and with (lower) the APC correction. The patterns are clearer in the lower triangle, and most of the bias shown as vertical and horizontal stripes has been removed.](image)
4.4.4 Pattern recognition

Contacts do not appear at random. Since the protein is a continuous chain, contacts are rarely isolated, and appear in groups. Furthermore, since most of the protein is locally organised in secondary structure elements, this gives rise to specific patterns in the contact map, as can be seen in Figure 4.3. The statistical methods like dca and Mutual Information do not consider this, so a refinement step can be done using pattern recognition. We can train a machine learning algorithm on the outputs of statistical methods and recognise the underlying patterns, which can remove much of the spurious contacts, noise, and artefacts in the alignments.

This was the focus of our contributions on Paper IV.

(a) Alpha helices, with their characteristic chequerboard pattern. (b) Beta strands, showing the typical parallel bands

Figure 4.3: Typical contact maps of secondary structure elements.

4.5 Assessing the field: Casp

The Critical Assessment of protein Structure Prediction (CASP) is a biennial community-wide experiment to evaluate the status of the field. Over the course of a few months, they release the sequences of around 100 unpublished protein structures, asking groups to predict contact maps, 3D models, and predicting the quality of the submissions.

Since the structures are unpublished at the time of the experiment, this is a completely blind test and can be used to establish the current status of the field.
Part III

MY WORK
ProQ3D: improved model quality assessments using deep learning.
Bioinformatics, 2017.

ProQ3 (Uziela et al., 2016) was a MQA program developed in our lab that combines a large number of input features – such as atom and residue contacts, secondary structure predictors, and Rosetta energy terms – with an SVM. In this paper, we took the same features and dataset used to develop ProQ3, but replaced the machine learning algorithm with a multi-layer perceptron. This allows us to make full use of the whole dataset, and boosted its performance, improving the correlation between predicted and true from 0.85 to 0.90. The main limitation of ProQ3D from a theoretical point of view is that the features are engineered for a traditional machine learning framework, as opposed to deep learning. That is, many features and derivations of features, with no attempt to preserve the structure. From a practical consideration, ProQ3D requires several predictors and other programs, that while they are all freely available for researchers, we cannot distribute as a single package. Hence, installing it and running is slow and difficult. On the other hand, ProQ3 is the product of several generations of progressive improvements in model quality assessment, and ProQ3D keeps all of them, with a minimal, focalised change.


Here we present a pipeline for contact-based *ab initio* protein structure prediction. It predicts contacts with PconsC3, folds models with confold, and selects and evaluates the accuracy using Pcons and ProQ3. This work combines, in a single package, the programs for structure prediction developed in the group. We then applied to all the PFAM families, of both known and unknown structure, and benchmarked our quality estimation methods. We found out that a linear combination of the uniformity of the different models – computed with Pcons (Lundström et al., 2008) –, the ProQ3 predictions, and the agreement with predicted contacts yielded the best results. The better the contact map, the better the model; and the main driving factor is the quality and depth of the msa, with some exceptions. We can successfully model (TM > 0.5, 0.1 FPR) 558 PFAM families. Ovchinnikov et al. (2017) published a similar study based on metagenomics data. They are
successful in a comparable number of families, but only 26% of them coincide. This shows that independent methods are still useful, as they can succeed where others fail.

PAPER III


Wang et al. (2017) were the first group to bring the full force of deep learning to contact prediction, and have been consistently ranked amongst the top performing groups in casp. Unfortunately, their method is only available as a web server, and not for download. Other programs, such as PconsC3 Michel et al. (2017), developed in this lab, have a large number of dependencies, making it hard to install, and slow to run.

PconsC4 is a contact predictor designed to be fast and easy to use, clocking an average of 12 s per msa. This is more than twice as fast as the reference implementation of Gaussdca (Baldassi et al., 2014), and 244 times faster than PconsC3. We accomplished this through heavily optimised code and algorithms. It also requires a single msa and no external predictors, which makes it easy to install, deploy, and apply in large-scale studies.

Our method is based on a combination of dca (a faster re-implementation of Gaussdca), mutual information, and other statistics derived from the msa, combined and refined by a convolutional neural network (cnn). In order to train as big a network as possible, we used the U-net architecture, which allows for large receptive fields and number of parameters on the limited memory of our gpus. Our training data is taken from pisces (Wang and Dunbrack, Jr., 2003).

In order to maximise the information from our training data, we used multi-task learning (see Section 2.6) to predict contacts at 6, 8, and 10 Å, as well as the distance between residues under the transformation $1/(d + 20 \text{ Å})$.

At the time of publication, it was the best freely available contact predictor that could use a single msa.

PAPER IV

A novel training procedure to train deep networks in the assessment of the quality of protein models. Manuscript

The power of deep learning lies in its ability to leverage the inherent structure of the data. In this work, we show how we can take this one step further, and bake the structure of the problem in the network architecture, introducing ProQ4. We demonstrate it is a viable alternative reaching or surpassing ProQ3D’s performance (from Paper I) on the same training and test sets, but using only a subset of the inputs, following the line of simplicity outlined in PconsC4, and no external predictors.

ProQ4 was designed as a coarse-grained program with a focus on model selection and ranking. Coarse-grained means we are not overly sensitive to
finer details of a model, such as the side chain packing, which is relatively unimportant for MQA, yet may change widely from group to group. The last step of a protein structure prediction is to select the best model from our pool. An improvement in model selection will impact positively any protein structure prediction pipeline by selecting better candidates, and with more confidence.

In this paper we made extensive use of transfer learning, as explained in Section 2.6, in both multi-task and pre-training to extract the maximum amount of information and minimise the impact of bias. In particular, we have developed a model that can be used to both predict secondary structure from a msa, and learn a representation of said msa that can be applied for any other deep learning tasks.

**PAPER V**


This paper presents the results on model quality assessment of the latest edition of the blind test Critical Assessment of Protein Structure Prediction, casp13, and serves as independent validation for the results in Papers I and IV.

We can see that indeed, the results are better on the loss selected for training. The different ProQ3D-xx variants – where xx stands for different target functions, but using the same training data, model, and features – performs best on the target function was trained on.

ProQ4 has a better ranking than ProQ3D, but has poor performance in local scores. We believe this is because of the limited description of the protein, using only the properties of the backbone. Further research should include physico-chemical and finer-grained descriptors that could add orthogonal information.
ACKNOWLEDGEMENTS

Many people have helped me during the years, and this space is to thank them. I want to thank my supervisor, Arne Elofsson, for giving me the opportunity to join his lab. Also for giving me the freedom to explore, always having an open door for discussions and questions, and for putting up with my mistakes. My gratitude extends to my co-supervisor, Erik Lindahl, for his advice on presentation and communication. On the topic of supervisors, I am also thankful to Lukas Käll, for introducing me into Bioinformatics in particular, and the practice of science in general, and for getting me the job.

My research on deep learning would not have been possible without Heydar Maboudi Afkham, who gave me the necessary advice with surgical precision to set me on the right path. And what I had left to learn, I did observing Mirco Michel at work. I cannot forget Karolis Uziela, for offering his data and opinions during very fruitful discussions on our ongoing projects. And I thank all my colleagues at work, past and present, for the discussions over lunch over a wide assortment of topics, from pensions to medieval metaphysics.

But work has only been a fraction of the time. On a personal level, Seméli deserves many thanks for always being there, both helping me and challenging me to be my best. I also thank my adopted family, for their unconditional support and help navigating a new country.

Gracias a mi familia por todo el apoyo, sin el cual me lo habría tenido que pensar dos veces antes de lanzarme a cerca del círculo de los osos.

Finally, a salute to my friends at spiff, for all the mutual poking with metal sticks and keeping the second half of the adage mens sana in corpore sano. Another to the roleplaying troupe, for all the shared assaults on defenceless banana trucks and uncanny valley dolls. And to everyone who braved the cold and darkness and visited me here.

Most grateful.

David.


H. de Carrança. *De la philosophía de las armas, y de su destreza, y de la aggressión y defensión christiana*. Published by Alonso Pérez de Guzmán, Duke of Medina-Sidonia, Sanlucar de Barrameda, 1582.


