Chapter 5

Neural Networks: The Theory

5.1 Introduction

Artificial neural networks (NNs) [456, 252, 70] were originally developed with the goal of modeling information processing and learning in the brain. While the brain metaphor remains a useful source of inspiration, it is clear today that the artificial neurons used in most NNs are quite remote from biological neurons [85]. The development of NNs, however, has led to a number of practical applications in various fields, including computational molecular biology. NNs have become an important tool in the arsenal of machine-learning techniques that can be applied to sequence analysis and pattern recognition problems.

At the most basic level, NNs can be viewed as a broad class of parameterized graphical models consisting of networks with interconnected units evolving in time. In this book we use only pairwise connections but, if desirable, one can use more elaborate connections associated with the interaction of more than two units, leading to the "higher-order" or "sigma-pi" networks [456]. The connection from unit j to unit i usually comes with a weight denoted by w_{ij}. Thus we can represent an NN with a weight-directed graph or "architecture." For simplicity, we do not use any self-interactions, so that we can assume that $w_{ii} = 0$ for all the units.

It is customary to distinguish a number of important architectures, such as recurrent, feed-forward, and layered. A *recurrent* architecture is an architecture that contains directed loops. An architecture devoid of directed loops is said to be *feed-forward*. Recurrent architectures are more complex with richer dynamics and will be considered in chapter 9. An architecture is *layered* if the units are partitioned into classes, also called layers, and the connectivity patterns are defined between the classes. A feed-forward architecture is not necessarily layered.

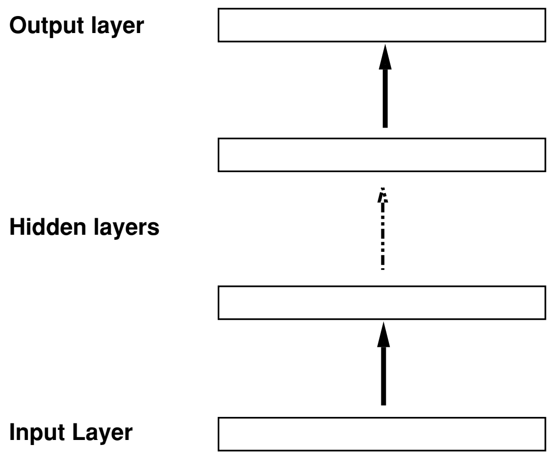


Figure 5.1: Layered Feed-Forward Architecture or Multilayer Perceptron (MLP). Layers may contain different numbers of units. Connectivity patterns between layers also may vary.

In most of this chapter and in many current applications of NNs to molecular biology, the architectures used are layered feed-forward architectures, as in figure 5.1. The units are often partitioned into *visible* units and *hidden* units. The *visible* units are those in contact with the external world, such as input and output units. Most of the time, in simple architectures the input units and the output units are grouped in layers, forming the input layer and the output layer. A layer containing only hidden units is called a hidden layer. The number of layers is often referred to as the "depth" of a network. Naturally NNs can be assembled in modular and hierarchical fashion to create complex overall architectures. The design of the visible part of an NN depends on the input representation chosen to encode the sequence data and the output that may typically represent structural or functional features.

The behavior of each unit in time can be described using either differential equations or discrete update equations (see [26] for a summary). Only the discrete formalism will be used in this book. In a layered feed-forward architecture, all the units in a layer are updated simultaneously, and layers are updated sequentially in the obvious order. Sometimes it is also advantageous to use stochastic units (see appendix C on graphical models and Bayesian networks). In the rest of this chapter, however, we focus on deterministic units. Typically a unit i receives a total input x_i from the units connected to it, and then produces an output $y_i = f_i(x_i)$, where f_i is the transfer function of the unit. In general, all the units in the same layer have the same transfer function, and the total input is a weighted sum of incoming outputs from the previous layer, so that

$$x_i = \sum_{j \in N^-(i)} w_{ij} y_j + w_i, \tag{5.1}$$

$$y_i = f_i(x_i) = f_i \left(\sum_{j \in N^-(i)} w_{ij} y_j + w_i \right), \tag{5.2}$$

where w_i is called the bias, or threshold, of the unit. It can also be viewed as a connection with weight w_i to an additional unit, with constant activity clamped to 1. The weights w_{ij} and w_i are the parameters of the NNs. In more general NNs other parameters are possible, such as time constants, gains, and delays. In the architectures to be considered here, the total number of parameters is determined by the number of layers, the number of units per layer, and the connectivity between layers. A standard form for the connectivity between layers is the “fully connected” one, where each unit in one layer is connected to every unit in the following layer. More local connectivity patterns are obviously more economical. Note, however, that even full connectivity between layers is sparse, compared with complete connectivity among all units. In situations characterized by some kind of translation invariance, it can be useful for each unit in a given layer to perform the same operation on the activity of translated groups of units in the preceding layer. Thus a single pattern of connections can be shared across units in a given layer. In NN jargon this is called “weight sharing.” It is routinely used in image-processing problems and has also been used with some success in sequence analysis situations where distinct features are separated by variable distances. The shared pattern of weights defines a filter or a convolution kernel that is used to uniformly process the incoming activity. With weight sharing, the number of free parameters associated with two layers can be small, even if the layers are very large. An example of this technique is given below in section 6.3 on secondary structure prediction.

There are a number of transfer functions that are widely used. Sometimes the transfer function is linear—like the identity function, as in regression problems, in which case the unit is called a linear unit. Most of the time, however, the transfer functions are nonlinear. Bounded activation functions are often called squashing functions. When f is a threshold function,

$$f(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{5.3}$$

the unit is also called a threshold gate. A threshold gate simulates a binary decision based on the weighted “opinion” of the relevant units. Obviously, the bias can be used to offset the location of the threshold. In this book we use a $(0, +1)$ formalism that is equivalent to any other scale or range, such as $(-1,+1)$. Threshold gates are discontinuous. Thus they are often replaced with sigmoidal transfer functions, which have the advantage of being continuous and differentiable. In this book, we use the *logistic* transfer function

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{5.4}$$

especially to estimate the probability of binary events. But other possible sigmoidal transfer functions lead to essentially equivalent results, such as $f(x) = \tanh(x)$ and $f(x) = \arctan(x)$. It is also possible to introduce a gain λ_i for each unit by writing $y_i = f_i(\lambda_i x_i)$. Another important type of unit in what follows is the *normalized exponential unit*, also called softmax, which is used to compute the probability of an event with n possible outcomes, such as classification into one of n possible classes. Let the index j run over a group of n output units, computing the n membership probabilities, and x_j denote the total input provided by the rest of the NN into each output unit. Then the final activity y_i of each output unit is given by

$$y_i = \frac{e^{-x_i}}{\sum_{k=1}^{n} e^{-x_k}}. \tag{5.5}$$

Obviously, in this case $\sum_{i=1}^{n} y_i = 1$. When $n = 2$, the normalized exponential is equivalent to a logistic function via a simple transformation

$$y_1 = \frac{e^{-x_1}}{e^{-x_1}+e^{-x_2}} = \frac{1}{1+e^{-(x_2-x_1)}}. \tag{5.6}$$

It is important to note that any probability distribution $P = (p_i)$ $(1 \le i \le n)$ can be represented in normalized exponential form from a set of variables x_j $(1 \le j \le m)$,

$$P_i = \frac{e^{-x_i}}{\sum_{k=1}^{m} e^{-x_k}}, \tag{5.7}$$

as long as $m \ge n$. This can be done in infinitely many ways, by fixing a positive constant K and letting $x_i = \log p_i + K$ for $i = 1, \dots, n$ (and $x_j = -\infty$ for $j > n$ if needed). If $m < n$ there is no exact solution, unless the p_i assume only m distinct values at most.

Another type of widely used functions is the *radial basis functions* (RBFs), where typically f is a bell-shaped function like a Gaussian. Each RBF unit i has a "reference" input x_i^*, and f operates on the distance $d(x_i^*, x_i)$ measured with respect to some metric $y_i = f(d(x_i^*, x_i))$. In spatial problems, d is usually the Euclidean distance.

Clearly a modeler should be able to choose the type of units, connectivity, and transfer functions as needed in relation to the task to be solved. As a result, the reader may be under the impression that the concept of NN is somewhat fuzzy, and rightly so! According to our loose definition, one can take the position that polynomials are NNs. Alternatively, one could of course put further restrictions on the definition of NNs. Historically, the term NN has been used mostly to refer to networks where the inputs satisfy (5.1) and the transfer functions are threshold functions or sigmoids. We do not think that much is to be gained by adopting such a dogmatic position. The current nomenclature of model classes is in part the product of historical accidents. The reality is that there is a continuous spectrum of possible parameterized models without precise boundaries. A modeler should be as free as possible in designing a model and proceeding with Bayesian inference.

In NN applications, it has been customary to distinguish between *regression* and *classification* or recognition problems. In regression problems, the goal is to approximate or fit a given surface. In classification or recognition problems, the goal is to be able to classify a given input into a relatively small number of classes. While useful, this distinction is also somewhat arbitrary since in the limit, classification—for example, into two classes—can be viewed as fitting a usually discontinuous binary function. The problem of learning the genetic code (see chapter 6) is a good example of a problem at the boundary of the two classes of problems. Classification problems have perhaps been slightly more frequent in past applications of NNs to molecular biology, due to the discrete nature of the sequence data and the standard problem of recognizing particular patterns such as alpha helices, fold classes, splice sites, or exons. But continuous data, such as hydrophobicity scales or stacking energies, can also be important. We shall examine both regression and classification NNs more closely in the coming sections.

One of the most important aspects of NNs is that they can learn from examples. Obviously, in the general Bayesian statistical framework this is nothing else than model fitting and parameter estimation. Very often the data D consist of input–output sample pairs $D = (D_1, \ldots, D_K)$, with $D_i = (d_i, t_i)$ (d for data, t for target) from the regression or classification function to be approximated. In practice, the data are often split into *training data* and *validation data* in some way. The training data are used for model fitting, and the validation data in model validation. The validation data can also be split into validation and *test data*, where the validation set is used for early stopping and the test data for assessing the overall performance of the model. These model-fitting tasks, where the target values of the outputs in the fitted data are known, are usually described in the literature as *supervised learning*. When the target values are not known, the terms *unsupervised* or *self-organization* are often used. Again, this historical distinction has its usefulness but should not be taken too dogmatically. As for supervised learning algorithms, one of the main practices in the past has been, starting from a random set of parameters, to define an “error function” by comparing the outputs produced by the network against the target outputs. Then the network parameters are optimized by gradient descent with respect to the error function. As pointed out in chapter 2, such practice is best analyzed in the general Bayesian statistical framework by explicitly stating the underlying probabilistic models and assumptions, and proceeding with the proper Bayesian inductions. Many forms of supervised and unsupervised learning for NNs in the literature can be viewed as ML or MAP estimation.

In the rest of the chapter we shall focus on layered feed-forward NN architectures, the multilayer perceptrons with inputs given by (5.1) and linear/threshold/sigmoidal/normalized exponential transfer functions, and their application within sequence analysis. In the next section, we briefly cover the universal approximation properties of NNs. In particular, we prove that any reasonable function can be approximated to any precision by a shallow, and possibly very large, NN. In section 5.3, we apply the general framework of chapter 2 to NNs. We examine priors and likelihood functions, how to design NN architectures, and how to carry out the first level of Bayesian inference. In section 5.4, we apply the general framework of chapter 4 to learning algorithms and derive the well-known backpropagation algorithm. Many other theoretical results on NNs, beyond the scope of this book, can be found in the references. Computational complexity issues for NNs and machine learning in general are reviewed in [314]. A more complete Bayesian treatment of NNs, including higher levels of Bayesian inference, is given in [373, 398, 517]. In addition to NNs, there are a number of other flexible parameterized models for regression and classification, such as splines [546], Gaussian processes [559, 206, 399] (appendix A), and support vector machines [533, 475].

5.2 Universal Approximation Properties

Perhaps one reassuring property of NNs is that they can approximate any reasonable function to any degree of required precision. The result is trivial[1] for Boolean functions, in the sense that any Boolean function can be built using a combination of threshold gates. This is because any Boolean function can be synthesized using NOT and AND gates, and it is easy to see that AND and NOT gates can be synthetized using threshold gates. For the general regression case, it can be shown that any reasonable real function $f(x)$ can be approximated to any degree of precision by a three-layer network with x in the input layer, a hidden layer of sigmoidal units, and one layer of linear output units,

[1]This section concentrates primarily on threshold/sigmoidal units. Obviously the result is also well known if polynomials are included among NNs.

as long as the hidden layer can be arbitrarily large. There are a number of different mathematical variations and proofs of this result (see, e.g., [264, 265]).

Here we give a simple constructive proof of a special case, which can easily be generalized, to illustrate some of the basic ideas. For simplicity, consider a continuous function $y = f(x)$ where both x and y are one-dimensional. Assume without loss of generality that x varies in the interval $[0, 1]$, and that we want to compute the value of $f(x)$ for any x within a precision ϵ. Since f is continuous over the compact interval $[0, 1]$, f is uniformly continuous and there exists an integer n such that

$$|x_2 - x_1| \leq \frac{1}{n} \Longrightarrow |f(x_2) - f(x_1)| \leq \epsilon. \tag{5.8}$$

Therefore it is sufficient to approximate f with a function g such that $g(0) = f(0)$, and $g(x) = f(k/n)$ for any x in the interval $((k-1)/n, k/n]$ and any $k = 1, \ldots, n$. The function g can be realized exactly by a NN with one input unit representing x, $n+1$ hidden threshold gate units all receiving connections from the input unit, and one output unit receiving a connection from each hidden unit. The hidden units are numbered from 0 to n. The output has a linear transfer function in order to cover the range of ys (figure 5.2). All the weights from the input unit to the n hidden units are set to 1, and the kth hidden unit has a threshold (bias) of $(k-1)/n$. Thus, for any x in the interval $((k-1)/n, k/n]$, all the hidden unit activations are set to 0 except for the first $k+1$, which take the value 1. Thus the value of the input is directly coded in the number of hidden units that are turned on. The weight of the connection from the kth hidden unit to the output unit is $\Delta_k f = f(k/n) - f(k-1/n)$, with $\Delta_0 f = f(0)$. The output unit is just the identity function, with 0 bias. Thus if $x = 0$, $g(x) = 0$. For any $k = 1, 2, \ldots, n$, if x is in the interval $[(k-1)/n, k/n]$, then $g(x) = f(0) + \sum_{j=1}^{k} f(j/n) - f(j-1/n) = f(k/n)$, as desired.

It should be clear that it is not too difficult to generalize the previous result in several directions, to encompass the following:

1. Multidimensional inputs and outputs

2. Sigmoidal transfer functions and other types

3. Inputs on any compact set

4. Functions f that may have a finite number of discontinuities and more

While it is useful to know that any function can be approximated by an NN, the key point is that the previous proof does not yield very economical architectures. In fact, one can show that for essentially random functions, compact architectures do not exist. It is only for "structured" functions that compact

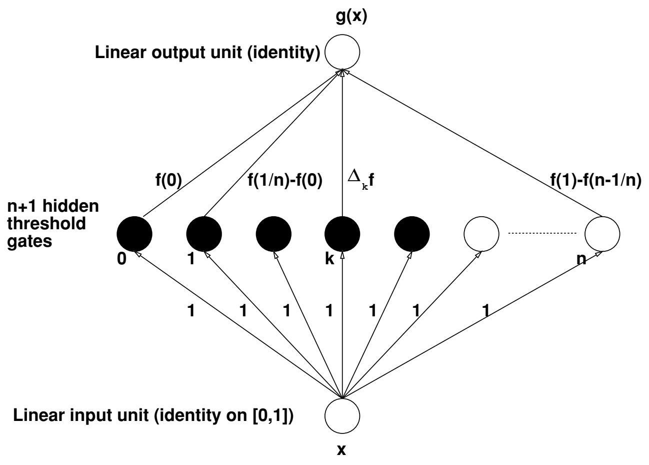

g(x)
Linear output unit (identity)
f(0)
f(1/n)-f(0)
Δ_k f
f(1)-f(n-1/n)
n+1 hidden threshold gates
0
1
k
n
1
1
1
1
1
1
1
Linear input unit (identity on [0,1])
x

Figure 5.2: Universal Approximation Architecture with One Input Unit, $n + 1$ Hidden Threshold Gate Units, and One Linear Output Unit Computing the Approximation $g(x)$ to $f(x)$.

architectures exist, and in this case the architecture constructed in the universal approximation theorems are far from optimal. Better architectures may exist, with a better allocation of hidden units, and possibly with more than a single hidden layer. It is for these cases that learning approaches become important.

5.3 Priors and Likelihoods

We now apply the general theory of chapter 2. In particular, we show how the theory can be used to determine the choice of an objective function and of the transfer functions of the output units. In this section we shall assume that the data consist of a set of independent input–output pairs $D_i = (d_i, t_i)$. The data are noisy in the sense that for a given d_i, different outputs t_i could be observed. Noise at the level of the input d could also be modeled, but will not be considered here. The operation of the NN itself is considered to be deterministic. We have

$$\mathbf{P}((d_i, t_i)|w) = \mathbf{P}(d_i|w)\mathbf{P}(t_i|d_i, w) = \mathbf{P}(d_i)\mathbf{P}(t_i|d_i, w), \tag{5.9}$$

the last equality resulting from the fact that in general we can assume that the inputs d are independent of the parameters w. Thus, for a given architecture parameterized by w, we have, using (2.9),

$$-\log \mathbf{P}(w|D) = -\sum_{i=1}^{K} \log \mathbf{P}(t_i|d_i, w) - \sum_{i=1}^{K} \log \mathbf{P}(d_i) - \log \mathbf{P}(w) + \log \mathbf{P}(D), \quad (5.10)$$

where we have used the fact that $\mathbf{P}((d_i, t_i)|w) = \mathbf{P}(d_i)P(t_i|d_i, w)$, and have taken into account the independence of the different data points. In the first level of Bayesian inference (MAP), we want to minimize the left-hand side. We can ignore $\mathbf{P}(D)$ as well as $\mathbf{P}(d_i)$, since these terms do not depend on w, and concentrate on the prior term and the likelihood.

In order to calculate the likelihood, we shall have to distinguish different cases, such as regression and classification, and further specify the probabilistic model. In doing so, we follow the analysis in [455]. But the basic idea is to consider that, for a given input d_i, the network produces an estimated output $y(d_i)$. The model is entirely defined when we specify how the observed data $t_i = t(d_i)$ can statistically deviate from the network output $y_i = y(d_i)$. If the output layer has many units, we need to write y_{ij} for the output of the jth unit on the ith example. For notational convenience, in what follows we will drop the index that refers to the input. Thus we derive online equations for a generic input–output pair (d, t). Offline equations can easily be derived by summing over inputs, in accordance with (5.10).

5.3.1 Priors

Unless additional information is available, the most natural and widely used priors for NN parameters are zero-mean Gaussian priors. Hyperparameters, such as the standard deviation of the Gaussians, can be chosen differently for connection weights and biases and for units in different layers. If a weight w has a Gaussian prior with standard deviation σ, the corresponding contribution to the negative log-posterior, up to constant factors, is given by $w^2/2\sigma^2$. This can also be viewed as a regularization factor that penalizes large weights often associated with overfitting. In gradient-descent learning, this adds a factor $-w/\sigma^2$ to the update of w. This factor is also called *weight decay*. *Weight sharing* is a different kind of prior obtained when different groups of units in a given layer are assumed to have identical incoming connection weights. Weight sharing is easily enforced during gradient-descent learning. It is useful in problems characterized by some form of translational invariance where the same operation, such as the extraction of characteristic features, needs to be applied to different regions of the input. The pattern of shared units essentially implements a convolution kernel, whence the name *convolutional networks*.

Gaussian and other priors for NN parameters and hyperparameters are studied in detail in [373, 398, 517]. In [373] Laplace approximation techniques are used to determine optimal hyperparameters. In [398] Monte Carlo methods are derived for the integration of priors and Bayesian learning in MLPs. The advantages of Bayesian learning include the automatic determination of regularization parameters without the need for a validation set, the avoidance of overfitting when using large networks, and the quantification of prediction uncertainty. In [398] it is shown that in the limit of a single hidden layer with an infinite number of hidden units, an NN with Gaussian weight priors defines a Gaussian process on the space of input–output functions. Hence the idea of using Gaussian processes directly [559, 399, 206], bypassing any NN implementation. While Gaussian processes provide a very flexible tool for both regression and classification problems, they are computationally demanding and can be applied only to moderate-size problems with currently available technology.

5.3.2 Gaussian Regression

In the case of regression, the range of y can be arbitrary, and therefore the simplest transfer functions in the output layer are linear (actually the identity) functions. It is also natural to assume a Gaussian probabilistic model, that is, $\mathbf{P}(t|d,w) = \mathbf{P}(t|y(d),w) = \mathbf{P}(t|y)$ is Gaussian, with mean vector $y = y(d)$. Assuming further that the covariance matrix is diagonal and that there are n output units indexed by j, we have

$$\mathbf{P}(t|d,w) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(t_j - y_j)^2}{2\sigma_j^2}\right). \tag{5.11}$$

The standard deviations σ_j are additional parameters of this statistical model. If we further assume that they are constant $\sigma_j = \sigma$, then the negative log-likelihood for the current input boils down to

$$\mathcal{E} = \sum_j \left(\frac{(t_j - y_j)^2}{2\sigma^2} - \frac{1}{2}\log 2\pi - \log\sigma \right). \tag{5.12}$$

Again the last two terms are independent of w, and can be ignored while trying to estimate the optimal set of parameters w. The first term of course is the usual least-mean-square (LMS) error, routinely used in many applications, sometimes without explicating the underlying statistical model. The derivative of the negative log-likelihood $\mathcal{E}$ with respect to an output y_j is

$$\frac{\partial \mathcal{E}}{\partial y_j} = \frac{\partial \mathcal{E}}{\partial x_j} = -\frac{t_j - y_j}{\sigma_j} = -\frac{t_j - y_j}{\sigma}, \tag{5.13}$$

the first equality resulting from the assumption that the output transfer function is the identity.

In summary, we see that in the regression case with Gaussian noise, the output transfer function should be linear, the likelihood error function is the LMS error function (possibly scaled by σ_j along each component j), and the derivative of $\mathcal{E}$ with respect to the total input activity into the output layer, for each example, has the simple expression $-(t_j - y_j)/\sigma_j = -(t_j - y_j)/\sigma$.

5.3.3 Binomial Classification

Consider now a classification problem with only two classes, A and $\bar{A}$. For a given input d, the target output t is 0 or 1. The natural probabilistic model is a binomial model. The single output of the network then represents the probability that the input is a member of the class A or $\bar{A}$, that is the expectation of the corresponding indicator function. This can be computed by a sigmoidal transfer function. Thus,

$$y = y(d) = \mathbf{P}(d \in A) = \mathbf{P}(t|d, w) = y^t(1-y)^{(1-t)} \tag{5.14}$$

and

$$\mathcal{E} = -\log \mathbf{P}(t|d, w) = -t \log y - (1-t)\log(1-y). \tag{5.15}$$

This is the relative entropy between the output distribution and the observed distribution, and

$$\frac{\partial \mathcal{E}}{\partial y} = -\frac{t-y}{y(1-y)}. \tag{5.16}$$

In particular, if the output transfer function is the logistic function, then

$$\frac{\partial \mathcal{E}}{\partial x} = -(t-y). \tag{5.17}$$

Therefore, in the case of binomial classification, the output transfer function should be logistic; the likelihood error function is essentially the relative entropy between the predicted distribution and the target distribution. The derivative of $\mathcal{E}$ with respect to the total input activity into the output unit, for each example, has the simple expression $-(t-y)$.

5.3.4 Multinomial Classification

More generally, consider a classification task with n possible classes $A_1, \ldots, A_n$. For a given input d, the target output t is a vector with a single 1 and $n-1$ zeros. The most simple probabilistic model is a multinomial model. The corresponding NN has n output units, each one giving the probability of the membership of the input in the corresponding class. Thus

$$\mathbf{P}(t|d,w) = \prod_{j=1}^{n} y_j^{t_j}, \tag{5.18}$$

with, as usual, $t_j = t_j(d)$ and $y_j = y_j(d)$. For each example,

$$\mathcal{E} = -\log \mathbf{P}(t|d,w) = -\sum_{j=1}^{n} t_j \log y_j. \tag{5.19}$$

Again, this is the relative entropy between the output distribution and the observed distribution, and

$$\frac{\partial \mathcal{E}}{\partial y_j} = -\frac{t_j}{y_j}. \tag{5.20}$$

In particular, if the output layer consists of a set of normalized exponentials, then for each input d_i,

$$\frac{\partial \mathcal{E}}{\partial x_j} = -(t_j - y_j), \tag{5.21}$$

where x_j is the total input into the jth normalized exponential.

Thus, in multinomial classification, the output transfer function should be normalized exponentials. The likelihood error function is essentially the relative entropy between the predicted distribution and the target distribution. The derivative of E with respect to the total input activity into the output layer, for each example and each component, has the simple expression $-(t_j - y_j)$.

5.3.5 The General Exponential Family Case

In fact, results similar to the previous cases can be derived every time the likelihood function belongs to the exponential family of distributions (see appendix A and [384, 94]). The exponential family contains many of the most common distributions such as Gaussian, gamma, binomial, multinomial, exponential, beta, Poisson, and negative binomial. For each member of the family, there is an appropriate choice of output transfer function $y = f(x)$ such that the derivative $\partial\mathcal{E}/\partial x_j$ of E with respect to the total input activity into the jth output unit has a simple expression, proportional for each example to $(t_j - y_j)$, the difference between the target output t_j and the actual output y_j.

We have just seen that the proper statistical framework allows one to construct suitable transfer functions for the output layer, as well as suitable error functions to measure network performance. The design of the hidden layers, however, is more problem-dependent, and cannot be dealt with in much

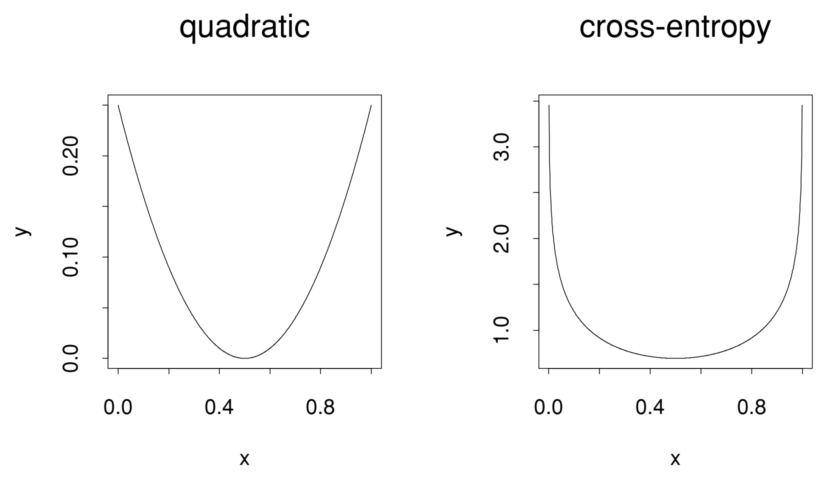


Figure 5.3: Comparison of the One-Dimensional Quadratic and Cross-Entropy Error Functions, with Respect to the Target Value of 0.5. Note the difference in ranges: the cross-entropy is infinite for $x = 0$ and $x = 1$.

generality. The framework described above has emerged only in recent years, and has not always been followed by NN practitioners, including many of the examples to be examined in the next sections. Many authors have used an LMS error function even in binomial classification problems, where a relative entropy error is more appropriate.

The question, then, is: "How have reasonably good results been derived, even when using a somewhat improper framework?" The answer to this question is best understood in the simple example above. Suppose that in a binary classification problem, the probability we wish to learn is, for the sake of argument, $p = 0.5$. For each x in $[0, 1]$ the LMS error is $(0.5 - x)^2$, whereas the relative entropy is $-0.5 \log x - 0.5 \log(1 - x)$. These two functions are plotted in figure 5.3. Both functions are convex (∪), with a a minimum at $p = 0.5$, as desired. The main difference, however, is in the dynamic range: unlike the relative entropy, the LMS error is bounded. The dynamic range difference can be important when the errors of many examples are superimposed, and also during learning.

5.4 Learning Algorithms: Backpropagation

In the majority of applications to be reviewed, MAP or ML estimation of NN parameters is done by gradient descent (see [26] for a general review). The calculations required to obtain the gradient can be organized in a nice fashion that leverages the graphical structure of NN. Using the chain rule, weights are updated sequentially, from the output layer back to the input layer, by propagating an error signal backward along the NN connections (hence the name "backpropagation"). More precisely, in the online version of the algorithm, and for each training pattern, we have for any weight parameter w_{ij}

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial y_i} f_i'(x_i) y_j. \tag{5.22}$$

Thus the gradient-descent learning equation is the product of three terms,

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}}{\partial w_{ij}} = -\eta \epsilon_i y_j, \tag{5.23}$$

where η is the learning rate, y_j is the output of the unit from which the connection originates (also called the presynaptic activity), and $\epsilon_i = (\partial \mathcal{E}/\partial y_i) f_i'(x_i)$ is a postsynaptic term called the backpropagated error. The backpropagated error can be computed recursively by

$$\frac{\partial \mathcal{E}}{\partial y_i} = \sum_{k \in N^+(i)} \frac{\partial \mathcal{E}}{\partial y_k} f_k'(x_k) w_{ki}. \tag{5.24}$$

The propagation from the children of a node to the node itself is the signature of backpropagation. While backpropagation is the most widely used algorithm for MAP estimation of MLPs, EM and simulated annealing have also been used. Algorithms for learning the architecture itself can also be envisioned, but they remain inefficient on large problems.

We can now review some of the main applications of NNs to molecular biology. Other general surveys of the topics can be found in [432, 571, 572].

Chapter 6

Neural Networks: Applications

The application of neural network algorithms to problems within the field of biological sequence analysis has a fairly long history, taking the age of the whole field into consideration. In 1982 the perceptron was applied to the prediction of ribosome binding sites based on amino acid sequence input [506]. Stormo and coworkers found that the perceptron algorithm was more successful at finding *E. coli* translational initiation sites than a previously developed set of rules [507]. A perceptron without hidden units was able to generalize and could find translational initiation sites within sequences that were not included in the training set.

This linear architecture is clearly insufficient for many sequence recognition tasks. The real boost in the application of neural network techniques first came after the backpropagation training algorithm for the multilayer perceptron was brought into common use in 1986 [456], and especially after Qian and Sejnowski published their seminal paper on prediction of protein secondary structure in 1988 [437]. This and other papers that quickly followed [78, 262] were based on an adaptation of the NetTalk multilayer perceptron architecture [480], which from its input of letters in English text predicted the associated phonemes needed for speech synthesis and for reading the text aloud. This approach could immediately be adapted to tasks within the field of sequence analysis just by changing the input alphabet into alphabets of the amino acids or nucleotides. Likewise, the encoding of the phonemes could easily be transformed into structural classes, like those commonly used for the assignment of protein secondary structure (helices, sheets, and coil), or functional categories representing binding sites, cleavage sites, or residues being posttranslationally modified.

In this chapter we review some of the early work within the application areas of nucleic acids and proteins. We go into detail with some examples of more recent work where the methodologies are advanced in terms of either the training principles applied or the network architectures, especially when networks are combined to produce more powerful prediction schemes. We do not aim to mention and describe the complete spectrum of applications. For recent reviews see, for example, [432, 61, 77, 320, 571, 572].

6.1 Sequence Encoding and Output Interpretation

One important issue, before we can proceed with NN applications to molecular biology, is the encoding of the sequence input. In any type of prediction approach, the input representation is of cardinal importance. If a very clever input representation is chosen, one that reveals exactly the essentials for a particular task, the problem may be more or less solved, or at least can be solved by simple linear methods. In an MLP the activity patterns in the last hidden layer preceding the output unit(s) should represent the transformed input information in linearly separable form. This clearly is much easier if the input representation has not been selected so as further to increase the nonlinearity of the problem.

One would think that a very "realistic" encoding of the monomers in a sequence, using a set of physical-chemical features of potential relevance, should always outperform a more abstract encoding taken from the principles and practice of information theory [137]. However, in line with the contractive nature of most prediction problems (see section 1.4), it does not always help just to add extra information because the network has to discard most of it before it reaches the output level.

During training of an MLP, the network tries to segregate the input space into decision regions using hyperplanes. The numerical representation of the monomers therefore has a large impact on the ease with which the hidden units can position the planes in the space defined by the representation that has been chosen.

In many sequence analysis problems, the input is often associated with a window of size W covering the relevant sequence segment or segments. Typically the window is positioned symmetrically so that the upstream and downstream contexts are of the same size, but in some cases asymmetric windows perform far better than symmetric ones. When the task is to predict signal peptide cleavage sites (section 6.4) or intron splice sites in pre-mRNA (section 6.5.2), asymmetric windows may outperform symmetric ones. Both these sequence types (N-terminal protein sorting signals and noncoding intronic DNA) are eventually removed, and it makes sense to have most of the features needed for their processing in the regions themselves, leaving the mature protein least constrained. Windows with holes where the sequence appears nonconsecutively have been used especially for the prediction of promoters and the exact position of transcriptional initiation in DNA, but also for finding beta-sheet partners in proteins [268, 46] and for the prediction of distance constraints between two amino acids based on the sequence context of both residues [368, 174].

For each position in a window W, there are $|\mathbf{A}|$ different possible monomers. The most used representation is the so-called *orthogonal* (also called local, as opposed to distributed) encoding, where the letters $\mathsf{X}_1, \mathsf{X}_2, \ldots$ are encoded by the orthogonal binary vectors $(1, 0, \ldots, 0)$, $(0, 1, \ldots, 0)$, and so on. Such a representation has the advantage of not introducing any algebraic correlations between the monomers. N- and C-terminal positions in incomplete windows of amino acid sequences are usually encoded using a dedicated character. Sometimes this character is also used to encode unknown monomers in a sequence, but unknown monomers may be handled better using just a string of zeros so that they have no impact on the input layer.

The sparse encoding scheme has the disadvantage of being wasteful because it requires an input layer of size $|\mathbf{A}| \times W$. $|\mathbf{A}|$ letters could in principle be encoded using as few as $\log_2 |\mathbf{A}|$ binary units. Furthermore, using continuous values in the input layer of an MLP, even a single unit could encode all possible letters. Such a compact encoding would in almost all cases give rise to drastically increased nonlinearity in the prediction problem at hand. If all amino acids were encoded using values between, say, 0 and 1, many of the induced correlations between the monomers would have no biological relevance, almost no matter in what order the monomers were mapped to the interval.

Obviously, there are trade-offs between different encodings that involve the complexity of the space in which the input windows live, the network architecture size, and ease of learning. In much of the best work done so far in this field, the orthogonal representation has been the most successful encoding scheme. With a more complex encoding of the sequence, whether orthogonal or not, the network must filter this extra information through a representation as a point in a space with dimensionality according to the number of hidden units, and then further on to a few, often a single, output unit(s). If one includes too much extra information related to the physicochemical properties of the residues in the input layer, possibly information that is not strongly correlated to the output, one makes the network's task harder. In this case, it is best to use more hidden units in order to be able to discard this extra information and find the relevant features in a sea of noise. This situation, with the lack of a better alternative, has contributed to the success of the orthogonal representation.

If one wants to use real-numbered quantification of residue hydrophobicity, volume, charge, and so on, one should be aware of the harmful impact it can have on the input space. Instead of just using a seemingly better representation of the input residues, it may be much better to use preprocessed versions of the original sequence segments. When designing such preprocessed versions, one may exploit the statistics of certain words present in the window, the average hydrophobicity over the window or separately in the left and right parts of a symmetric window, and so on. Another interesting possibility, demonstrated in one of the examples below, is to let an NN learn its own representation. In another example a binary word encoding was shown to have a positive effect on protein secondary structure prediction [313, 548, 17]. In this case, it was possible, from the optimal encoding scheme generated by a simulated annealing approach, to discover physicochemical properties relevant to the formation of secondary structure.

An important strategy for decreasing the nonlinearity of a prediction problem is to switch from a representation based on monomers to one based on dimers or trimers. In the case of nucleotides, 16- and 64-letter alphabets result, and in a large number of biological recognition problems, the pair or triplet correlations are so large that the gain in significant correlations compares favorably with the negative impact of the increased dimensionality of the input space. In DNA, base pair stacking is the most important thermodynamic contribution to helical stability (more important than base pairing). Pair correlations in RNA–RNA recognition interactions, for example, have their physical basis in the stacking energies between adjacent base pairs [112]. In proteins the dipeptide distribution similarly has a strong bias associated with steric hindrance, translation kinetics, and other purely biochemical factors.

If RNA and DNA sequences are encoded by dinucleotides or trinucleotides, there is also the possibility of letting the multimers *overlap*. The sparse encoding of the multimers ensures that no a priori relationship is imprinted on the sequence data. The advantage of the encoding of the sequence as overlapping triplets is that the hidden units directly receive context information for each single nucleotide that they otherwise would have to deduce from the training process.

Yet another strategy for decreasing (or in some cases increasing) the nonlinearity of a prediction problem is to group monomers from one alphabet to form new alphabets in which the pattern that should be detected will have more contrast to the background [306]. The reduced alphabets can then be encoded using the orthogonal vector representation, and at the same time reduce the dimensionality of the input space and thus the number of adjustable parameters in the network. Meaningful groupings can be based on physicochemical properties or on estimated mutation rates found in evolutionary studies of protein families. Table 6.1 lists some of the previously used groupings based on ab initio descriptions of the monomers or on their structural or functional preferences as observed in experimental data.

Molecule	Size	Grouping
DNA	2	Purines vs. pyrimidines: R = A, G; Y = C, T
DNA	2	Strong vs. weak hydrogen bonding: S = C, G; W = A, T
DNA	2	Less physiochemical significance:
		keto, K = T, G vs. amino, M = A, C
Protein	3	Structural alphabet:
		ambivalent (Ala, Cys, Gly, Pro, Ser, Thr, Trp, Tyr)
		external (Arg, Asn, Asp, Gln, Glu, His, Lys)
		internal (Ile, Leu, Met, Phe, Val)
Protein	8	Chemical alphabet:
		acidic (Asp, Glu)
		aliphatic (Ala, Gly, Ile, Leu, Val)
		amide (Asn, Gln)
		aromatic (Phe, Trp, Tyr)
		basic (Arg, His, Lys)
		hydroxyl (Ser, Thr):
		imino (Pro)
		sulfur (Cys, Met)
Protein	4	Functional alphabet:
		acidic and basic (same as in chemical alphabet)
		hydrophobic nonpolar (Ala, Ile, Leu, Met, Phe, Pro, Trp, Val)
		polar uncharged (Asn, Cys, Gln, Gly, Ser, Thr, Tyr)
Protein	3	Charge alphabet:
		acidic and basic (as in chemical alphabet)
		neutral (all the other amino acids)
Protein	2	Hydrophobic alphabet:
		hydrophobic (Ala, Ile, Leu, Met, Phe, Pro, Trp, Val)
		hydrophilic
		(Arg, Asn, Asp, Cys, Gln, Glu, Gly, His, Lys, Ser, Thr, Tyr)

Table 6.1: Merged Alphabets of Biomolecular Monomers. Some of these alphabets are based on ab initio descriptions of the monomers, others are derived from statistical properties of the monomers as indicated by structural or functional preferences. Random partition of the amino acids into *k* classes that maximizes a similarity measure between sequences can also be constructed. Source: [306]. See also references therein.

In one case, it has been shown recently that a protein can largely maintain its folded structure, even if the total number of different amino acids in its composition is reduced from the conventional twenty down to five [443]. Apart from a few positions close to a binding site, fifteen amino acid types were replaced by other residues taken from the smaller, representative group of five (I, K, E, A, and G). Further reduction in the diversity down to three different amino acids did not work. This means that proteins in earlier evolutionary time still may have been able to obtain stable, folded structures with a much smaller repertoire of amino acid monomers. It should be noted that this reduced alphabet is in no way canonical: many proteins will certainly not be able to do without cysteines. While the recoding of sequences using smaller alphabets (table 6.1) at first may seem purely a computational trick, more experimental work on "essential" amino acids can possibly be exploited in bioinformatics approaches to construct simpler sequence spaces that can be better covered by the limited amounts of data available. The simplification strategy arrived at here was also inspired by the phylogenetic variation in this protein. This is exactly the type of information that has been used to improve protein structure prediction methods, as described in the next sections of this chapter.

In other applications the encoding does not preserve the consecutive order of the residues in a sequence but seeks to cast the whole sequence, or large segments of it, into global preprocessed measures that can be used as input information in a network. For example, this is the case when the aim is to predict the fold class or family relationship of a protein from the frequencies of the 400 dipeptides it contains [179, 18]. In the indirect sequence encoding used in one approach to discriminate between exons and introns, 6-mer statistics, GC composition, sequence vocabulary, and several other indicators are included as global measures in the input layer [529].

In the NN applications described below, we also show how important it is to design good strategies for *output interpretation* or postprocessing. In most cases, however, intelligent postprocessing may be as important as, or even more important than, selecting optimal network architectures in terms of the smallest numerical generalization error as quantified by the activities of the output neurons. Often the number of output neurons corresponds directly to the number of output classes, again using sparse encoding by orthogonal vectors of zeros and ones. The output interpretation and postprocessing will always be designed individually for each task, based on features known previously from the biological frame of reference. If it is known a priori that, say, alpha-helices in proteins have a minimum length of four amino acids, small "helices" that are predicted can often be removed and lead to a better overall predictive performance. In cases where a sequence is known to possess a *single* functional site of a given type only—for example, a cleavage site in the N-terminal signal peptide—a carefully designed principle for the numerical threshold used for assignment of sites may lead to much better recognition of true sites and significantly lower rates of false positives. A discussion of the relation between the analog network error and the discrete classification error can be found in [90].

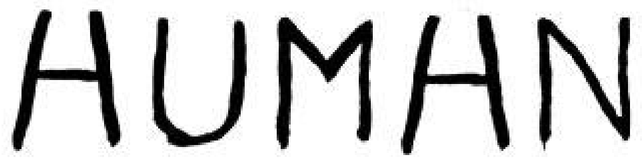


Figure 6.1: English-Reading People Will Normally Interpret the Two Identical Symbols in this Word Differently: the first as an *h* and the second as an *a*. In biological sequences a similar information processing capability is needed as structural and functional features most often result from the cooperativity of the sequence rather than from independent contributions from individual nucleotides or amino acids. The neural network technique has the potential to detect such short- and long-range sequence correlations, and in this way complement what can be obtained by conventional alignment and analysis by hidden Markov models.

6.2 Sequence Correlations and Neural Networks

Many structural or functional aspects of sequences are not conserved in terms of sequence, not even when amino acid similarities are taken into account. It is well known that protein structures, for example, can be highly conserved despite a very low sequence similarity when assessed and quantified by the amino acid identity position by position. What makes up a protein structure, either locally or globally, is the *cooperativity* of the sequence, and not just independent contributions from individual positions in it.

This holds true not only for the protein as a whole but also locally, say for a phosphorylation site motif, which must be recognized by a given kinase. Even for linear motifs that are known to interact with the same kinase, sequence patterns can be very different [331]. When the local structures of such sequence segments are inspected (in proteins for which the structure has been determined and deposited in the Protein Data Bank), they may indeed be conserved structurally despite the high compositional diversity [74].

The neural network technique has the potential of sensing this cooperativity through its ability to correlate the different input values to each other. In fact, the cooperativity in the weights that result from training is supposed to mirror the relevant correlations between the monomers in the input, which again are correlated to the prediction task carried out by the network.

The ability of the artificial neural networks to sense correlations between individual sequence positions is very similar to the ability of the human brain when interpreting letters in natural language differently based on their language!naturalcontext. This is well known from pronunciation where, for example, the four a's in the sentence *Mary had a little lamb* correspond to three different phonemes [480]. Another illustration of this kind of ability is shown in figure 6.1. Here the identical symbol will be interpreted differently *provided* the brain receiving the information that is projected onto the retina has been trained to read the English language, that is, trained to understand the sequential pattern in English language!Englishtext.

It is precisely this ability that has made the neural networks successful in the sequence analysis area, in particular because they complement what one can obtain by weight matrices and to some degree also by hidden Markov models. The power of the neural network technique is not limited to the analysis of local correlations, as the sequence information being encoded in the input layer can come from different parts of a given sequence [368]. However, most applications have focused on local and linear sequence segments, such as those presented in the following sections.

6.3 Prediction of Protein Secondary Structure

When one inspects graphical visualizations of protein backbones on a computer screen, local folding regularities in the form of repeated structures are immediately visible. Two such types of secondary structures, which are maintained by backbone hydrogen bonds, were actually suggested by theoretical considerations before they were found in the first structures to be solved by X-ray crystallography. There is no canonical definition of classes of secondary structure, but Ramachandran plots representing pairs of dihedral angles for each amino acid residue show that certain angular regions tend to be heavily overrepresented in real proteins. One region corresponds to alpha-helices, where backbone hydrogen bonds link residues i and $i + 4$; another, to beta-sheets, where hydrogen bonds link two sequence segments in either a parallel or antiparallel fashion.

The sequence preferences and correlations involved in these structures have made secondary structure prediction one of the classic problems in computational molecular biology [362, 128, 129, 196]. Many different neural network architectures have been applied to this task, from early studies [437, 78, 262, 370, 323] to much more advanced approaches [453, 445].

The assignment of the secondary structure categories to the experimentally determined 3D structure is nontrivial, and has in most of the work been performed by the widely used DSSP program [297]. DSSP works by analysis of the repetitive pattern of potential hydrogen bonds from the 3D coordinates of the backbone atoms. An alternative to this assignment scheme is the program STRIDE, which uses both hydrogen bond energy and backbone dihedral angles rather than hydrogen bonds alone [192]. Yet another is the program DEFINE, whose principal procedure uses difference distance matrices for evaluating the match of interatomic distances in the protein to those from idealized secondary structures [442].

None of these programs can be said to be perfect. The ability to assign what visually appears as a helix or a sheet, in a situation where the coordinate data have limited precision, is not a trivial algorithmic task. Another factor contributing to the difficulty is that quantum chemistry does not deliver a nice analytical expression for the strength of a hydrogen bond. In the prediction context it would be ideal not to focus solely on the visual, or topological, aspects of the assignment problem, but also to try to produce a more predictable assignment scheme. A reduced assignment scheme, which would leave out some of the helices and sheets and thereby make it possible to obtain close to perfect prediction, could be very useful, for example in tertiary structure prediction, which often uses a predicted secondary structure as starting point.

6.3.1 Secondary Structure Prediction Using MLPs

The basic architecture used in the early work of Qian and Sejnowski is a fully connected MLP with a single hidden layer [437]. The input window has an odd length W, with an optimal size typically of 13 amino acids. Orthogonal encoding is used for the input with an alphabet size $|\mathbf{A}| = 21$, corresponding to 20 amino acids and one terminator symbol to encode partial windows at the N- or C-terminal. Thus, the input layer has $13 \times 21 = 273$ units. The typical size of the hidden layer consists of 40 sigmoidal units. The total number of parameters of this architecture is then $273 \times 40 + 40 \times 3 + 40 + 3 = 11,083$. The output layer has three sigmoidal units, with orthogonal encoding of the alpha-helix, the beta-sheet, and the coil classes. The output represents the classification, into one of the three classes, of the residue located at the center of the input window. The classification is determined by the output unit with the greatest activity, an interpretation strategy known as the winner-take-all principle. This principle acts as an extra nonlinear feature in the relation between the input and the final output classification. Networks without hidden units will therefore, when interpreted by the winner-take-all principle, not be entirely linear. Another way to put it is that the internal representation in the hidden layer of the sequence input does not need to be perfectly linearly separable. As long as the distance to the separating hyperplane is smallest for the correct output unit, it does not matter that the input representation ends up slightly in the wrong decision region.

The networks are initialized using random uniform weights in the $[-0.3,0.3]$ interval, and subsequently trained using backpropagation with the LMS error function (note that a normalized exponential output layer with the relative entropy as error function would have been more appropriate). The typical size of a training set is roughly 20,000 residues extracted from the Brookhaven Protein Data Bank (PDB). Thus the ratio of parameters to examples is fairly high, larger than 0.5. Today many more protein structures have been solved experimentally, so that a similar database of secondary structure assignments will be much larger.

When training on protein sequences, a random presentation order of input windows across the training set is used to avoid performance oscillations associated with the use of contiguous windows. With this architecture, performance goes from a 33% chance level to 60%, after which overfitting begins. More precisely, the overall correct percentage is $Q_3 = 62.7\%$, with the correlation coefficients $C_\alpha = 0.35$, $C_\beta = 0.29$, and $C_c = 0.38$ [382]. As a consequence of the imbalance in the amount of helix, sheet, and coil in natural proteins (roughly found in proportions 0.3/0.2/0.5), mere percentages of correctly predicted window configurations can be bad indicators of the predictive performance. A much used alternative measure, which takes into account the relation between correctly predicted positives and negatives as well as false positives and negatives, is the correlation coefficient [382],

$$C_X = \frac{(P_X N_X) - (N_X^f P_X^f)}{\sqrt{(N_X + N_X^f)(N_X + P_X^f)(P_X + N_X^f)(P_X + P_X^f)}}, \tag{6.1}$$

where X can be any of the categories helix, sheet, coil, or two or more of these categories merged as one. P_X and N_X are the correctly predicted positives and negatives, and P_X^f and N_X^f are similarly the incorrectly predicted positives and negatives. A perfect prediction gives $C(X) = 1$, whereas a fully imperfect one gives $C(X) = -1$ (for a more detailed discussion of this and other performance measures, see section 6.7 below).

The authors conducted a number of experiments to test architectural and other variations and concluded that increasing the size of the input beyond 13 or adding additional information, such as amino acid hydrophobicities, does not lead to performance improvement. Likewise, no improvement appears to result from using finer secondary structure classification schemes, higher-order or recurrent networks, or pruning methods.

The main improvement is obtained by cascading the previous architecture with a second network that can take advantage of the analog certainty values present in the three output units and their correlations over adjacent positions. The second network also has an input window of length 13, corresponding to 13 successive outputs of the first network. Thus the input layer of the top network is 13×3. The top network also has a hidden layer with 40 units, and the usual 3 output units. With this cascaded architecture, the overall performance reaches $Q_3 = 64.3\%$, with the correlations $C_\alpha = 0.41$, $C_\beta = 0.31$, and $C_c = 0.41$. After training, the authors observed that the top network ultimately cleans up the output of the lower network, mostly by removing isolated assignments. From these and other results, it was concluded that there appears to be a theoretical limit of slightly above 70% performance for any "local" method, where "local" refers to the size of the input window to the prediction algorithm. In 1988 these overall results appeared to be much better than all previous methods, including the renowned Chou-Fasman method [129]. The subsequent growth in the data material has significantly increased the performance of more advanced NN approaches to this problem, but the increase has not caused a similar improvement in the performance of the Chou-Fasman method [549]. As can be seen below, several secondary structure prediction methods have now exceeded the level of 70% with a comfortable margin—some are even quite close to the level of 80%.

6.3.2 Prediction Based on Evolutionary Information and Amino Acid Composition

Most of the subsequent work on predicting secondary structure using NNs [78, 262, 323, 505, 451, 452, 290, 427] has been based on the architecture described above, sometimes in combination with other methods [582, 377] such as the Chou-Fasman rules [129].

In one interesting case the Chou-Fasman rules were used to initialize a network [377]. This knowledge-based network was born with a performance similar to the one obtained by encoding the rules directly into the weights. Experimental data from PDB could then be used to train extra free connections that had been added. All the exceptions in the relation between input sequence and conformational categories not covered by the rules would then be handled by the extra parameters adjusted by training. This network structure is also interesting because it allows for easy inspection of the weights, although it still performs only slightly better than the Qian–Sejnowski architecture. Compared to the Chou-Fasman rules, the performance was, as expected, greatly improved.

An evaluation of the MLP architecture in comparison with Bayesian methods has also been made [505]. In this work the Bayesian method makes the unphysical assumption that the probability of an amino acid occurring in each position in the protein is independent of the amino acids occurring elsewhere. Still, the predictive accuracy of the Bayesian method was found to be only minimally less than the accuracy of the neural networks previously constructed. A neural formalism in which the output neurons directly represent the conditional probabilities of structural classes was developed. The probabilistic formalism allows introduction of a new objective function, the mutual information, that translates the notion of correlation as a measure of predictive accuracy into a useful training measure. Although an accuracy similar to other approaches (utilizing a mean-square error) is achieved using this new measure, the accuracy on the training set is significantly higher, even though the number of adjustable parameters remains the same. The mutual information measure predicts a greater fraction of helix and sheet structures correctly than the mean-square-error measure, at the expense of coil accuracy.

Although tests made on different data sets can be hard to compare, the most significant performance improvement as compared to previous methods has been achieved by the work of Rost and Sander, which resulted in the PHD prediction server [451, 452, 453]. In the 1996 Asilomar competition CASP2 (Critical Assessment of Techniques for Protein Structure Prediction), this method performed much better than virtually all other methods for making predictions of secondary structure [161]. This unique experiment attempts to gauge the current state of the art in protein structure prediction by means of blind prediction. Sequences of a number of target proteins that are in the process of being solved are made available to predictors before the experimental structures are available. The PHD method reached a performance level of 74% on the unknown test set in the ab initio section of the competition, which contains contact, secondary structure, and molecular simulation predictions. This category is the most prestigious and inherently the most difficult prediction category, where the only prior knowledge is the primary structure in the amino acid sequence.

Prediction of secondary structure in a three-state classification based on single sequences seems to be limited to < 65–68% accuracy. In the mid-1980s, prediction accuracy reached 50–55% three-state accuracy, but more advanced neural network algorithms and increased data sets pushed the accuracy to the 65% level, a mark long taken as insurmountable. The key feature in the PHD approach, as well as in other even more powerful methods that have been constructed recently, has been to go beyond the local information contained in stretches of 13–21 consecutive residues by realizing that sequence families contain much more useful information than single sequences. Previously, this conclusion had also been reached in many studies using alignment of multiple sequences; see for example [587, 139, 60].

The use of evolutionary information improved the prediction accuracy to $> 72\%$, with correlation coefficients $C_\alpha = 0.64$ and $C_\beta = 0.53$. The way to use evolutionary information for prediction was the following. First, a database of known sequences was scanned by alignment methods for similar sequences. Second, the list of sequences found was filtered by a length-dependent threshold for significant sequence identity. Third, based on all probable 3D homologues, a profile of amino acid exchanges was compiled. Fourth, this profile was used for prediction.

The first method been proven in a cross-validation experiment based on 250 unique protein chains to predict secondary structure at a sustained level

DSSP	E	L	L	L	L	L	E	E	E	E	E	E	E	E	E	E	E	E	E	E	H	H	H
SH3	N	S	T	N	K	D	W	W	K	V	E	V	N	D	R	Q	G	F	V	P	A	A	Y
a1	N	K	S	N	P	D	W	W	E	G	E	L	N	G	Q	R	G	V	F	P	A	S	Y
a2	E	E	H	.	G	E	W	W	K	A	K	s	s	K	R	E	G	F	I	P	S	N	Y
a3	R	S	T	.	G	D	W	W	L	A	r	v	T	G	R	E	G	Y	V	P	S	N	F
a4	F	S	.	.	.	.	F	F	G	V	e	v	D	D	L	Q	V	F	V	P	P	A	Y
V	0	0	0	0	0	0	0	0	0	40	0	60	0	0	0	0	20	20	60	0	0	0	0
L	0	0	0	0	0	0	0	0	20	0	0	20	0	0	20	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	20	0	0	0	0	0	20	20	0	0	0	0	0	0	0	0	0	60	20	0	0	0	20
W	0	0	0	0	0	0	80	80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	80
G	0	0	0	0	50	0	0	0	20	20	0	0	0	40	0	0	80	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	40	40	0
P	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	20	0	0
S	0	60	25	0	0	0	0	0	0	0	0	20	20	0	0	0	0	0	0	0	40	20	0
T	0	0	50	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	20	0	0	0	0	0	0	0	0	0	20	0	0	0	60	20	0	0	0	0	0	0	0
K	0	20	0	0	25	0	0	0	40	0	20	0	0	20	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	40	0	0	0	0	0	0	0
E	20	20	0	0	0	25	0	0	20	0	60	0	0	0	0	40	0	0	0	0	0	0	0
N	40	0	0	100	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	40	0
D	0	0	0	0	0	75	0	0	0	0	0	0	20	40	0	0	0	0	0	0	0	0	0
N_{del}	0	0	1	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N_{ins}	0	0	0	0	0	0	0	0	0	0	2	3	1	0	0	0	0	0	0	0	0	0	0
CW	1.0	0.8	0.7	0.8	0.6	1.1	1.5	1.5	0.8	0.9	1.0	0.7	0.7	0.9	0.9	0.7	1.5	1.0	1.2	1.5	0.9	0.7	1.5

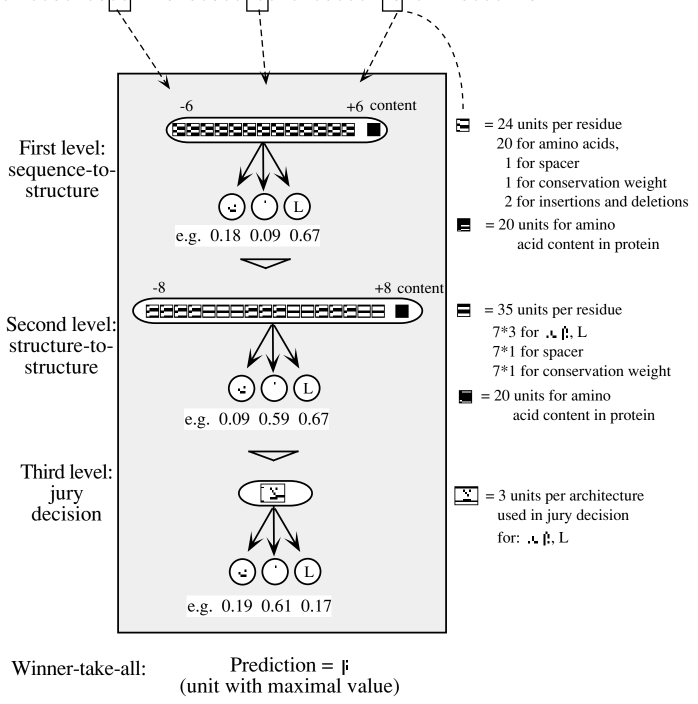

-6
+6 content
First level: sequence-to-structure
L
e.g. 0.18 0.09 0.67
= 24 units per residue
20 for amino acids,
1 for spacer
1 for conservation weight
2 for insertions and deletions
= 20 units for amino acid content in protein
-8
+8 content
Second level: structure-to-structure
L
e.g. 0.09 0.59 0.67
= 35 units per residue
7*3 for , L
7*1 for spacer
7*1 for conservation weight
= 20 units for amino acid content in protein
Third level: jury decision
L
e.g. 0.19 0.61 0.17
= 3 units per architecture used in jury decision
for: , L
Winner-take-all:
Prediction =
(unit with maximal value)

Figure 6.2: The PHD Architecture for Secondary Structure Prediction Developed by Rost and Sander. The input is based not on a conventional orthogonal encoding of the query sequence, but on a profile made from amino acid occurrences in columns of a multiple alignment of sequences with high similarity to the query sequence.

of > 72% three-state accuracy was the PHD neural network scheme [451, 452, 453]. For this method the profiles, along with additional information derived from the multiple sequence alignments and the amino acid content of the protein, were fed as input into a neural network system, as shown in figure 6.2. The input was based not on a conventional orthogonal encoding of a single sequence, but on a profile made from amino acid occurrences in columns of a multiple alignment of sequences with high similarity to the query sequence. In the example shown in figure 6.2 five sequences are included in the profile. The lowercase letters indicate deletions in the aligned sequence. To the resulting 20 values at one particular position in the protein (one column), three values are added: the number of deletions, the number of insertions, and a conservation weight. Thirteen adjacent columns are used as input. The “L” (loop) category is equivalent to the coil category in most other work. The whole network system for secondary structure prediction consists of three layers: two network layers and one layer averaging over independently trained networks.

In this work the profiles were taken from the HSSP database [471]. HSSP is a derived database merging structural and sequence information. For each protein of known 3D structure from PDB, the database has a multiple sequence alignment of all available homologues and a sequence profile characteristic of the family.

The backpropagation training of the networks was either *unbalanced* or *balanced*. In a large, low-similarity database of proteins the distribution over the conformational categories helix, sheet, and coil is, as indicated above, roughly 30%, 20%, and 50%, respectively. In unbalanced training the 13 amino acid-wide profile vectors were presented randomly with the same frequency. In the balanced version, the different categories were presented equally often. This means that the helix and sheet examples were presented about twice as often as the coil. In the final network system a mixture of networks trained by these two approaches was used. Networks trained by the balanced approach allow a much more reliable prediction of the sheet category.

Many other details of the architectures are important in yielding a prediction with a high overall accuracy, a much more accurate prediction of sheets than previously obtained, and a much better prediction of secondary structure segments rather than single residues. For 40% of all residues predicted with *high reliability*, the method reached a value of close to 90%, that is, was as accurate as homology modeling would be, if applicable. Almost 10 percentage points of the improvement in overall accuracy stemmed from using evolutionary information.

Clearly, one of the main dangers of the Qian-Sejnowski architecture is the overfitting problem. Rost and Sander started with the same basic architecture, but used two methods to address the overfitting problem. First, they used early stopping. Second, they used ensemble averages [237, 340] by train-

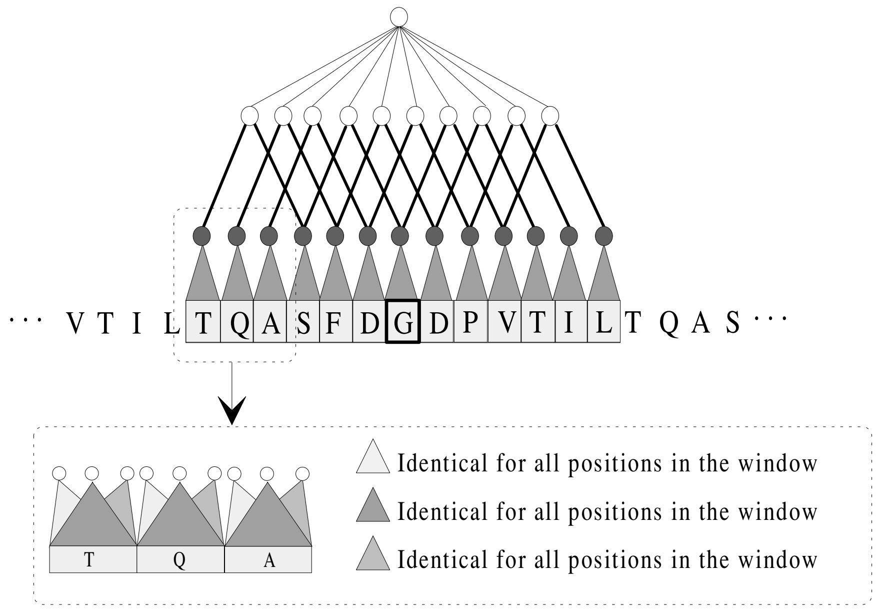

··· V T I L T Q A S F D G D P V T I L T Q A S ···
T
Q
A
Identical for all positions in the window
Identical for all positions in the window
Identical for all positions in the window

Figure 6.3: Riis and Krogh Network for Predicting Helices. The network uses the local encoding scheme and has a built-in period of three residues. Dark circles symbolize three hidden units, and heavy lines, three weights. In the lower part of the figure, shaded triangles symbolize 20 shared weights, and shaded rectangles, 20 input units. The network has a window size of 13 residues and has one output neuron.

ing different networks independently, using different input information and learning procedures. But the most significant new aspect of their work is the use of multiple alignments, in the sense that profiles (i.e. position-dependent frequency vectors derived from multiple alignments), rather than raw amino acid sequences, are used in the network input. The reasoning behind this is that multiple alignments contain more information about secondary structure than do single sequences, the secondary structure being considerably more conserved than the primary sequence.

6.3.3 Network Ensembles and Adaptive Encoding

Another interesting NN approach to the secondary structure prediction problem is the work of Riis and Krogh [338, 445], who address the overfitting problem by careful design of the NN architecture. Their approach has four main components. First, the main reason underlying the large number of parameters of the previous architectures is the large input layer (13×21). This number is greatly reduced by using an adaptive encoding of amino acids, that is, by letting the NN find an optimal and compressed representation of the input letters. This is achieved by encoding each amino acid using the analog values of M units, that is, with a local or distributed encoding. More precisely, the authors first use an orthogonal encoding with 20 units, the zero vector being used to represent the N- and C-terminal spacer symbols. Thus the input layer has size $W \times 20$. This input layer is connected to a first hidden layer of $M \times W$ units, but with a particular connectivity pattern. Each sequence position in the input layer is connected to a set of M sigmoidal units, and such connections are forced to be translation-invariant, that is, identical across sequence positions. This technique is also called weight-sharing in the NN literature. In an image-processing problem, the fixed set of connections would equivalently define the kernel of a convolution filter. The weight-sharing property is easily enforced during training by an obvious modification of the backpropagation algorithm, where weight updates are summed for weights sharing the same value. Thus each letter of the alphabet is encoded into the analog values of M units. In pattern-recognition problems, it is also common to think of the M units as feature detectors. Note that the features useful in solving the problems are discovered and optimized during learning, and not hardwired in advance. The number of free connections, including biases, between the full input layer and this representation layer is only $21 \times M$, regardless of the window size W. This leads to a great reduction from the over 10,000 parameters typically used in the first layer of the previous architectures. In their work, the authors use the values $M = 3$ and $W = 15$.

Second, Riis and Krogh design a different network for each of the three classes. In the case of alpha-helices, they exploit the helix periodicity by building a three-residue periodicity between the first and second hidden layers (see figure 6.3). The second hidden layer is fully interconnected to the output layer. In the case of beta-sheets and coils, the first hidden layer is fully interconnected to the second hidden layer, which has a typical size of 5–10 units. The second hidden layer is fully connected to the corresponding output unit. Thus a typical alpha-helix network has a total of 160 adjustable parameters, and a typical beta-sheet or coil network contains 300–500 adjustable parameters. The authors used balanced training sets, with the same number of positive and negative examples, when training these architectures in isolation.

Third, Riis and Krogh use *ensembles* of networks and filtering to improve the prediction. Specifically, they use five different networks for each type of structure at each position. The networks in each ensemble differ, for instance, in the number of hidden units used. The combining network takes a window of 15 consecutive single predictions. Thus the input layer to the combining network has size $15 \times 3 \times 5 = 225$ (figure 6.4). In order to keep the number of parameters within a reasonable range, the connectivity is restricted by having one hidden unit per position and per ensemble class (α, β, or coil). Thus the input is locally connected to a hidden layer with $3 \times 15 = 45$ units. Finally, the hidden layer is fully connected to three softmax (normalized exponentials) output units, computing the probability of membership in each class for the central residue. Consistent with the theory presented above, the error measure used is the negative log-likelihood, which in this case is the relative entropy between the true assignments and the predicted probabilities.

Finally, Riis and Krogh use multiple alignments together with a weighting scheme. Instead of profiles, for which the correlations between amino acids in the window are lost, predictions are made first from single sequences and are then combined using multiple alignments. This strategy is also used elsewhere [587, 457, 358], and can be applied to any method for secondary structure prediction from primary sequences, in combination with any alignment method. The final prediction is made by combining all the single-sequence predictions in a given column of the multiple alignment, using a weighting scheme. The weighting scheme used to compensate for database biases is the maximum-entropy weighting scheme [337]. The individual score in a given column can be combined by weighted average or weighted majority, depending on whether the averaging operates on the soft probability values produced by the single-sequence prediction algorithm, or on the corresponding hard decisions. One may expect soft averaging to perform better, since information is preserved until the very last decision; this is confirmed by the authors' observations, although the results of weighted average and weighted majority are similar. A small network with a single hidden layer of five units is then applied to filter the consensus secondary structure prediction derived, using the multiple alignment (see [445] for more detail). The small network also uses the fact that coil regions are less conserved and therefore have higher per-column entropy in a multiple alignment.

A number of experiments and tests are presented showing that (1) the architecture, with its local encoding, avoids the overfitting problem; (2) the performance is not improved by using a number of additional inputs, such as the normalized length of the protein or its average amino acid composition; (3) the improvement resulting from each algorithmic component is quantified—for instance, multiple alignments lead to roughly a 5% overall improvement, mostly associated with improvement in the prediction of the more conserved α and β structures; (4) the network outputs can be interpreted as classification probabilities. Most important, perhaps, the basic accuracy achieved is $Q_3 = 66.3\%$ when using sevenfold cross-validation on the same database of 126 nonhomologous proteins used by Rost and Sander. In combination with multiple alignments, the method reaches an overall accuracy of $Q_3 = 71.3\%$, with correlation coefficients $C_\alpha = 0.59$, $C_\beta = 0.50$, and $C_c = 0.41$. Thus, in

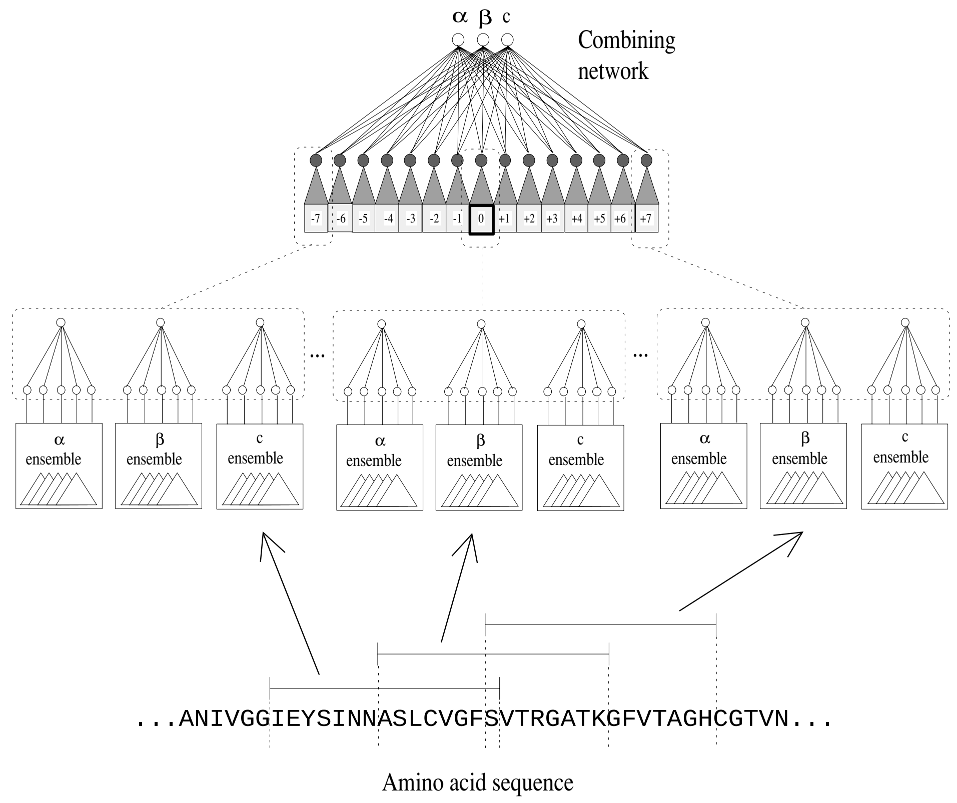


Figure 6.4: The Ensemble Method in the Riis and Krogh Prediction Scheme for Combining and Filtering Ensembles of Networks. The combining network (top of figure) takes a window of $3 \times 5 \times 15$ predictions from the ensembles of the dedicated secondary structures. In the combining network, the ensembles for each of the three structures are weighted separately by position-specific weights for each window position.

spite of a considerable amount of architectural design, the final performance is practically identical to [453]. This of course adds evidence to the consensus of an accuracy upper bound slightly above 70-75% on any prediction method based on local information only.

6.3.4 Secondary Structure Prediction Based on Profiles Made by Position-Specific Scoring Matrices

The key contribution of the PHD method was the use of sequence profiles that contain more structural information for extraction by the neural network. Profiles are based on sequences identified by alignment, and the profile quality obviously depends on the alignment approach used to select the sequences behind the profile.

The PSI-BLAST method [12] is an iterative approach where the sequences found in an initial scan of a database (typically Swiss-Prot) based on a single sequence are used to generate a new search profile, which in turn is used to pick up additional sequences. This type of "sequence walking" will normally reach more family members, even if it is also associated with risk of picking up unrelated sequences, weakening the structurally conserved, family-specific bias in the profile.

In the PSIPRED method [290, 386] Jones used this iterative approach as a clever way of generating profiles for use as improved input to the network scheme. These profiles were based on so-called *position-specific* scoring matrices and did significantly increase the predictive power of the neural network. When obtaining the profiles, the initial database scan was performed using the Blosum62 substitution matrix, while in subsequent scans substitution scores were calculated from the multiple alignment position by position.

Replacing the HSSP profiles used in the PHD method by this more sophisticated approach led to an increase in predictive performance of several percentage points, up to 76.5% for the prediction based on three categories, helix (DSSP H/G/I), sheet (DSSP E/B) and coil. If the G and I helix categories are included in the coil category, the percentage increases further to 78.3%. Thus, depending on the precise definition of observed secondary structure, the overall percent correct varies 1-2%. This variation is essentially the same for most of the neural network methods, and is in fact observed also for other methods as well. In the 1998 Asilomar CASP3 competition (Critical Assessment of Techniques for Protein Structure Prediction), the PSIPRED method was indeed the best for secondary structure, reaching a performance of 77% on one set of sequences and 73% on a subset of diffucult targets [324], which is comparable to the level reported for a larger set of test sequences consisting of 187 unique folds.

6.3.5 Prediction by Averaging over 800 Different Networks

Although helices and sheets have preferred lengths, as observed in wildtype proteins, the length distributions for both types of structure are quite broad. If only a single neural network is used to provide the prediction, the window size will have to be selected as a compromise that can best find the transition from coil to noncoil and from noncoil to coil as measured on a large data set. However, larger windows can benefit from the additional signal in long secondary structures, while short windows often will perform better on structures of minimal length that do not overlap with the previous or next secondary structures, and so on. As single networks, the large and small windows will perform worse, but in individual cases they will typically be more confident, i.e., output values are closer to the saturated values of zero and one.

When combining many different networks, the critical issue is therefore how to benefit from those networks that overall are suboptimal, but in fact are more reliable on a smaller part of the data. When the number of networks becomes large, simple averaging will make the noise from the suboptimal networks become very destructive—in one case an upper limit for the productive number of networks to be combined has been suggested to be around eight [118]. In this work, approximate performance values for the networks were at the level of 73.63% (one network), 74.70% (two), 74.73% (four), and 74.76% (eight) for a three-category prediction scheme.

However, Petersen and coworkers showed recently [427] that it is possible to benefit from as many as 800 networks in an ensemble with strong architectural diversity: different window sizes, different numbers of hidden units, etc. The key element in the procedure is to identify, from the 800 networks, those predictions that are likely to be of high confidence for a given amino acid residue in the test data. By averaging over the highly confident predictions only, it becomes possible to exploit many networks and prevent the noise from the suboptimal networks from eradicating the signal from the confident true positive (and true negative) predictions.

Using this scheme, it was possible to increase the prediction level above what could be obtained with the PSIPRED method. When measured on the two different ways of merging the DSSP categories into categories of helix, sheet, and coil, the improvement ranged from 77.2% (standard merging) to 80.2% as measured as the mean at the per-amino acid level. The percentages are slightly higher when reported as mean per-chain (77.9%-80.6%).

Output Expansion

In this study the performance was improved by introducing another new feature in the output layer. The Petersen scheme was designed to incorporate so-called *output expansion* where the networks provide a prediction not only for the secondary structure category corresponding to a single (central) amino acid in the input window, but simultaneous predictions for the neighboring residues as well.

This idea is related to earlier ideas of constructing networks by training using *hints* that essentially further constrain the network weights, thereby leading to improved generalization.

Networks that are trained to predict currency exchange rates, e.g. dollar versus yen, may be improved if they also are forced to predict the American budget deficit, or similar features that are somehow related to the original output [1]. This idea must also be formulated as the learning-of-many-related-tasks-at-the-same-time approach, or multitask learning [115]. A network learning many related tasks at the same time can use these tasks as inductive bias for one another and thus learn better.

In protein secondary structure, it is certainly true that the conformational categories for the adjacent residues represent information that is correlated to the category one wants to predict. Other hints could be, for example, the surface exposure of the residue (as calculated from the structure in PDB), or the hydrophobicity as taken from a particular hydrophobicity scale.

6.4 Prediction of Signal Peptides and Their Cleavage Sites

Signal peptides control the entry of virtually all proteins to the secretory pathway in both eukaryotes and prokaryotes [542, 207, 440]. They comprise the N–terminal part of the amino acid chain, and are cleaved off while the protein is translocated through the membrane.

Strong interest in automated identification of signal peptides and prediction of their cleavage sites has been evoked not only by the huge amount of unprocessed data available but also by the commercial need for more effective vehicles for production of proteins in recombinant systems. The mechanism for targeting a protein to the secretory pathway is believed to be similar in all organisms and for many different kinds of proteins [296]. But the identification problem is to some extent organism-specific, and NN-based prediction methods have therefore been most successful when Gram-positive and Gram-negative bacteria, and eukaryotes have been treated separately [404, 131]. Signal peptides from different proteins do not share a strict consensus sequence—in fact, the sequence similarity between them is rather low. However, they do share a common structure with a central stretch of 7–15 hydrophobic amino acids (the hydrophobic core), an often positively charged region in the N-terminal of the preprotein, and three to seven polar, but mostly uncharged, amino acids just before the cleavage site.

This (and many other sequence analysis problems involving "sites") can be tackled from two independent angles: either by prediction of the site itself or by classifying the amino acids in the two types of regions into two different categories. Here this would mean classifying all amino acids in the sequence as cleavage sites or noncleavage sites; since most signal peptides are below 40 amino acids in length, it makes sense to include only the first 60–80 amino acids in the analysis. Alternatively, the amino acids could be classified as belonging to the signal sequence or the mature protein. In the approach described below, the two strategies have been combined and found to contribute complementary information. While the prediction of functional sites often is fairly local and therefore works best using small windows, larger windows are often needed to obtain good prediction of regional functional assignment.

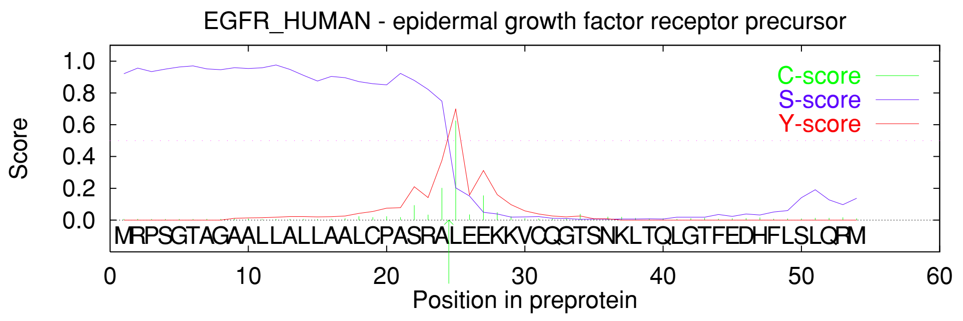
6.4.1 SignalP

In the SignalP prediction scheme [404], two types of networks provide different scores between 0 and 1 for each amino acid in a sequence. The output from the signal peptide/nonsignal peptide networks, the *S-score*, can be interpreted as an estimate of the probability of the position's belonging to the signal peptide, while the output from the cleavage site/noncleavage site networks, the *C-score*, can be interpreted as an estimate of the probability of the position's being the first in the mature protein (position +1 relative to the cleavage site).

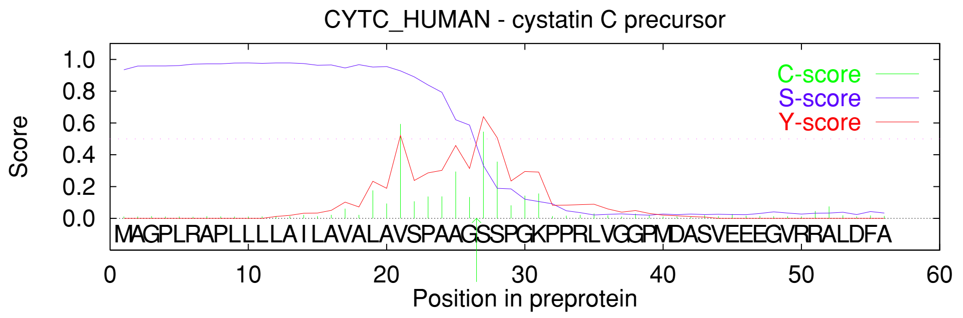
In figure 6.5, two examples of the values of C- and S-scores for signal peptides are shown. A typical signal peptide with a typical cleavage site will yield curves like those shown in figure 6.5A, where the C-score has one sharp peak that corresponds to an abrupt change in S-score. In other words, the example has 100% correctly predicted positions, according to both C-score and S-score. Less typical examples may look like figure 6.5B, where the C-score has several peaks.

In this work the data were divided into five subsets, and five independent networks were selected based on cross-validation for each task (and for each organism class). The individual C- and S-scores were therefore obtained by averaging over these five networks. In the final implementation for the three classes of organisms, 15 networks are included for each score. The work on signal peptide prediction provides another example of the importance of post-processing of the network outputs, and of how "intelligent" interpretation can significantly improve the overall performance.

The C-score problem was best solved by networks with asymmetric windows, that is, windows including more positions upstream than downstream of the cleavage site: 15 and 2–4 amino acids, respectively. This corresponds well with the location of the cleavage site pattern information when viewed as a signal peptide sequence logo [404]. The S-score problem, on the other hand, was overall best solved by symmetric windows, which not surprisingly are better at identifying the contrast between the compositional differences of signal peptides and of mature protein. For human and *E. coli* sequences, these



A



B

Figure 6.5: Examples of Predictions for Sequences with Verified Cleavable Signal Peptides. The values of the C-score (output from cleavage site networks), S-score (output from signal peptide networks), and Y-score (combined cleavage site score, $Y_i = \sqrt{C_i \Delta_d S_i}$) are shown for each position in the sequences. The C- and S-scores are averages over five networks trained on different parts of the data. The C-score is trained to be high for the position immediately *after* the cleavage site, that is, the first position in the mature protein. The true cleavage sites are marked with arrows. **A** is a sequence with all positions correctly predicted according to both C-score and S-score. **B** has two positions with C-score higher than 0.5—the true cleavage site would be incorrectly predicted when relying on the maximal value of the C-score alone, but the combined Y-score is able to predict it correctly.

windows were larger: 27 and 39 amino acids, respectively.

Since the sequences in most cases have only one cleavage site, it is not necessary to use as assignment criterion a fixed cutoff of, say, 0.5 when interpreting the C-score for single positions. The C-score networks may also be evaluated at the *sequence* level by assigning the cleavage site of each signal peptide to the position in the sequence with the maximal C-score and calculating the percentage of sequences with the cleavage site correctly predicted by this assignment. This is how the performance of the earlier weight matrix method [539] was calculated. Evaluating the network output at the sequence level improved the performance; even when the C-score had no peaks or several peaks above the cutoff value, the true cleavage site was often found at the position where the C-score was highest.

If there are several C-score peaks of comparable strength, the true cleavage site may often be found by inspecting the S-score curve in order to see which of the C-score peaks coincides best with the transition from the signal peptide to the nonsignal peptide region. The best way of combining the scores turned out to be a simple geometric average of the C-score and a smoothed derivative of the S-score. This combined measure has been termed the *Y-score*:

$$Y_i = \sqrt{C_i \Delta_d S_i}, \tag{6.2}$$

where $\Delta_d S_i$ is the difference between the average S-score of d positions before and d positions after position i:

$$\Delta_d S_i = \frac{1}{d}\left(\sum_{j=1}^{d} S_{i-j} - \sum_{j=0}^{d-1} S_{i+j}\right). \tag{6.3}$$

The Y-score gives a certain improvement in sequence level performance (percent correct) relative to the C-score, but the single-position performance (C_C) is not improved. An example in which the C-score alone gives a wrong prediction while the Y-score is correct is shown in figure 6.5B.

It is interesting that this method also can be used for detection of seemingly wrong assignments of the initiation methionine. Inspection of a number of long signal peptides deposited in SWISS-PROT has shown that such sequences often contain a second methionine 5–15 amino acids from the annotated N-terminus [422]. Figure 6.6 shows a SignalP prediction for the sequence of human angiotensinogen. In the N-terminal, this sequence has a surprisingly low S-score, but after the second methionine in the sequence it increases to a more reasonable level. The prediction strongly indicates that the translation initiation has been wrongly assigned for this sequence.

6.5 Applications for DNA and RNA Nucleotide Sequences

6.5.1 The Structure and Origin of the Genetic Code

Since the genetic code was first elucidated [407], numerous attempts have been made to unravel its potential underlying symmetries [216, 6, 509, 514, 125] and evolutionary history [168, 563, 294, 569]. The properties of the 20 amino acids and the similarities among them have played a key role in this type of

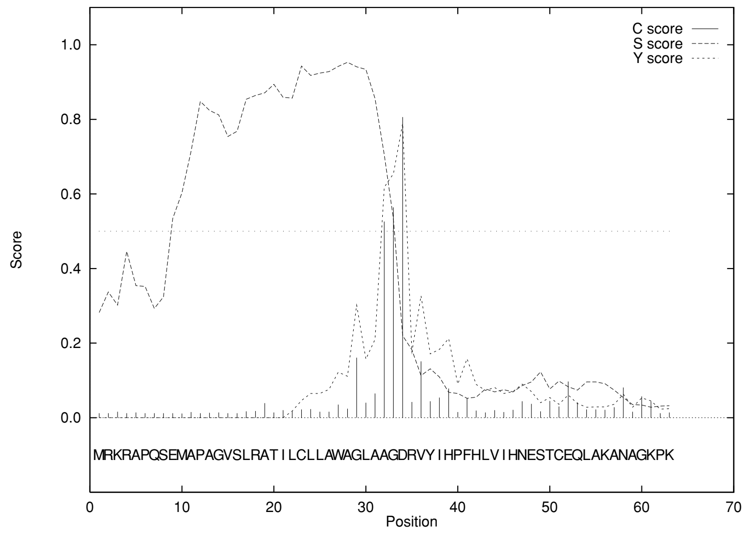


Figure 6.6: SignalP Prediction of the Sequence of ANGT_HUMAN, human angiotensinogen. The S-score (signal peptide score) has a high value for residues within signal peptides, while the C- and Y-scores (cleavage site scores) are high at position +1 immediately after possible cleavage sites. Note that the S-score is comparatively low for the region between the first Met and the second Met.

analysis. The codon assignments are correlated to the physical properties of the amino acids in a systematic and error-correcting manner. The three positions in the triplets associate to widely different features of the amino acids. The first codon position is correlated to amino acid biosynthetic pathways [569, 514], and to their evolution evaluated by synthetic "primordial soup" experiments [159, 478]. The second position is correlated to the hydrophatic properties of the amino acids [140, 566], and the degeneracy of the third position is related to the molecular weight or size of the amino acids [240, 514]. These features are used for error correction in two ways. First, the degeneration is correlated to the abundance of the amino acids in proteins, which lowers the chance that a mutation changes the amino acid [371]. Second, similar amino acids have similar codons, which lowers the chance that a mutation has a hazardous effect on the resulting protein structure [140, 216, 6].

In the neural network approach to studying the structure of the genetic code, the analysis is new and special in that it is unbiased and completely data-driven [524]. The neural network infers the structure directly from the mapping between codons and amino acids as it is given in the standard genetic code (figure 6.7). Hence, no a priori relationships are introduced between the

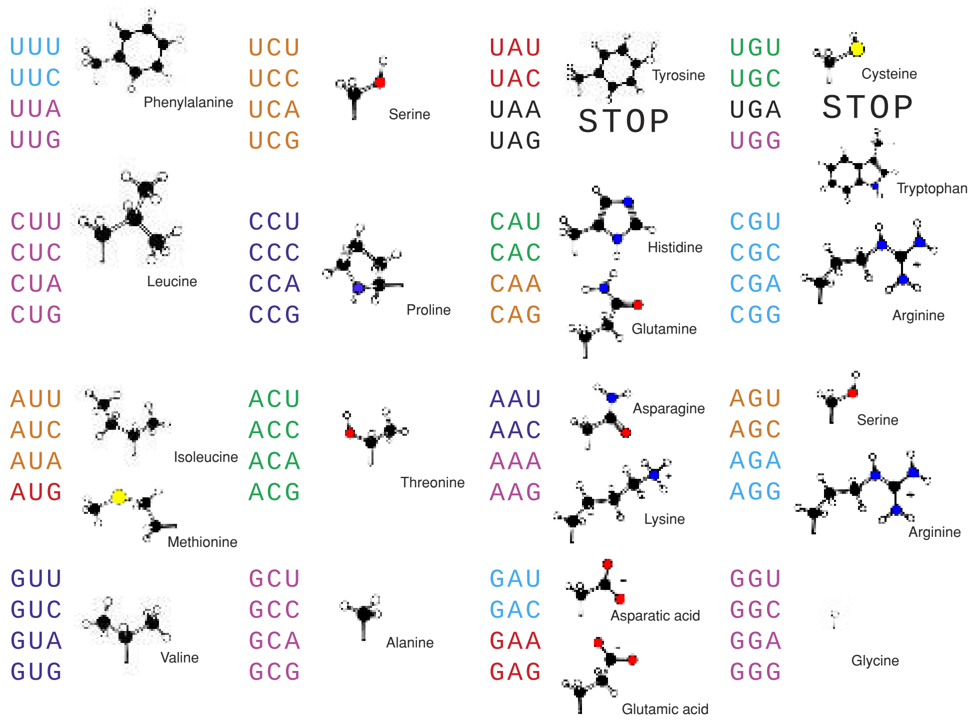


Figure 6.7: The Standard Genetic Code. Triplets encoding the same amino acid are shown in the same shade of gray.

nucleotides or amino acids.

In the network that learns the genetic code, the input layer receives a nucleotide triplet and outputs the corresponding amino acid. Thus the 64 different triplets are possible as input, and in the output layer the 20 different amino acids appear (see figure 6.8). Inputs and outputs are sparsely encoded; 12 units encode the input and 20 units encode the output.

Networks with three and four intermediate units were relatively easy to train; it was harder to obtain perfect networks with two intermediate units. The only way minimal networks (with two intermediates) could be found was by an adaptive training procedure [524]. For this task, at least, it was observed that the conventional backpropagation training scheme treating all training examples equally fails to find a minimal network, which is known to exist.

The standard technique for training feed-forward architectures is backpropagation; the aim is to get a low analog network error $\mathcal{E}$ but not necessarily

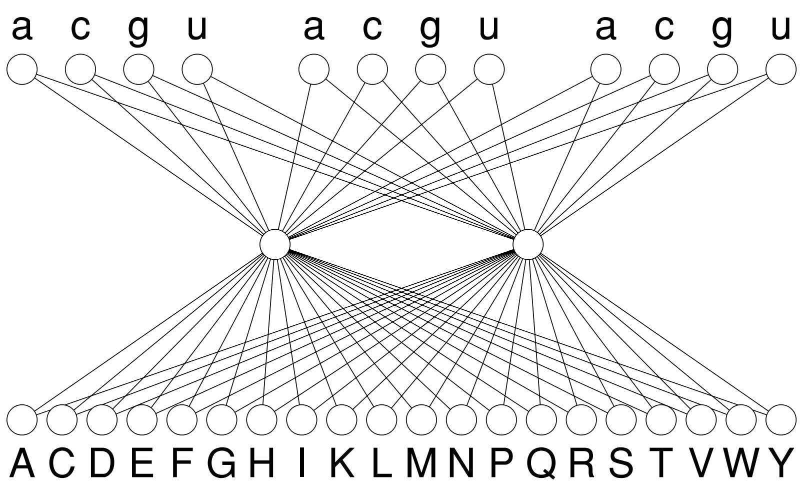


Figure 6.8: Architecture of the Neural Network Trained to Learn the Standard Genetic Code. The network had 12 input units, two (or more) intermediate units, and 20 output units. The input layer encoded the nucleotide triplets as a binary string comprising three blocks of four bits, with adenine as 0001, cytosine as 0010, guanine as 0100, and uracil as 1000. The output layer encoded the amino acids with alanine as 10000000000000000000, cysteine as 01000000000000000000, ... Intermediate and output units had real-valued activities in the range between 0.0 and 1.0. The network parameters ($12 \cdot 2 + 2 \cdot 20 = 64$ weights and $2 + 20 = 22$ thresholds) were adjusted using the backpropagation algorithm [456] in a balanced form, where for each codon the training cycle was repeated in inverse proportion to the number of codons associated with the amino acid. Thus each of the leucine codons received on the average six times less training than the single methionine codon. During training, the criterion for successful learning was that the activity of the corresponding output unit should be larger than all others (the winner-take-all principle). In each training epoch the codons were presented to the network in random order.

a low classification error $\mathcal{E}_C$. This often makes it difficult to train networks down to $\mathcal{E}_C = 0.0$. In the literature, several training strategies have been pursued in order to obtain low classification errors. First, a simple but effective modification is to use a high learning rate for wrongly classified examples and a low learning rate for correctly classified examples. In the first phase of training, most examples are wrongly classified. This results in a high learning rate and therefore a fast decrease in network error $\mathcal{E}$. Later in training only the hard cases have a high learning rate. Thereby noise is introduced and jumps in the network error level to lower plateaus are favored.

Second, another effective procedure is to modify the presentation frequencies for the different categories so that a more balanced situation results. In the case of the genetic code this means that the same number of codons should be presented for each amino acid, no matter how many appear in the original code. Thus the single methionine codon should be included in the set six times and each cysteine codon should appear three times. The training set is then enlarged from 61 to 186 codons, tripling the training time for each epoch.

Third, an even more powerful strategy for obtaining a low classification error will be to have an adaptive training set, where the training examples are included or excluded after determining whether they are classified correctly by the current network. Such a scheme may introduce more noise in the learning process, which helps to avoid local minima. Introducing noise in the training is usually done by updating the network after each example rather than after each epoch. The next step is to shuffle the examples within each epoch prior to presentation. Using the adaptive procedure makes the epoch concept disappear, and each example is chosen at random from a pool with subsequent updating of the network. To increase the frequency of a hard-to-learn example, each example misclassified is put into the pool of examples, replacing one in the pool. To ensure that no examples are lost, only part of the pool is open for exchange. In the long run this procedure ensures that every example is shown and hard-to-learn examples are shown more often. In summary, the procedure is as follows:

1. Initialize the first and second parts of the pool with the training examples.
2. Choose an example randomly from the pool and present it to the network.
3. Train the network by backpropagation.
4. If the example is classified correctly, then go to 2.
5. If the example is misclassified, put it in the second part of the pool, thus replacing a randomly chosen example.
6. Repeat until $\mathcal{E}_C = 0$.

A network with two hidden units was successfully trained using this adaptive training scheme. During training, the network develops an internal representation of the genetic code mapping. The internal representation of the structure of the code is given by the activities of the two intermediate units, which may easily be visualized in the plane. The network transforms the 61 12-component vectors representing the amino acid encoding codons into 61 points in the plane, which, provided the network has learned the code, may be separated linearly by the 20 output units.

Figure 6.9 shows how the network arrives at the internal representation by adaptive backpropagation training. Each codon is mapped into a point (x, y)

in the plane indicated by the one-letter abbreviation for the matching amino acid. During training, the 61 points follow trajectories emerging from their prelearning positions close to the center ($x \approx 0.5, y \approx 0.5$) and ending at their final locations on the edge of a slightly distorted circular region.

The network identifies three groups of codons corresponding to three parts of the circular region (figure 6.9). Subsequently it was discovered that these groups divide the GES scale of transfer free energies [166] into three energy intervals, $[-3.7 : -2.6]$, $[-2.0 : 0.2]$, and $[0.7 : 12.3]$ (kcal/mol), respectively (see table 6.2). The only case that does not conform to the network grouping is the extremely hydrophilic amino acid arginine, which is known to be an exception in the context of the genetic code [319, 509, 514]. The number of arginine codons is in conflict with its abundance in naturally occurring proteins [470]. Arginine has been suggested as a late addition to the genetic code [294]. In alpha-helices it has a surprising tendency to be at the same side as hydrophobic residues [136]. The network locates arginine in the intermediate group. The trained network maps the three stop codons to (x, y) positions in the vicinity of adjacent codons in the code: UAA, UAG close to Tyr (Y), and UGA close to Trp (W) (points not shown).

The fact that the network needs at least two intermediate units to learn the genetic code mapping means that the code is inherently nonlinear. In classification terms this means that the genetic code is nonlinearly separable. This holds true for the (otherwise sensible) sparse encoding of the nucleotides used by most workers. Computerized analysis of DNA or pre-mRNA striving to relate patterns in the nucleotides to amino acids [100, 102] will therefore be a nonlinear problem regardless of the algorithm applied. It is quite easy to prove that the genetic code is indeed nonlinear, since all serine codons cannot be separated linearly from the other codons.

The weights of the trained network have, unlike many other neural networks, a fairly comprehensible structure (figure 6.10). The size of the weights connecting the input units to the intermediates reflects the importance of particular nucleotides at specific codon positions. Interestingly, the second codon position has by far the largest weights, followed by the first and third positions, in agreement with earlier observations [424]. The two intermediate units to a large extent share the discrimination tasks between them; the unit to the left is strongly influenced by A or G at the second codon position, and the unit to the right, by C or U. At the first codon position A and C, and G and U, influence the two units, respectively. In the genetic code C and U are equivalent at the third codon position for all amino acids, and similarly for A and G with the exception of three amino acids (Ile, Met, and Trp). The network handles this equivalence by having positive and negative weights at the third codon position for the two pairs.

The rationale behind the correlation between the second position and the

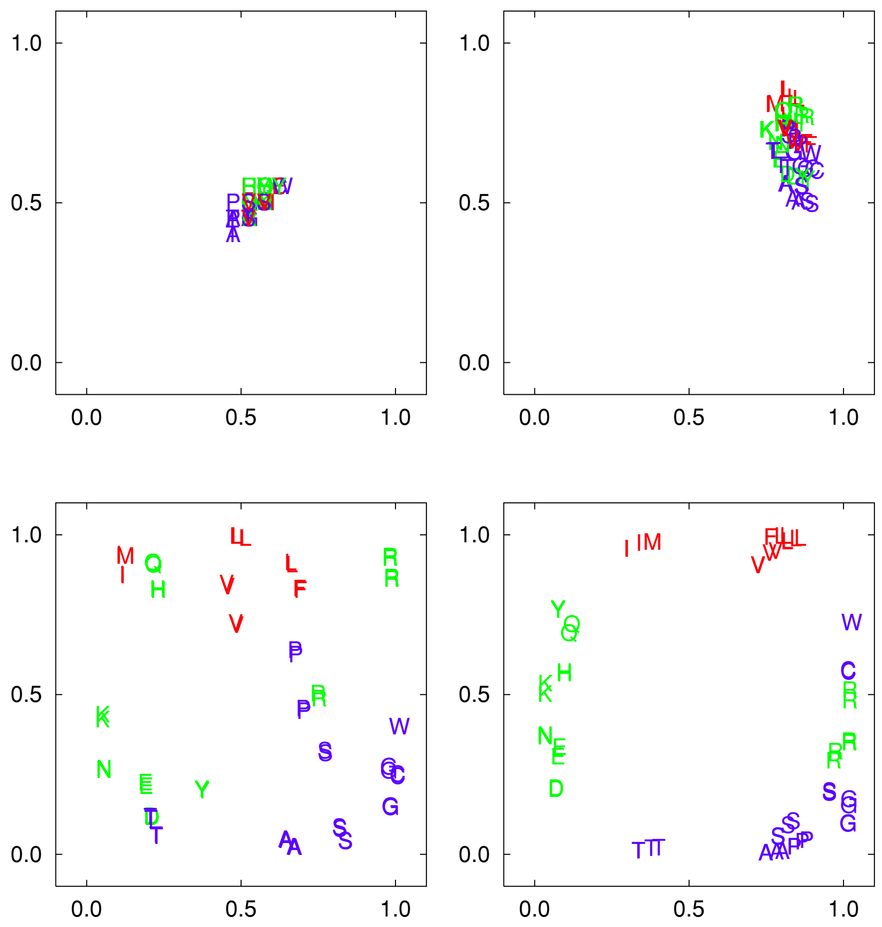


Figure 6.9: Hidden Unit Activities in the Genetic Code Neural Network. Each plot shows the two real-valued activities for all 61 amino acid encoding triplets in the code. In the untrained network with randomly assigned weights, all 61 points are located near the center of the square; after seven epochs the points have moved into a transient local minimum, where the activities of the intermediate units are close to 1 and the activities of all the output units are close to 0; at 30 epochs the groups have started to segregate but are still mixed; finally, at 13,000 epochs the network positions the 61 codons in groups on the edge of the circular region. After the four epochs shown, the number of correctly classified codons was 2, 6, 26, and 61, respectively.

Amino acid	Water-oil	Codons					
Phe	−3.7	UUU	UUC				
Met	−3.4	AUG					
Ile	−3.1	AUU	AUC	AUA			
Leu	−2.8	UUA	UUG	CUU	CUC	CUA	CUG
Val	−2.6	GUU	GUC	GUA	GUG		
Cys	−2.0	UGU	UGC				
Trp	−1.9	UGG					
Ala	−1.6	GCU	GCC	GCA	GCG		
Thr	−1.2	ACU	ACC	ACA	ACG		
Gly	−1.0	GGU	GGC	GGA	GGG		
Ser	−0.6	UCU	UCC	UCA	UCG	AGU	AGC
Pro	0.2	CCU	CCC	CCA	CCG		
Tyr	0.7	UAU	UAC				
His	3.0	CAU	CAC				
Gln	4.1	CAA	CAG				
Asn	4.8	AAU	AAC				
Glu	8.2	GAA	GAG				
Lys	8.8	AAA	AAG				
Asp	9.2	GAU	GAC				
Arg	12.3	CGU	CGC	CGA	CGG	AGA	AGG

Table 6.2: The amino acids and their transfer free energies in kcal/mol as given by the GES scale [166]. The GES scale separates codons with uracil and adenine at the second codon position from each other, leaving as an intermediate class amino acids with cytosine and guanine. The transfer free energy values are computed by considering a hydrophobic term based on the surface area of the groups involved, and two hydrophilic terms accounting for polar contributions arising from hydrogen bond interactions and the energy required to convert the charged side chains to neutral species at pH 7.

hydrophobicity of the amino acids may, in addition to the obvious advantage of minimizing the likelihood of mutation or mistranslation events changing a hydrophobic amino acid into a hydrophilic one [538, 409, 87], be more fundamental. In the early version of the genetic code, classes of codons coded for classes of amino acids [562]. Mostly these classes were purely related to the problem of folding a polypeptide chain in an aqueous environment. Lipid membranes, which may be phylogenetically older than the cytoplasm [316, 73, 117, 76], have not played a major role in the literature on the early protein synthesis apparatus. The problem of understanding the origin of cells is often dismissed by stating that “somehow” primitive ribosomes and genes became enclosed by a lipid membrane [117]. In scenarios described by Blobel and Cavalier-Smith [73, 117], genes and ribosomes associated with the surface of liposome-like vesicles where a mechanism for the cotranslational in-

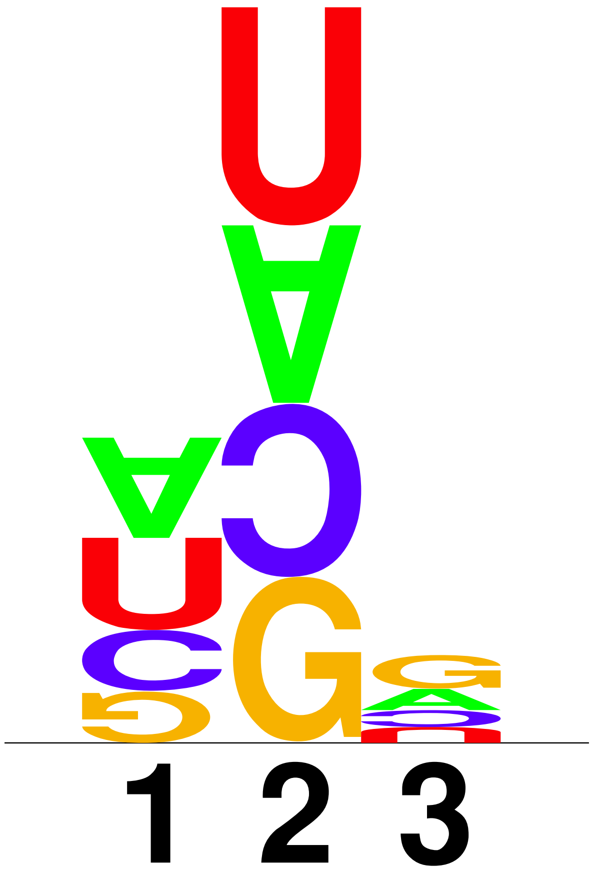


Figure 6.10: A Graphical Representation of the Input Unit Weights in the Trained Genetic Code Network. For each of the three codon positions the height of the letters indicates the size of the sum of the two weights connecting an input unit to the two intermediate units. If the sum is negative, the letters are upside down.

sertion of membrane proteins evolved. A segregation into genetic code classes founded on the amino acid properties in their relation to lipid environments may therefore also have been a basic necessity.

6.5.2 Eukaryotic Gene Finding and Intron Splice Site Prediction

Since the beginning of the 1980s a highly diverse set of methods has been developed for the problem of identifying protein-coding regions in newly sequenced eukaryotic DNA. Correct exon assignments may in principle be obtained by two independent approaches: prediction of the location of the alternating sequence of donor and acceptor sites, or classification of nucleotides—or continuous segments of nucleotides—as belonging to either the coding or the noncoding category.

Intron splice sites have a relatively confined local pattern spanning the range of 15–60 nucleotides; protein-coding regions (exons) are often much larger, having typical lengths of 100–150 nucleotides, an interval that is quite stable across a wide range of eukaryotic species. For both types of objects, the pattern strength or regularity is the major factor influencing the potential accuracy of their detection.

Some intron splice site sequences are very close to the "center of gravity" in a sequence space [344], while others deviate considerably from the consensus pattern normally described in textbooks on the subject (see Figure 1.10 for a sequence logo of donor sites from the plant *Arabidopsis thaliana*). Likewise, exon sequence may conform strongly or weakly to the prevailing reading frame pattern in a particular organism. The strength of the coding region pattern may be correlated, for example, to the gene expression level or the amino acid composition of the protein. The codon usage specific to a given organism and a given gene in most cases creates a fairly strong 3-periodicity with biased frequencies at the three codon positions [525, 305]. In some organisms, such as bacteria, the bias is largest on the first position, while in mammals the bias is strongest on the third position (see figure 6.11). Proteins rich in proline, serine, and arginine residues will often be associated with bad reading frames because they have codons that deviate from the prevailing choices on the first and second codon positions. However, in the context of the mRNA translation by the ribosome, the strength of a reading frame should be quantified by inspecting the three different possibilities, not just the average codon usage statistics [525]. Figure 6.11 shows the overall bias in the nucleotide distribution on the three codon positions in triplets from coding regions in genes from *Enterobacteria*, mammals, *Caenorhabditis elegans*, and the plant *A. thaliana*, respectively.

In a study using neural networks for the prediction of intron splice sites, it was observed [102] that there is a kind of compensating relationship between the strength of the donor and acceptor site pattern and the strength of the pattern present in the associated coding region. Easily detectable exons may allow weaker splice sites, and vice versa. In particular, very short exons, which usually carry a weak signal as coding regions, are associated with strong splice

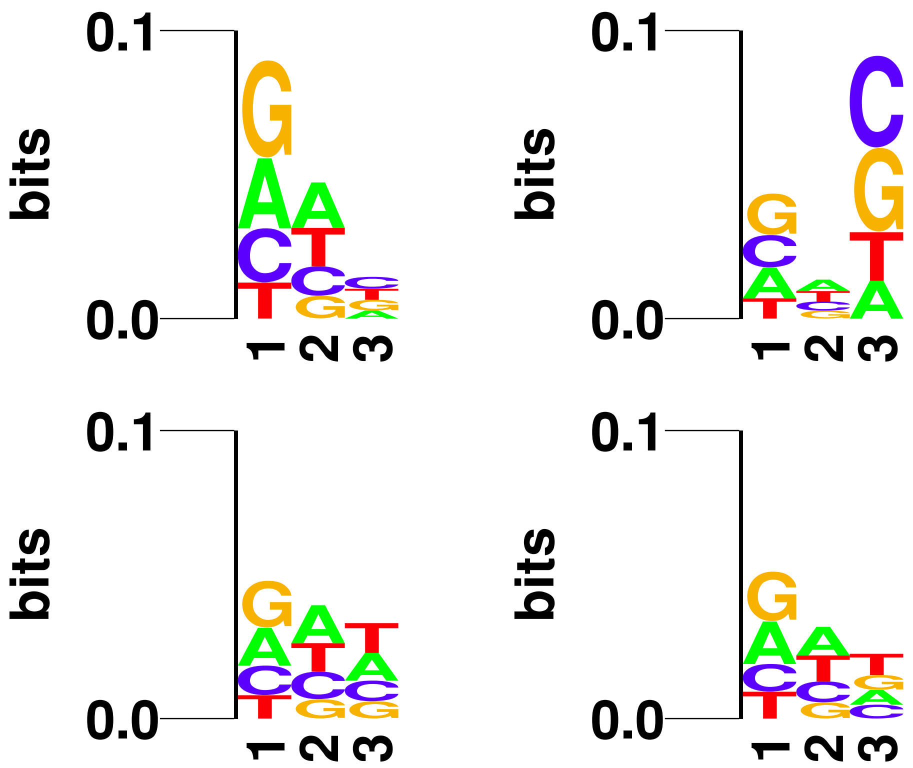


Figure 6.11: Sequence Logos of Codons from Four Different Types of Organisms: (top) Enterobacteria and Mammals; (bottom) *C. elegans* and *A. thaliana.* While bacterial genes have a strong bias on the first codon position, the bias is strongest on the third codon position in mammals.

sites. This relation is also moderated by the distribution for the intron length, which varies considerably from organism to organism.

The correlation between splice site *strength* and "exonness" has been exploited in the artificial neural network-based prediction scheme known as NetGene [102], where two local splice site networks are used jointly with an exon prediction network equipped with a large window of 301 nucleotides. This scheme considerably reduces the number of false positive predictions and, at the same time, enhances the detection of weak splice sites by lowering the prediction threshold when the signal from the exon prediction network is sharp in the transition region between coding and noncoding sequence segments (see section 6.5.4).

6.5.3 Examples of Gene Structure Prediction by Sensor Integration

The use of a combination of sensors for detection of various signals related to a complex object has a long history in the theory of pattern recognition. Several schemes have been developed in which NN components play a major role, earliest the GRAIL and GeneParser systems.

The GRAIL system is an NN-based example of sensor integration used for coding-region recognition. The first network, from 1991, combined into a joint prediction seven measures known to provide a nontrivial indication of the presence of a coding region [528]. The later GRAIL II coding system considers discrete coding-region candidates, rather than using a fixed-size sliding window for evaluation of the coding potential, as in the earlier system [529]. As one of the input features, the network is provided with a measure of the length of the coding-region candidate, and can therefore correlate the other measures specifically to known differences between short and long exons.

The performance of the GRAIL system has evolved over the years primarily by the development of more complex sensor indicators, and not by more sophisticated neural networks. The MLP with one hidden layer trained by backpropagation remains the same. Among the most advanced indicators is a fifth-order nonhomogeneous Markov chain for 6-mer based evaluation of the coding potential in DNA [529]. The GRAIL system not only performs recognition of coding-region candidates, but also gene modeling (exon assembly), detection of indel errors and suggestion of likely corrections, detection of CpG islands, and recognition of PolII promoters and polyadenylation sites.

In the GeneParser scheme [494] intron/exon and splice site indicators are weighted by a neural network to approximate the log-likelihood that a sequence segment exactly represents an intron or exon (first, internal, or last). A dynamic programming algorithm is then applied to this data to find the combination of introns and exons that maximizes the likelihood function. Using this method, suboptimal solutions can be generated rapidly, each of them the optimum solution containing a given intron–exon junction. The authors also quantified the robustness of the method to substitution and frame-shift errors and showed how the system can be optimized for performance on sequences with known levels of sequencing errors.

Dynamic programming (DP) is applied to the problem of precisely identifying internal exons and introns in genomic DNA sequences. The GeneParser program first scores the sequence of interest for splice sites and for the following intron- and exon-specific content measures: codon usage, local compositional complexity, 6-tuple frequency, length distribution, and periodic asymmetry. This information is then organized for interpretation by DP. GeneParser employs the DP algorithm to enforce the constraints that introns and exons must be adjacent and nonoverlapping, and finds the highest-scoring combination of introns and exons subject to these constraints. Weights for the various classification procedures are determined by training a simple feed-forward neural network to maximize the number of correct predictions. In a pilot study, the system has been trained on a set of 56 human gene fragments containing 150 internal exons in a total of 158,691 bps of genomic sequence. When tested against the training data, GeneParser precisely identifies 75% of the exons and correctly predicts 86% of coding nucleotides as coding, while only 13% of non-exon bps were predicted to be coding. This corresponds to a correlation coefficient for exon prediction of 0.85. Because of the simplicity of the network weighting scheme, generalized performance is nearly as good as with the training set.

6.5.4 Prediction of Intron Splice Sites by Combining Local and Global Sequence Information

The complementarity of the splice site strength and coding region intensity was discovered in an NN study where prediction of sites was combined with a coding/noncoding prediction in the NetGene method [102]. The first NetGene program from 1991 was trained exclusively on human sequences, and has been available on the Internet since 1992 (netgene@cbs.dtu.dk). In this method three separate networks work together: the assignment thresholds of two local donor and acceptor sites' networks are regulated by a global network performing a coding/noncoding prediction. The three networks use windows of 15, 41, and 301 bp, respectively. Instead of a fixed threshold for splice site assignment in the local networks, the sharpness of the transition of the global exon-to-intron signal regulates the actual threshold. The aim was to improve the ratio between true and false donor and acceptor sites assigned. In regions with abruptly decreasing exon activity, donor sites should be "promoted" and acceptor sites suppressed, and vice versa in regions with abruptly increasing exon activity. In regions with only small changes in exon activity—that is, where the level was constantly high (inside exons) and where it was constantly low (inside introns, in untranslated exons and in the intergenic DNA)—a rather high confidence level in the splice site assignment should be demanded in order to suppress false positives.

To detect edges in the coding/noncoding output neuron levels, essentially the first derivative of the coding output neuron activity was computed by summing activities to the right of a given point and subtracting the corresponding sum for the left side, then dividing this difference by the number of addends. In order to reduce the number of cases where the individual sums covered both intron and exon regions, half the average length of the internal exons in the training set, 75 bp, was used as the scope when summing, giving derivatives close to $+1$ in the 3' end of introns and close to -1 in the 5' end for perfect coding/noncoding assignments.

In figure 6.12 an average-quality example of the coding/noncoding signal, the derivative Δ, and the forest of donor and acceptor site signals exceeding an activity of 0.25 are given for the GenBank entry HUMOPS, taken from the test set used in the study [102]. Note that some regions inside introns and in the nontranscribed part of the sequences show exonlike behavior.

In compiling algorithms that regulate the splice site assignment levels, the following expressions were used to assess possible weightings between the strength of the exon signal and the output, O, from the separate splice site networks: if

$$O_{\text{donor}} > e_D\Delta + c_D \tag{6.4}$$

assign splicing donor site, and if

$$O_{\text{acceptor}} > e_A\Delta + c_A \tag{6.5}$$

assign splicing acceptor site, where Δ was computed as described above. The constants c_D and c_A are equivalent to the ordinary cutoff assignment parameters, whereas e_D and e_A control the impact of the exon signal. Together the four constants control the relative strengths between the donor site and the coding/noncoding network on one side, and between the acceptor site and the coding/noncoding network on the other.

The four constants were optimized in terms of correlation coefficients and the percentage of true splice sites detected [102]. Compared with other methods, the number of false positives assigned was much lower—by a factor between 2 and 30, depending on the required level of detection of true sites—when the cutoff assignment level was controlled by the exon signal.

The general picture concerning the confidence levels on the splice site prediction and the coding/noncoding classification (as measured by the output neuron activities) was that exons smaller than 75 bp had rather weak exon output neuron levels (0.3–0.6) but relatively strong donor and acceptor site output neuron levels (0.7–1.0). Conversely, longer internal exons in general had rather sharp edges in the output neuron activities, with donor and acceptor site activities being somewhat weaker.

A similar method has been developed for prediction of splice sites in the plant *Arabidopsis thaliana*, NetPlantGene [245]. This plant model organism was the first for which the complete genome was sequenced, as the size of its genome (400Mbp) is very moderate compared with many other plants (see figure 1.2).

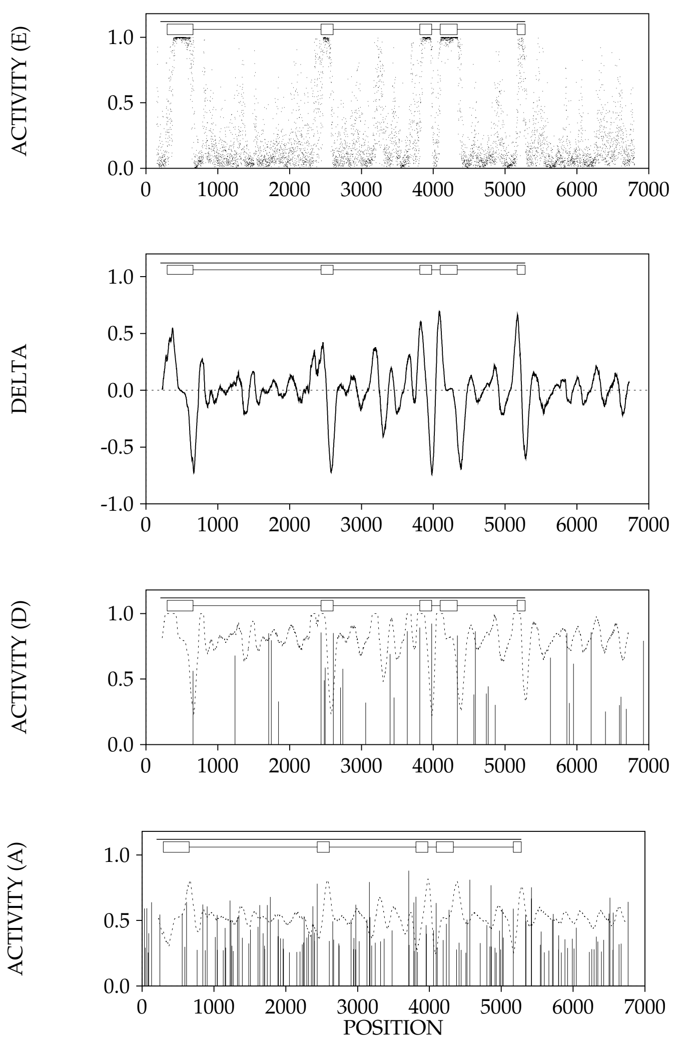


Figure 6.12: The Steps in the Operation of the NetGene Method for Prediction of Splice Sites on the Test Gene with GenBank Locus HUMOPS. A. (Top) Analog output from the output neuron in the coding/noncoding network, showing strong exon signals from the correct exons. Inside the introns and in the terminal nontranscribed part of the sequence, a number of regions show exonlike behavior. The boxes correspond to correct exons and the joining lines to introns, whereas the top line gives the extension of the transcript. B. The derivative of the analog coding/noncoding output. C. The donor site activities ($\geq$ 0.25) from the donor site assignment network. D. (Bottom) The acceptor site activities ($\geq$ 0.25) from the acceptor site assignment network. The variable cutoff assignment level for a 90% detection of true splice sites is shown as a dashed curve on the pin diagrams (3 and 4).

6.5.5 Doing Sequence Analysis by Inspecting the Order in Which Neural Networks Learn

Neural networks are well known for their ability to generalize from a set of examples to new cases. Recently another ability of the method has been recognized: important information on the internal structure of the training material can be obtained if the learning process is monitored carefully. A network does not learn the examples in random order; depending on the number of adjustable parameters, it will learn the linearly separable part of the data first and outliers that deviate from the prevailing patterns later. This was clear from the early work of Qian and Sejnowski [437] and from other work on alpha-helix prediction in proteins [244]. The training proceeds in two phases: a linearly separable part of the data is learned fast, while other examples are correctly classified by the network more slowly. In this later phase the network will often unlearn some of the examples learned in the first phase. It is obvious from the work on the genetic code described above that a similar, but more complex, picture emerges here.

The order in which a set of examples is learned first by a network reveals information on the relative nonlinearity of each single example, and hence on the pattern of regularity present in the complete set [97, 98]. This in turn can be used to identify abnormal examples that deviate strongly from the prevailing patterns, either due to the application of unnatural classification strategies or simply due to classification errors introduced randomly, without systematics. The neural network method provides a means for obtaining a high-quality feedback on low-quality data with a sound balance between abilities to model complex systematics and to detect deviations from ill-defined rules.

The ability to find errors has been exploited in many different projects, not only in the area of sequence analysis but also in other cases where input data may be assigned to an incorrect category. In the work using networks to predict intron splice sites, it was described in detail how to find errors produced by very different sources [100, 101]. During a training session the success of the learning was monitored in two different ways, one taking the training set as a whole and the other inspecting each window configuration separately. The performance on the training set as a whole is quantified by monitoring the decrease in the total network error E. If E remains constant for a large number of presentations of the training set, it indicates that no improvement can be gained from further training. The network has learned a single window configuration if the real–numbered activity of the output neuron fell on the same side of a cutoff value as the training target of the window configuration in question. The cutoff value separating the two output category assignments was mostly chosen to be 0.5. Thus, at any moment during training, an objective criterion for successful learning points uniquely to those inputs not being

Epoch	GenBank Locus	Sequence
1	HUMA1ATP	TACATCTTCTTTAAAGGTAAGGTTGCTCAACCA
1	HUMA1ATP	CCTGAAGCTCTCCAAGGTGAGATCACCCTGACG
1	HUMACCYBA	CCACACCCGCCGCCAGGTAAGCCCGGCCAGCCG
1	HUMACCYBA	CGAGAAGATGACCCAGGTGAGTGGCCCGCTACC
1	HUMACTGA	GCGCCCCAGACACCAGGTGAGTGGATGGCGCCG
1	HUMACTGA	AGAGAAGATGACTCAGGTGAGGCTCGGCCGACG
1	HUMACTGA	CACCATGAAGATCAAGGTGAGTCGAGGGGTTGG
1	HUMADAG	TCTTATACTATGGCAGGTAAGTCCATACAGAAG
1	HUMALPHA	CGTGGCTCTGTCCAAGGTAAGTGCTGGGCTACC
1	HUMALPI	CCTGGCTCTGTCCAAGGTAAGGGCTGGGCCACC
1	HUMALPPD	TGTGGCTCTGTCCAAGGTAAGTGCTGGGCTACC
1	HUMAPRTA	CCTGGAGTACGGGAAGGTAAGAGGGCTGGGGTG
1	HUMCAPG	GAAGGCTGCCTTCAAGGTAAGGCATGGGCATTG
1	HUMCFVII	GGAGTGTCCATGGCAGGTAAGGCTTCCCCTGGC
1	HUMCP21OH	CACCTTGGGCTGCAAGGTGAGAGGCTGATCTCG
1	HUMCP21OHC	CACCTTGGGCTGCAAGGTGAGAGGCTGATCTCG
1	HUMCS1	GTGGCAATGGCTCCAGGTAAGCGCCCCTAAAAT
1	HUMCSFGMA	AATGTTTGACCTCCAGGTAAGATGCTTCTCTCT
1	HUMCSPB	AAAGACTTCCTTTAAGGTAAGACTATGCACCTG
1	HUMCYC1A	GCTACGGACACCTCAGGTGAGCGCTGGGCCGGG
...	...	...
2	HUMA1ATP	CCTGGGACAGTGAATCGTAAGTATGCCTTTCAC
2	HUMA1ATP	AAAATGAAGACAGAAGGTGATTCCCCAACCTGA
2	HUMA1GLY2	CGCCACCCTGGACCGGGTGAGTGCCTGGGCTAG
2	HUMA1GLY2	GAGAGTACCAGACCCGGTGAGAGCCCCCATTCC
2	HUMA1GLY2	ACCGTCTCCAGATACGGTGAGGGCCAGCCCTCA
2	HUMA1GLY2	GGGCTGTCTTTCTATGGTAGGCATGCTTAGCAG
2	HUMA1GLY2	CACCGACTGGAAAAAGGTAAACGCAAGGGATTG
2	HUMACCYBA	GCGCCCCAGGCACCAGGTAGGGGAGCTGGCTGG
2	HUMACCYBA	CAGCCTTCCTTCCTGGGTGAGTGGAGACTGTCT
2	HUMACCYBA	CACAATGAAGATCAAGGTGGGTGTCTTTCCTGC
2	HUMACTGA	TCGCGTTTCTCTGCCGGTGAGCGCCCCGCCCCG
2	HUMADAG	CTTCGACAAGCCCAAAGTGAGCGCGCGCGGGGG
2	HUMADAG	TGTCCAGGCCTACCAGGTGGGTCCTGTGAGAAG
2	HUMADAG	CGAAGTAGTAAAAGAGGTGAGGGCCTGGGCTGG
...	...	...
11	HUMCS1	AACGCAACAGAAATCCGTGAGTGGATGCCGTCT
11	HUMGHN	AACACAACAGAAATCCGTGAGTGGATGCCTTCT
52	HUMHSP90B	CTCTAATGCTTCTGATGTAGGTGCTCTGGTTTC
80	HUMMETIF1	ACCTCCTGCAAGAAGAGTGAGTGTGAGGCCATC
112	HUMHSP90B	ATACCAGAGTATCTCAGTGAGTATCTCCTTGGC
113	HUMHST	GCGGACACCCGCGACAGTGAGTGGCGCGGCCAG
113	HUMLACTA	GACATCTCCTGTGACAGTGAGTAGCCCCTATAA
151	HUMKAL2	ATCGAACCAGAGGAGTGTACGCCTGGGCCAGAT
157	HUMCS1	CACCTACCAGGAGTTTGTAAGTTCTTGGGGAAT
157	HUMGHN	CACCTACCAGGAGTTTGTAAGCTCTTGGGGAAT
164	HUMALPHA	CAACATGGACATTGATGTGCGACCCCCGGGCCA
622	HUMCFVII	CTGATCGCGGTGCTGGGTGGGTACCACTCTCCC
636	HUMADAG	CCTGGAACCAGGCTGAGTGAGTGATGGGCCTGG
895	HUMAPOCIB	TCCAGCAAGGATTCAGGTTGTTGAGTGCTTGGG
970	HUMALPHA	CGGGCCAAGAAAGCAGGTGGAGCTGGGGCCCGG
2114	HUMAPRTA	ATCGACTACATCGCAGGCGAGTGCCAGTGGCCG

Table 6.3: Donor Site Window Configurations Learned Early and Late in the Course of Training. The applied network was small (nine nucleotides in the window, two hidden units, and one output unit), and it was trained on small 33bp segments surrounding the 331 splice sites in part I of the data set. Shown is the training epoch at which a configuration was assigned that spliced donor correctly by the network, its GenBank locus, and the nucleotide context surrounding the central G. Segments with large deviations from the standard donor site consensus sequence, $\frac{\mathsf{C}}{\mathsf{A}}AG/GT\frac{\mathsf{G}}{\mathsf{A}}$AGT, were learned only after a relatively large number of presentations.

classified correctly.

Table 6.3 shows how a small network with limited resources learns the donor sites. Many of them are learned quickly, while others require a considerable number of epochs. By examining the unlearnable window configurations, it was shown that a surprisingly large number of wrongly assigned donor sites could be detected. They appear in the public databases due to insufficient proofreading of the entries, but also due to experimental errors and erroneous interpretation of experiments. Information about the degree of regularity of, for example, donor site window configurations could be obtained by monitoring the course of the training. For the donor site assignment problem, window configurations learned early in the training process showed stronger conformity to the standard 5' consensus sequence $\frac{\mathsf{C}}{\mathsf{A}}\mathsf{AG/GT}\frac{\mathsf{G}}{\mathsf{A}}\mathsf{AGT}$ than those learned late.

6.6 Prediction Performance Evaluation

Over the years different means for evaluating the accuracy of a particular prediction algorithm have been developed [31]. Some prediction methods are optimized so as to produce very few false positives, others to produce very few false negatives, and so on. Normally it is of prime interest to ensure, for any type of prediction algorithm, that the method will be able to perform well on novel data that have not been used in the process of constructing the algorithm. That is, the method should be able to generalize to new examples from the same data domain.

It is often relevant to measure accuracy of prediction at different levels. In signal peptide prediction, for example, accuracy may be measured by counting how many *sequences* are correctly classified as signal peptides or nonsecretory proteins, instead of counting how many residues are correctly predicted to belong to a signal peptide. Similarly, protein secondary structure may be evaluated at the mean per-chain level, or at the per-amino acid level.

At higher levels, however, the measures tend to be more complicated and problem-specific. In the signal peptide example, it is also relevant to ask how many signal peptide sequences have the position of the cleavage site correctly predicted. In gene finding, a predicted exon can have have both ends correct, or only overlap to some extent. Burset and Guigo [110] have defined four simple measures of gene-finding accuracy at the exon level—sensitivity, specificity, "missing exons", and "wrong exons"—counting only predictions that are completely correct or completely wrong. For secondary structure prediction, this approach would be too crude, since the borders of structure elements (helices and sheets) are not precisely defined. Instead, the segment overlap measure (SOV) can be applied [454, 580]. This is a set of segment-based heuristic evaluation measures in which a correctly predicted segment position can give maximal score even though the prediction is not identical to the assigned segment. The score punishes broken predictions strongly, such as two predicted helices where only one is observed compared to one too small unbroken helix. In this manner the uncertainty of the assignment's exact borders is reflected in the evaluation measure. As this example illustrates, a high-level accuracy measure can become rather *ad hoc* when the precise nature of the prediction problem is taken into consideration.

For the sake of generality, we will therefore focus our attention on single residue/nucleotide assessment measures. For the secondary structure problem, consider an amino acid sequence of length N. The structural data $\mathbf{D}$ available for the comparison is the secondary structure assignments $\mathbf{D} = d_1, \ldots, d_N$. For simplicity, we will first consider the dichotomy problem of two alternative classes, for instance α-helix versus non-α-helix. In this case, the d_is are in general equal to 0 or 1. We can also consider the case where d_i has a value between 0 and 1, for example representing the surface exposure of amino acids, or the probability or degree of confidence, reflecting the uncertainty of our knowledge of the correct assignment at the corresponding position. The analysis for the multiple-class case, corresponding for example to three states, α-helices, β-sheets, and coil, is very similar. We now assume that our prediction algorithm or model, outputs a prediction of the form $\mathbf{M} = m_1, \ldots, m_N$. In general, m_i is a probability between 0 and 1 reflecting our degree of confidence in the prediction. Discrete 0/1 outputs, obtained for instance by thresholding or "winner-take-all" approaches, are also possible and fall within the theory considered here. The fundamental and general question we address is: How do we assess the accuracy of $\mathbf{M}$, or how do we compare $\mathbf{M}$ to $\mathbf{D}$?

A variety of approaches have been suggested in different contexts and at different times and this may have created some confusion. The issue of prediction accuracy is strongly related to the frequency of occurrence of each class. In protein secondary structure prediction the non-helix class covers roughly 70% of the cases in natural proteins, while only 30% belong to the helix class. Thus a constant prediction of "non-helix" is bound to be correct 70% of the time, although it is highly non-informative and useless.

Below we review different approaches and clarify the connections among them and their respective advantages and disadvantages.

A fundamental simplifying assumption underlying all these approaches is that the amino acid positions are weighted and treated equally (the independence and equivalence assumption). Thus, we assume e.g. that there is no weighting scheme reducing the influence of positions near the N- or C-termini, or no built-in mechanism that takes into account the fact that particular predictions must vary somewhat "smoothly" (for instance, if a residue belongs to the α-helix category, its neighbors have a slightly higher chance of also being in the α-helix category). Conversely, when predicting functional sites such as intron splice sites, translation start sites, glycosylation, or phosphorylation sites, we assume the prediction of a site is either true or false, so that there is no reward for almost correctly placed sites.

Under the independence and equivalence assumption, if both **D** and **M** are binary, it is clear that their comparison can be entirely summarized by four numbers

- TP = number of times d_i is helix, m_i is helix (true positive)
- TN = the number of times d_i is non-helix, m_i is non-helix (true negative)
- FP = the number of times d_i is non-helix, m_i is helix (false positive)
- FN = the number of times d_i is helix, m_i is non-helix (false negative)

satisfying $TP + TN + FP + FN = N$. When **D** and/or **M** are not binary, then of course the situation is more complex and four numbers do not suffice to summarize the situation. When **M** is not binary, binary predictions can still be obtained by using cutoff thresholds. The numbers TP, TN, FP, and FN will then vary with the threshold choice. The numbers TP, TN, FP, and FN are often arranged into a 2×2 contingency matrix,

	$\mathbf{M}$	$\bar{\mathbf{M}}$
$\mathbf{D}$	TP	FN
$\bar{\mathbf{D}}$	FP	TN

Even with four numbers alone, it is not immediately clear how a given prediction method fares. This is why many of the comparison methods aim at constructing a single number measuring the "distance" between **D** and **M**. But it must be clear from the outset that information is always lost in such a process, even in the binary case, i.e. when going from the four numbers above to a single one. In general, several different vectors (TP, TN, FP, FN) will yield the same distance. We now review several ways of measuring the performance of **M** and their merits and pitfalls.

6.7 Different Performance Measures

6.7.1 Percentages

The first obvious approach is to use percentages derived from TP, TN, FP, and FN. For instance, Chou and Fasman [128, 129] used the percentage of correctly predicted helices

$$PCP(\mathbf{D},\mathbf{M}) = 100\frac{TP}{TP+FN}, \tag{6.6}$$

which is the same as the sensitivity (see section 6.7.9) expressed as a percentage. This number alone provides no information whatsoever about false positives. It can be complemented by the percentage of correctly predicted non-helices

$$PCN(\mathbf{D},\mathbf{M}) = 100\frac{TN}{TN+FP}. \tag{6.7}$$

The average of the previous two numbers has been used in the literature [128, 129] and is often called Q_α. While Q_α is a useful indicator, it can be misleading [549] and can be computed only if both **D** and **M** are binary. Intuitively, any single number that is constructed using only two numbers out of the four (TP, TN, FP, FN) is bound to be highly biased in some trivial way. If the two numbers are TP and FP, for instance, then any two situations (TP, TN, FP, FN) and (TP, TN', FP, FN') lead to the same score regardless of how different they may be.

6.7.2 Hamming Distance

In the binary case, the Hamming distance between **D** and **M** is defined by

$$HD(\mathbf{D},\mathbf{M}) = \sum_i |d_i - m_i|. \tag{6.8}$$

This sum is obviously equal to the total number of errors $FP + FN$. Thus it is equivalent to a single percentage measure. This distance does not take into account the proportion of examples that belong to a given class. It becomes less and less useful as this proportion moves away from 50%. In the non purely binary case, the Hamming distance is called the L^1 distance.

6.7.3 Quadratic “Distance”

The quadratic or Euclidean or LMS (least mean square) “distance” is defined by

$$Q(\mathbf{D},\mathbf{M}) = (\mathbf{D}-\mathbf{M})^2 = \sum_i (d_i - m_i)^2. \tag{6.9}$$

Strictly speaking, a proper distance is defined by taking the square root of the above quantity (see the L^2 distance in the next section). In the purely binary case, the quadratic distance reduces to the Hamming distance and is again equal to $FP + FN$. This quantity has the advantage of being defined for non-binary variables, and it is often associated with a negative log-likelihood approach for a Gaussian model of the form

$$P(d_i|m_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-(d_i - m_i)^2/2\sigma^2) \tag{6.10}$$

where σ acts as a scaling factor with respect to $Q(\mathbf{D}, \mathbf{M})$. For binary variables, the quadratic distance is identical to the Hamming distance. The main drawback is that the Gaussian model is often not relevant for prediction problems and the value of the quadratic distance again poorly reflects the proportion of positions that belongs to a given class. Another problem is that the LMS distance has a limited dynamic range due to the fact that m_i and d_i are between 0 and 1. This is not ideal for learning algorithms where large error signals can be used to accelerate the learning process. A logarithmic variation on the LMS distance that obviates this problem is given by

$$LQ(\mathbf{D}, \mathbf{M}) = -\sum_i \log[1 - (d_i - m_i)^2]. \tag{6.11}$$

This modified error function has been used in several neural network implementations; see for example [99, 245, 236].

6.7.4 L^p Distances

More generally, the L^p distance is defined by

$$LP(\mathbf{D}, \mathbf{M}) = [\sum_i |d_i - m_i|^p]^{1/p}. \tag{6.12}$$

Such a distance applies of course to any numerical values. When $p = 1$ we find the Hamming distance, and when $p = 2$ we find the proper Euclidean distance. When $p \to \infty$, the L^∞ distance is the sup distance: $\max_i |d_i - m_i|$. This distance provides an upper bound associated with the worst case, but is not very useful in assessing the performance of a protein secondary structure prediction algorithm. Other values of p are rarely used in practice, and are of little help for assessing prediction performance in this context. In the binary case, the L^p distance reduces to $(FP + FN)^{1/p}$. For $p = 1$, this reduces again to the Hamming distance.

6.7.5 Correlation

One of the standard measures used by statisticians is the correlation coefficient, also called the Pearson correlation coefficient:

$$C(\mathbf{D},\mathbf{M}) = \sum_i \frac{(d_i - \bar{d})(m_i - \bar{m})}{\sigma_{\mathbf{D}}\sigma_{\mathbf{M}}}, \tag{6.13}$$

where $\bar{d} = \sum d_i/N$ and $\bar{m} = \sum m_i/N$ are the averages and $\sigma_{\mathbf{D}}, \sigma_{\mathbf{M}}$ the corresponding standard deviations. In the context of secondary structure prediction, this is also known as the Matthews correlation coefficient in the literature since it was first used in [382]. The correlation coefficient is always between -1 and $+1$ and can be used with non-binary variables. It is a measure of how strongly the normalized variables $(d_i - \bar{d})/\sigma_{\mathbf{D}}$ and $(m_i - \bar{m})/\sigma_{\mathbf{M}}$ tend to have the same sign and magnitude. A value of -1 indicates total disagreement and $+1$ total agreement. The correlation coefficient is 0 for completely random predictions. Therefore it yields easy comparison with respect to a random baseline. If two variables are independent, then their correlation coefficient is 0. The converse in general is not true.

In vector form, the correlation coefficient can be rewritten as a dot product between normalized vectors

$$C(\mathbf{D},\mathbf{M}) = \frac{(\mathbf{D} - \bar{d}\mathbf{1})(\mathbf{M} - \bar{m}\mathbf{1})}{\sqrt{(\mathbf{D} - \bar{d}\mathbf{1})^2}\sqrt{(\mathbf{M} - \bar{m}\mathbf{1})^2}} = \frac{\mathbf{DM} - N\bar{d}\bar{m}}{\sqrt{(\mathbf{D}^2 - N\bar{d}^2)(\mathbf{M}^2 - N\bar{m}^2)}}, \tag{6.14}$$

where $\mathbf{1}$ denotes the N-dimensional vector of all ones. As such, $C(\mathbf{D},\mathbf{M})$ is related to the L^2 distance, but is not a distance itself since it can assume negative values. If the vectors $\mathbf{D}$ and $\mathbf{M}$ are normalized, then clearly $Q(\mathbf{D},\mathbf{M}) = (\mathbf{D} - \mathbf{M})^2 = 2 - 2\mathbf{DM} = 2 - 2C(\mathbf{D},\mathbf{M})$. Unlike some of the previous measures, the correlation coefficient has a global form rather than being a sum of local terms.

In the case where $\mathbf{D}$ and $\mathbf{M}$ consist of binary 0/1 vectors, we have $\mathbf{D}^2 = TP + FN$, $\mathbf{M}^2 = TP + FP$, $\mathbf{DM} = TP$, etc. With some algebra the sum above can be written as

$$C(\mathbf{D},\mathbf{M}) = \frac{TP - N\bar{d}\bar{m}}{N\sqrt{\bar{d}\bar{m}(1 - \bar{d})(1 - \bar{m})}}. \tag{6.15}$$

Here the average number of residues in the helix class satisfies $\bar{d} = (TP + FN)/N$, and similarly for the predictions $\bar{m} = (TP + FP)/N$. Therefore

$$\begin{aligned} C(\mathbf{D},\mathbf{M}) &= \frac{N \times TP - (TP + FN)(TP + FP)}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}} \\ &= \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FN)(TP + FP)(TN + FP)(TN + FN)}}. \end{aligned} \tag{6.16}$$

The correlation coefficient uses all four numbers (TP, TN, FP, FN) and may often provide a much more balanced evaluation of the prediction than for instance the percentages. There are situations, however, where even the correlation coefficient is unable to provide a completely fair assessment. The correlation coefficient will, for example, be relatively high in cases where a prediction algorithm gives very few or no false positives, but at the same time very few true positives. One simple observation that will be useful in a later section is that C is symmetric with respect to FP and FN.

One interesting property of the correlation coefficient is that there is a simple approximate statistical test for deciding whether it is significantly better than zero, i.e. whether the prediction is significantly more correlated with the data than a random guess with the same $\bar{m}$ would be. If the chi-squared test is applied to the 2×2 contingency matrix containing TP, TN, FP, and FN, it is easy to show that the test statistic is $\chi^2 = N \times C^2(\mathbf{D}, \mathbf{M})$.

6.7.6 Approximate Correlation

Burset and Guigo [110] defined an "approximate correlation" measure to compensate for an alleged problem with the Matthews correlation coefficient: that it is not defined when any of the sums $TP+FN$, $TP+FP$, $TN+FP$, or $TN+FN$ reaches zero, e.g. if there are no positive predictions. Instead, they use the Average Conditional Probability (ACP), which is defined as

$$ACP = \frac{1}{4}\left[\frac{TP}{TP+FN} + \frac{TP}{TP+FP} + \frac{TN}{TN+FP} + \frac{TN}{TN+FN}\right] \tag{6.17}$$

if all the sums are nonzero; otherwise, it is the average over only those conditional probabilities that are defined. The Approximate Correlation (AC) is a simple transformation of the ACP:

$$AC = 2 \times (ACP - 0.5). \tag{6.18}$$

Like C, AC gives 1, 0, and -1 for perfect, random, and all-false predictions, respectively, and Burset and Guigó observe that it is close to the real correlation value.

However, the problem they intend to solve does not exist, since it is easy to show that C approaches 0 if any of the sums approaches 0. This also makes intuitive sense, since a prediction containing only one category is meaningless and does not convey any information about the data. On the contrary, it can be shown that the AC approach introduces an unfortunate discontinuity in this limit because of the deletion of undefined probabilities from the expression for ACP, so it does *not* give 0 for meaningless predictions. Since there is furthermore no simple geometrical interpretation for AC, it is an unnecessary approximation and we see no reason to encourage its use.

6.7.7 Relative Entropy

The relative entropy, or cross entropy, or KL (Kullback-Leibler) contrast between two probability vectors $\mathbf{X} = (x_1, \ldots, x_M)$ and $\mathbf{Y} = (y_1, \ldots, y_M)$ with $x_i, y_i \geq 0$ and $\sum x_i = \sum y_i = 1$ is defined as

$$H(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{M} x_i \log \frac{x_i}{y_i} = -H(\mathbf{X}) - \sum_i x_i \log y_i \tag{6.19}$$

where $H(\mathbf{X}) = -\sum x_i \log x_i$ is the usual entropy. It has its roots in information theory [342, 341]. It is well known that $H(\mathbf{X}, \mathbf{Y})$ is always positive, convex in both its variables, and equal to 0 if and only if $\mathbf{X} = \mathbf{Y}$. Strictly speaking, it is not a distance, for instance because it is not symmetric. It is easy to construct a distance using a symmetrized version. In practice, however, this is rarely necessary and the form above is sufficient. If $\mathbf{Y} = \mathbf{X} + \epsilon$ is close to $\mathbf{X}$, then a simple Taylor expansion shows that

$$H(\mathbf{X}, \mathbf{X} + \epsilon) = -\sum_i x_i \left[\log(1 + \frac{\epsilon_i}{x_i}) \right] \approx \sum_i \frac{\epsilon_i^2}{x_i}. \tag{6.20}$$

In particular, if $\mathbf{X}$ is uniform, then in its neighborhood the relative entropy behaves like the LMS error.

Returning to the secondary structure prediction problem, we can then assess the performance of the prediction $\mathbf{M}$ by the quantity:

$$H(\mathbf{D}, \mathbf{M}) = \sum_{i=1}^{N} \left[d_i \log \frac{d_i}{m_i} + (1 - d_i) \log \frac{(1 - d_i)}{(1 - m_i)} \right]. \tag{6.21}$$

This is just the sum of the relative entropies at each position i. This form of course works perfectly well on non-binary data (for example, binding affinities), or when $\mathbf{D}$ alone is binary. When $\mathbf{M}$ is also binary, then the relative entropy has $FP + FN$ components that are infinite (it behaves like $H(\mathbf{D}, \mathbf{M}) \approx (FP + FN)\infty$) and cannot really be used.

6.7.8 Mutual Information

Consider two random variables $\mathcal{X}$ and $\mathcal{Y}$ with probability vectors $\mathbf{X} = (x_1, \ldots, x_M)$ and $\mathbf{Y} = (y_1, \ldots, y_K)$. Let $\mathcal{Z}$ be the joint random variable $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$ over the cartesian product with probability vector $\mathbf{Z}$. The mutual information $I(\mathcal{X}, \mathcal{Y})$ or $I(\mathbf{X}, \mathbf{Y})$ between $\mathcal{X}$ and $\mathcal{Y}$ is defined as the relative entropy between $\mathcal{Z}$ and the product $\mathcal{X}\mathcal{Y}$:

$$I(\mathcal{X}, \mathcal{Y}) = H(\mathcal{Z}, \mathcal{X}\mathcal{Y}). \tag{6.22}$$

As such it is always positive. It is easy to understand the mutual information in Bayesian terms: it represents the reduction in uncertainty of one variable when the other is observed, that is between the *prior* and *posterior distributions*. The uncertainty in $\mathcal{X}$ is measured by the entropy of its prior $H(\mathcal{X}) = \sum x_i \log x_i$. Once we observe $\mathcal{Y} = y$, the uncertainty in $\mathcal{X}$ is the entropy of the posterior distribution, $H(\mathcal{X}|\mathcal{Y} = y) = \sum_x P(\mathcal{X} = x|\mathcal{Y} = y) \log P(\mathcal{X} = x|\mathcal{Y} = y)$. This is a random variable that depends on the observation y. Its average over the possible ys is called the *conditional entropy*:

$$H(\mathcal{X}|\mathcal{Y}) = \sum_y P(y)H(\mathcal{X}|\mathcal{Y} = y). \tag{6.23}$$

Therefore the difference between the entropy and the conditional entropy measures the average information that an observation of $\mathcal{Y}$ brings about $\mathcal{X}$. It is straightforward to check that

$$I(\mathcal{X},\mathcal{Y}) = H(\mathcal{X}) - H(\mathcal{X}|\mathcal{Y}) = H(\mathcal{Y}) - H(\mathcal{Y}|\mathcal{X}) = H(\mathcal{X}) + H(\mathcal{Y}) - H(\mathcal{Z}) = I(\mathcal{Y},\mathcal{X}) \tag{6.24}$$

or, using the corresponding distributions,

$$I(\mathbf{X},\mathbf{Y}) = H(\mathbf{X}) - H(\mathbf{X}|\mathbf{Y}) = H(\mathbf{Y}) - H(\mathbf{Y}|\mathbf{X}) = H(\mathbf{X}) + H(\mathbf{Y}) - H(\mathbf{Z}) = I(\mathbf{Y},\mathbf{X}). \tag{6.25}$$

Going back to the secondary structure problem, when **D** and **M** are both binary, the mutual information is measured by

$$\begin{aligned} I(\mathbf{D},\mathbf{M}) \quad = \quad & -H\left(\frac{TP}{N},\frac{TN}{N},\frac{FP}{N},\frac{FN}{N}\right) \\ & -\frac{TP}{N}\log\left[\frac{TP+FP}{N}\frac{TP+FN}{N}\right] - \frac{FN}{N}\log\left[\frac{TP+FN}{N}\frac{TN+FN}{N}\right] \\ & -\frac{FP}{N}\log\left[\frac{TP+FP}{N}\frac{TN+FP}{N}\right] - \frac{TN}{N}\log\left[\frac{TN+FN}{N}\frac{TN+FP}{N}\right] \end{aligned} \tag{6.26}$$

or

$$\begin{aligned} I(\mathbf{D},\mathbf{M}) \quad = \quad & -H\left(\frac{TP}{N},\frac{TN}{N},\frac{FP}{N},\frac{FN}{N}\right) \\ & -\frac{TP}{N}\log[\bar{d}\bar{m}] - \frac{FN}{N}\log[\bar{d}(1-\bar{m})] \\ & -\frac{FP}{N}\log[(1-\bar{d})\bar{m}] - \frac{TN}{N}\log[(1-\bar{d})(1-\bar{m})] \end{aligned} \tag{6.27}$$

(see also [549]), where $\bar{d} = (TP + FN)/N$ and $\bar{m} = (TP + FP)/N$ (as before), and

$$H(\frac{TP}{N}, \frac{TN}{N}, \frac{FP}{N}, \frac{FN}{N}) = -\frac{TP}{N}\log\frac{TP}{N} - \frac{TN}{N}\log\frac{TN}{N} - \frac{FP}{N}\log\frac{FP}{N} - \frac{FN}{N}\log\frac{FN}{N} \tag{6.28}$$

is the usual entropy. Like the correlation, the mutual information is a global measure rather than a sum of local terms. It is clear that the mutual information always satisfies $0 \leq I(\mathbf{D}, \mathbf{M}) \leq H(\mathbf{D})$. Thus in the assessment of prediction performance, it is customary to use the normalized mutual information [452, 454] coefficient

$$IC(\mathbf{D}, \mathbf{M}) = \frac{I(\mathbf{D}, \mathbf{M})}{H(\mathbf{D})} \tag{6.29}$$

with

$$H(\mathbf{D}) = -\frac{TP + FN}{N}\log\left[\frac{TP + FN}{N}\right] - \frac{TN + FP}{N}\log\left[\frac{TN + FP}{N}\right] \tag{6.30}$$

or, more briefly expressed, $H(\mathbf{D}) = -\bar{m}\log\bar{m} - (1-\bar{m})\log(1-\bar{m})$. The normalized mutual information satisfies $0 \leq IC(\mathbf{D}, \mathbf{M}) \leq 1$. When $IC(\mathbf{D}, \mathbf{M}) = 0$, then $I(\mathbf{D}, \mathbf{M}) = 0$ and the prediction is totally random (**D** and **M** are independent). When $IC(\mathbf{D}, \mathbf{M}) = 1$, then $I(\mathbf{D}, \mathbf{M}) = H(\mathbf{D}) = H(\mathbf{M})$ and the prediction is perfect. Like the correlation coefficient, the mutual information coefficient is a global measure rather than a sum of local terms. The mutual information is symmetric in FP and FN, but the mutual information coefficient is *not* symmetric because of its denominator.

6.7.9 Sensitivity and Specificity

In a two-class prediction case where the output of the prediction algorithm is continuous, the numbers TP, TN, FP, and FN depend on how the threshold is selected. Generally, there is a tradeoff between the number of false positives and the number of false negatives produced by the algorithm.

In a ROC curve (receiver operating characteristics) one may summarize such results by displaying for threshold values within a certain range the "hit rate" (sensitivity, $TP/(TP + FN)$) versus the "false alarm rate" (also known as false positive rate, $FP/(FP + TN)$. Typically the hit rate increases with the false alarm rate (see figure 8.10). Alternatively, one can also display the sensitivity $(TP/(TP + FN))$ versus the specificity $(TP/(TP + FP))$ in a similar plot or separately as a function of threshold in two different plots.

While the sensitivity is the probability of correctly predicting a positive example, the specificity as defined above is the probability that a positive prediction is correct. In medical statistics, the word "specificity" is sometimes used in a different sense, meaning the chance of correctly predicting a negative example: $TN/(FP+TN)$, or 1 minus the false positive rate. We prefer to refer to this as the sensitivity of the negative category.

If we write $x = TP/(TP+FN)$ for the sensitivity and $y = TP/(TP+FP)$ for the specificity, then

$$\begin{aligned} TP + FP = \frac{TP}{y} \qquad TP + FN &= \frac{TP}{x} \\ TN + FP = N - (TP + FN) &= \frac{Nx - TP}{x} \\ TN + FN = N - (TP + FP) &= \frac{Ny - TP}{y} \end{aligned} \tag{6.31}$$

provided $x \neq 0$ and $y \neq 0$, which is equivalent to $TP \neq 0$, a rather trivial case. In other words, we just reparameterize (TP, TN, FP, FN) using (TP, x, y, N). In this form, it is clear that we can substitute these values in (6.16) to derive, after some algebra, an expression for the correlation coefficient as a function of the specificity and the sensitivity:

$$C(\mathbf{D}, \mathbf{M}) = \frac{Nxy - TP}{\sqrt{(Nx - TP)(Ny - TP)}}. \tag{6.32}$$

Notice that this expression is entirely symmetric in x and y, i.e. in the specificity and sensitivity, or equivalently also in FP and FN, the number of false positives and false negatives. In fact, for a given TP, exchanging FP and FN is equivalent to exchanging x and y. A similar calculation can be done in order to re-express the mutual information of (6.27) or the mutual information coefficient of (6.29) in terms of TP, x, y, and N. The mutual information is entirely symmetric in x and y, or FP and FN (this is not true of the mutual information coefficient).

6.7.10 Summary

In summary, under the equivalence and independence assumption, if both **D** and **M** are binary, then all the performance information is contained in the numbers TP, TN, FP, and FN. Any measure of performance using a single number discards some information. The Hamming distance and the quadratic distance are identical. These distances, as well as the percentages and the L^p distances, are based on only two out of the four numbers TP, TN, FP, and FN. The correlation coefficient and the mutual information coefficient are based on all four parameters and provide a better summary of performance in this case. In the continuous case, the recommended measures are the correlation coefficient and the relative entropy.