

The Applicability of Recurrent Neural Networks for Biological Sequence Analysis

John Hawkins

School of Information Technology and Electrical Engineering

University of Queensland, QLD 4072, Australia.

E-mail: jhawkins@itee.uq.edu.au

Mikael Bodén

School of Information Technology and Electrical Engineering

University of Queensland, QLD 4072, Australia.

E-mail: mikael@itee.uq.edu.au

Abstract

Selection of machine learning techniques requires a certain sensitivity to the requirements of the problem. In particular the problem can be made more tractable by deliberately using algorithms that are biased towards solutions of the requisite kind. In this paper we argue that recurrent neural networks have a natural bias towards a problem domain of which biological sequence analysis tasks are a subset. We use experiments with synthetic data to illustrate this bias. We then demonstrate that this bias can be exploitable using a dataset of protein sequences containing several classes of subcellular localisation targeting peptides. The results show that com-

pared with feed forward, recurrent neural networks will generally perform better on sequence analysis tasks. Furthermore, as the patterns within the sequence become more ambiguous, the choice of specific recurrent architecture becomes more critical.

Keywords: machine learning, neural network architecture, recurrent neural network, bias, biological sequence analysis, motif, subcellular localisation, pattern recognition, classifier design,

1 Introduction

Recurrent neural networks were initially developed for sequence recognition tasks in the domain of natural language processing [6, 15]. In spite of the observations that they appear biased to problems of this kind [12, 5], theoretical foundations and empirical studies by which to qualify the limits and bounds of their bias are scarce. The ability of recurrent networks are of particular interest, since the recent emergence of massive genomic sequence data prompts a search for effective methods of sequence analysis.

A wide range of techniques have been applied to the classification of biological sequences, ranging from carefully constructed hidden markov models for theory-rich applications, e.g. [11], to a plethora of machine learning techniques for data-rich theory-poor domains. In general all machine learning approaches use a parameterised model to perform a, usually non-linear, separation of the sequence space into classes. What differentiates among them is the structure of the parameterised model (the architecture) and the learning algorithm used for searching the parameter space. Baldi and Brunak provide a comprehensive overview of a wide range of applications [3]. In practice, all machine learning techniques will have a bias, or sensitivity, toward patterns of a particular kind. This is a consequence of the necessity of inductive bias, first raised by Tom Mitchell [14]. Thus, to obtain optimal results in the application of machine learning algorithms we should be careful to employ architectures that have the ‘right’ kind of bias.

In this study we provide two empirical studies that elaborate on what it means in practice for an architecture to be biased towards sequential processing. Firstly, we use a method for estimating *architectural bias* to show that recurrent neural networks have an *a priori* sensitivity to patterns of a sequential nature, with some tolerance to movement and deformation in the pattern. Secondly we demonstrate that this sensitivity is accessible in the training process by performing a case study: the determination of protein subcellu-

lar localisation from linear amino acid sequences [7]. The results indicate that recurrent networks are able to be trained to exploit their inherent sensitivity to flexible sequential patterns. Furthermore, when the sequence task is difficult, then different recurrent architectures exploit different features within the sequence. The results in general provide an explanation of why recurrent networks, when compared to standardly used feed forward networks, are sometimes more successful at biological sequence analysis tasks [4, 20].

2 Estimating architectural bias

The first study relies on a method devised by Christiansen and Chater for estimating the architectural bias of a neural network [5]. This technique has since been employed in the recent work of Tino, Hammer and others [19, 9, 18]. The architectural bias is a measure of how well the network's internal nodes cluster the presented data into the required classes. The measure is performed prior to training, and as such captures any tendency the architecture alone has for differentiating between the different classes of data.

Two recurrent architectures - a simple recurrent neural network and a sequential cascaded neural network - are benchmarked against a conventional feed forward neural network architecture using a synthetically generated data set. The two classes to be separated are determined by the presence or absence of a particular motif within the sequence. We wish to stress at the outset, the intention here is not to assess the ability of such architectures to directly detect motifs (for which there are many effective methods, e.g. MEME [2]), nor is the intention to detect sequence homologies by way of flexible alignments of sequence segments as is done by Blast [1] or PatternHunter [13]. The purpose of the study is to assess how *accessible* these sequential patterns are to different neural networks. This accessibility is an indication of how well the machine will be able to learn to distinguish between classes of biological molecules.

What constitutes a motif in molecular biology varies depending on the particular biological polymer and on the molecular function. However there are certain regular and important characteristics, they are often relatively short (compared to the entire sequence), they consist of a moderately fixed length subsequence with some varying degree of polymorphism, within 10 to 50 percent. This polymorphism is almost always restricted to very particular sites, and may allow for the removal of a monomer altogether. It is worth noting that the types of motifs sought in our simulations are more in line with PROSITE patterns than profiles, meaning that we include no information about the relative frequencies of the set of symbols that are allowed at a particular position. Because the more complex motif specifications like PROSITE profiles involve extending the idea of symbol subsets to have probabilistic scores, they do not describe a fundamentally different type of motif and hence our results will still hold some relevance for motifs described that way.

The generation of the motifs is regulated by a number of parameters, that determine the flexibility of the match. The simulations involve observing the relative bias of each architecture to detecting each of the different motifs. Thus, the use of artificial data allows us to investigate the corresponding bias of the architectures as individual properties of the motifs change. After initial trials we refined the problem to focus on two biologically important features; we allow for a small proportion of monomers to be deleted from fixed positions within the motifs and we study the effect of allowing the entire motif to be shifted in its position along the parent sequence.

It may be objected that the highly specific sequential patterns being described as motifs here are too much of a simplification of the problem to provide any meaningful insight. However experience indicates that the complex patterns found in real sequences, are often composed of smaller motifs arranged sequentially themselves, with varying distance between them [10, 2]. More complex patterns like these are simply meta-patterns of the kind being evaluated in this study, hence in principle the recurrent architectures should be no

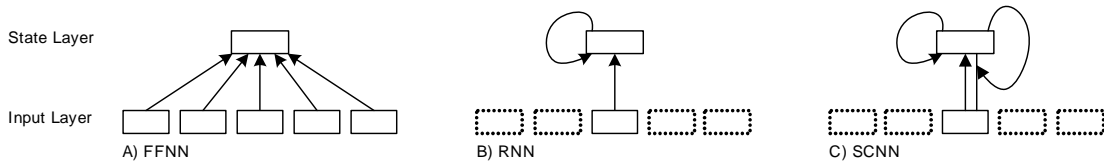


Figure 1: Simplified model of the three network architectures. A) FFNN: The feed forward neural network takes the entire input sequence in one step. It has a set of weights connecting the input layer to the state layer. B) RNN: The simple recurrent neural network takes in a smaller window of inputs and works its way along the sequence until it reaches the final window. In addition to the standard feed forward weights, it has a set of recurrent weights connecting the previous activations of the hidden layer back into itself. C) SCNN: The sequential cascaded neural network extends the simple recurrent network with the addition of a layer of second order recurrent connections. Each connection in this weight layer multiplies together an input node activation with a hidden node activation from the previous time step. The result is a bank of connections that feedback all possible multiplicative combinations of the current inputs with the hidden node activations from the previous time step.

less able to deal with these meta-patterns.

2.1 Networks

We benchmark three neural network architectures against each other. The networks are depicted schematically in Figure 1 and described formally in Section 5.3. Due to the fact that we are studying the behaviour of the hidden nodes prior to training, the networks were only built with input and hidden layers. The output layer and its connections were excluded because in all architectures they did not influence the dynamics of the neural network. No training was conducted, because we were investigating the bias of the machine architecture alone, and the weights were initialised randomly for each run.

FFNN: A standard feed forward neural network, for which there is one set of connections fully connecting the input layer to the hidden layer. In this architecture the input sequence is taken in as a whole, resulting in the activations of the hidden nodes.

RNN: A simple recurrent neural network [6] has a set of feed forward connections from

the input to hidden nodes, but also has a set of delayed recurrent connections from the hidden layer back onto itself. The simple recurrent neural network differs also from the feed forward network in that the input sequence is presented one symbol at a time. As such the input layer contains many fewer nodes, and requires as many presentations there are symbols in the sequence.

SCNN: A sequential cascaded neural network [15] is a higher order recurrent network. Rather than just having recurrent connections on the hidden nodes only, it also has recurrent connections that are second order combinations of the hidden nodes and input nodes. Consequently, each symbol in the input sequence is interpreted in the context of the previous state of the machine.

2.2 Data

The synthetic data set is generated as is described in Section 5.1. For each configuration, we randomly generate 500 sequences of which half are positive (containing a motif) and half is negative (not containing the motif). The motif pattern is generated randomly for each configuration. Each symbol is represented, as convention prescribes, by a unique one-hot (orthonormal) bit code.

2.3 Simulations

The first round of simulations are designed to illustrate the sequence features for which each of the networks has closest affinity. The synthetic data sets are presented to the set of architectures to be investigated. At the end of each presentation the state of the hidden nodes in each network is stored. Once the entire data set has been presented, the hidden node activation vectors are clustered using the K -means algorithm, and then judged in terms of whether the desired classes within the data set are represented in the

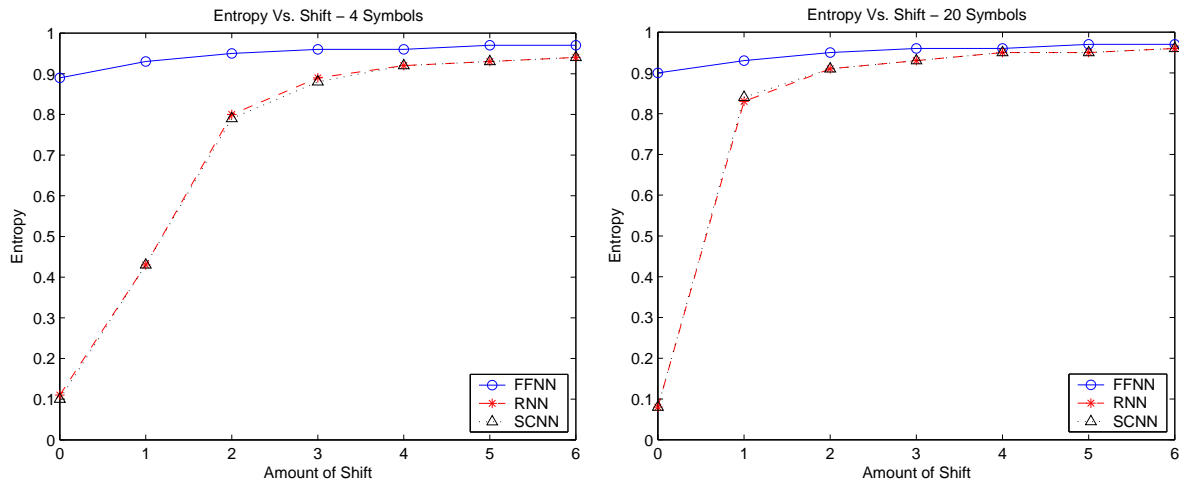


Figure 2: The average entropy within the networks as the amount of allowed shift in the motif is varied. These values are averaged across sequences ranging in length from 50 to 300, with the motifs ranging from one third to an eighth of the sequence, with between 0 and 4 deletes. Between one fifth and a twentieth of each motif consisted of wildcards. All networks were equipped with 5 state units and $K = 10$ clusters were identified. The graphs show that the difference in bias decays with the amount of allowed shift, but is interestingly dependent upon the number of symbols in the alphabet.

clusters that are formed. The resulting score is an entropy measure, indicating how well the organisation of the vectors is aligned with the problem. Values close to one indicate random organisation, while values close to zero indicate that the state space is organised in line with the classes within the data. The details of this analysis are presented in Section 5.2. All results are averages over at least 10 differently seeded runs.

2.4 Results

Firstly the problem of motifs that have a variable position within the sequence is illuminated in Figure 2. We see that both the recurrent networks have a substantial advantage over the feed forward network while ever the amount of shift is relatively small. The advantage of recurrent networks decays away as the shift increases and both recurrent architectures follow a strikingly similar pattern as the amount of shift grows.

Shift	Number of Deletes				
	0	1	2	3	4
0	0.86 (0.10)	0.88 (0.11)	0.89 (0.11)	0.90 (0.11)	0.92 (0.10)
1	0.92 (0.43)	0.92 (0.43)	0.93 (0.43)	0.94 (0.43)	0.94 (0.44)
2	0.94 (0.80)	0.95 (0.80)	0.95 (0.80)	0.95 (0.80)	0.95 (0.80)
3	0.95 (0.89)	0.96 (0.89)	0.96 (0.88)	0.96 (0.88)	0.96 (0.88)
4	0.96 (0.92)	0.96 (0.92)	0.96 (0.92)	0.96 (0.92)	0.96 (0.92)
5	0.96 (0.93)	0.96 (0.94)	0.96 (0.94)	0.97 (0.93)	0.97 (0.93)
6	0.96 (0.94)	0.97 (0.95)	0.97 (0.94)	0.97 (0.94)	0.97 (0.94)

Table 1: The average entropy of the two network architectures - FFNN (RNN) discriminating between sequences with varying amounts of shift and number of deletes. Sequences varied between 50 and 300 symbols long from an alphabet of 4 symbols. Motifs varied between 33% and 13% of the sequence length with between 20% and 5% wildcards. All networks were equipped with 5 state units and $K = 10$ clusters were identified.

Secondly as Figure 3 indicates the recurrent neural networks are consistently better prepared for motifs that contain deletes. Notably the entropy varies very little as the number of deletes increases, indicating that this is a reliable advantage to using recurrent architectures. This effect is demonstrated more definitively in Table 1, where we see that for any specific amount of shift, both the recurrent and feed forward architectures retain a relatively constant entropy as the number of deletes is changed. However for all configurations the recurrent architecture has lower entropy than the feed forward.

Viewing the relationship between the proportional size of the motif and the amount of shift reveals slightly more complicated behaviour for feed forward architectures. As Table 2 shows, the feed forward network can access the motifs more easily when they are larger. As the motif shrinks relative to the length of the sequence the feed forward network loses access to the patterns. On the other hand, the recurrent network has none of this sensitivity. Apart from outperforming the feed forward network, the behaviour of the recurrent networks remains relatively constant over all combinations.

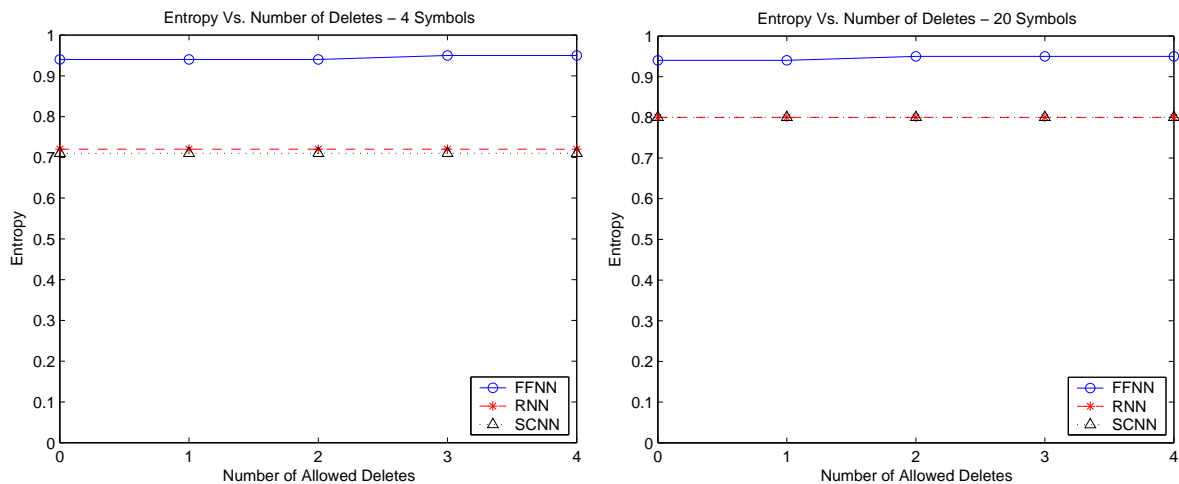


Figure 3: The average entropy within the networks as the number of deletes in the motif is varied. These values are averaged across sequences ranging in length from 50 to 300, with the motifs ranging from one third to an eighth of the sequence. Between one fifth and a twentieth of each motif consisted of wildcards, with no subsets (as in Figure 2). All networks were equipped with 5 state units and $K = 10$ clusters were identified. The graphs show that the difference in bias appears to be independent of the number of deletes, however the exact difference varies with the number of symbols in the alphabet.

Motif-size	Number of Deletes				
	0	1	2	3	4
33%	0.91 (0.72)	0.92 (0.72)	0.92 (0.72)	0.93 (0.72)	0.93 (0.72)
20%	0.94 (0.72)	0.95 (0.72)	0.95 (0.72)	0.96 (0.72)	0.96 (0.72)
13%	0.96 (0.71)	0.96 (0.71)	0.96 (0.71)	0.97 (0.71)	0.97 (0.71)

Table 2: The average entropy of the two network architectures - FFNN (RNN) discriminating between sequences with for varying length of motifs and number of deletes. Sequences varied between 50 and 300 symbols long from an alphabet of 4 symbols. Motifs had between 20% and 5% wildcards and were able to shift from 0 to 6 positions. All networks were equipped with 5 state units and $K = 10$ clusters were identified.

3 Case study: Subcellular localisation by targeting peptides

After proteins have been synthesised, the cellular machinery often relies on an N-terminal peptide to transport the protein to its appropriate subcellular location. These targeting peptides are a diverse group of sequences of varying lengths with a bare minimum of dependable features [21]. This diversity and ambiguity makes the problem ideal for a benchmarking study, illustrating performance on a broad scope of non-trivial pattern recognition problems.

A series of neural network based predictors have shown special abilities in handling the task of predicting biological sequence features relating to subcellular localisation. SignalP, ChloroP and TargetP [7] all predict subcellular targets and cleavage sites of various proteins using simple feed forward networks. The most general of the predictors, TargetP, distinguishes between proteins destined for mitochondria, for chloroplasts, for the secretory pathway (the endoplasmic reticulum), and proteins which lack a targeting peptide. The peptide that indicates that a protein is bound for the secretory pathway has come to be known as a ‘signal peptide’.

By experimenting with various configurations, Emanuelsson *et al.* [8] found that target specific feed forward networks which slide over a limited window of residues can work as targeting peptide detectors. Each network distinguishes between residues that belong to the targeting peptide (to be cleaved off) and those that belong to the mature protein. The detection outputs for the 100 first residues (from each of the target specific networks) are fed into another feed forward network which makes a final decision on which subcellular compartment the protein is destined for (see Figure 4).

We replicate the layer of target specific feed forward networks described by Emanuelsson *et al.* and benchmark their classification errors against a simple recurrent neural network

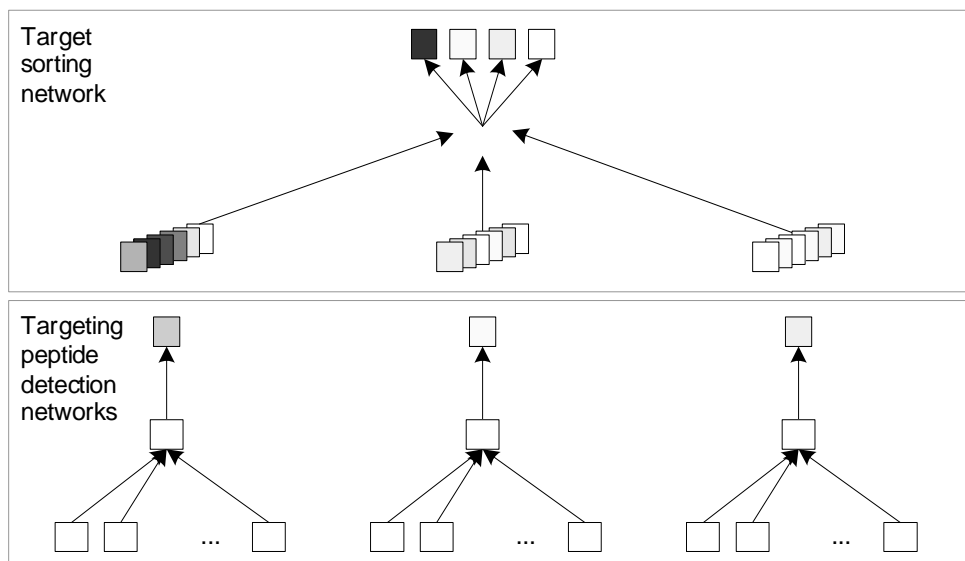


Figure 4: The TargetP neural network architecture. A set of targeting peptide detector networks (one for each target) receive as input a window of residues and output the status of the middle residue. The outputs for each of the peptide detector networks are presented to the target sorting network which outputs the probabilities of the presence of the targeting peptide types (SP, mTP, cTP and the probability of not having a targeting peptide at all).

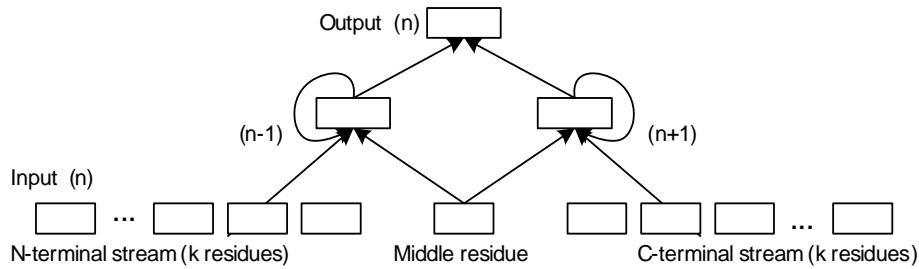


Figure 5: The recurrent targeting peptide detector network operates by traversing the sequence from two directions, accumulating two separate states, until the middle residue is reached, and when the network produces the classification (part of targeting peptide ‘1’ or not ‘0’) at its output. As an example, the symbol G within the sequence ABCDEFGHIJKLM is classified by presenting a window of residues, say 2, from each direction: [AB:-:LM], then [CD:-:JK] and finally [EF:G:HI], where ‘-’ represents a *nil* pattern (all zeros) and ‘:’ indicates node bank boundaries between residues taken from the N-terminal flank, the residue at the point of prediction, and residues taken from the C-terminal flank, respectively. Note that each side of the architecture corresponds to an individual standard, uni-directional recurrent network.

and a sequential cascaded neural network trained under the same conditions. The results offer strong support for the applicability of recurrent architectures to biological sequence problems.

In conventional work with recurrent neural networks, the sequence is processed from one end to the another. For biological sequence processing this is not appropriate: in contrast to language processing, biological sequences have no dominant directionality that would prescribe the direction of processing. The sequence components which provide information about a particular point in a sequence can be in either direction. Baldi *et al.* proposed a bi-directional architecture [4] which consists of two *wheels* that move in toward a central residue from a distance out in either direction. The classification is done once these wheels reach the residues directly beside the target residue, see Figure 5. Each wheel is governed by the same dynamics as a standard, uni-directional recurrent network, with the addition of the input of the central residue (which feeds into both wheels’ states). Thus the recurrent architectures we applied are, in principle, pairs of recurrent networks.

3.1 Data

We use the data set which was used to develop and evaluate TargetP [8]. Each simulation is evaluated by cross-validation, and repeated multiple times with different starting conditions. Details are provided in Section 5.4.

There are two versions of TargetP: one for plants and one for non-plants, each of which has their own data set. The non-plant version is trained to classify sequences as either one of two specific target classes (mitochondria, signal peptides) or as “other”. The plant version is trained to classify sequences into three specific target classes (mitochondria, chloroplast, signal peptides) or “other”.

All sequences are, as in the study presented in Section 2, presented to the networks as one-hot bit-strings. The set element is unique for the amino acid, resulting in a 20-bit vector for each residue in the sequence, mutually orthogonal to all others. A single bit is added to accommodate unknown residues.

3.2 Networks

The TargetP plant version is equipped with three targeting peptide detection networks: one for mitochondrial, one for chloroplast and one for signal peptides. These networks are equipped with an input window of sizes 35, 55, and 31 amino acid residues respectively. Each detection network is also fitted with a hidden layer consisting of four hidden nodes. All networks were reportedly close to optimal with these configurations [8]. Similarly, the TargetP non-plant version has two detection networks: one for mitochondrial and one for signal peptides, fitted with input windows of sizes 35 and 29 residues respectively, and four hidden nodes. Training is performed by presenting each network with a sequence randomly drawn from the training subsets (uniformly over the target classes). The sequence is then processed by training the network to classify each residue as ‘1’ or ‘0’ [8].

The recurrent networks are similarly used to scan and detect targeting peptides. By iteratively creating a state from the residues next to each position in the sequence, the middle residue is classified as being part of the specific targeting peptide or not (see Figure 5). We tried a few configurations and the results reported below are taken from recurrent networks with input windows of $k = 10$ residues both from the N-terminal and the C-terminal flank. States consist of $h = 4$ nodes of which all are fully recurrent (all nodes feed back to all others within the same state layer). As configurations have yet to be fully explored, we do not claim that the reported configuration is optimal. We use the same configuration ($k = 10$ and $h = 4$) for both plant and non-plant data, and for all subcellular targets.

3.3 Results

After 30,000 training sequences have been presented, the actual output for each position in each test sequence is recorded. Moreover, the squared difference between the target output ('1' or '0') and the actual output is used to assess the classification ability of the network. As the cleavage site determines the end of the string of 1's, the error indicates success of both the classification of the peptide and identification of the cleavage point.

We aligned all the output vectors by the cleavage point (for targeting peptides) or at the beginning of the sequence (for proteins without a targeting peptide). We then took a window of thirty outputs around the cleavage point (-23 to 6) and produced a sum of the squared error plot. We also aligned this error plot with a sequence "logo" [17] of the same size window. The logo gives an indication of the kind of sequence features that are present close to the cleavage point.

For each subcellular location we produced two plots. The first plot is shows the error on positive test cases only (e.g. for the secretory pathway, only known signal peptides were tested). In other words the plot gives an indication of how the sensitivity of the network

varies across the sequence. The second plot shows the total mean squared error across the whole data set (including negative sequences), thus demonstrating how reliable the particular network architecture will be overall.

3.3.1 Non-plant proteins

In Figure 6 we see the average error of a window of thirty amino acids around the non-plant signal peptide, 23 within the signal peptide and 7 within the mature protein. The classification error is generally higher around the cleavage site, then drops down to a reasonable level along the hydrophobic region in the middle of the peptide. However, the error for the feed forward network employed by TargetP starts to increase again before the end of the hydrophobic stretch, whereas the recurrent networks sustain reasonably constant error levels. On the other side of the cleavage site the error drops off quickly for both network types. The differences between the specificity and total error plots are subtle, demonstrating that the networks are all performing in a relatively consistent way with both positive and negative testing examples.

In Figure 7 we see a similar analysis of the errors within mitochondrial targeting peptide detection network. The sensitivity of the simple recurrent network is considerably better before and around the cleavage sites of the nascent protein. However the total error close to the cleavage point is worse for the simple recurrent network. The sequential cascaded network, by comparison, performs much more consistently overall, and apart from the slightly higher error after the cleavage point, it offers a clear and reliable improvement over the feed forward architecture.

We can see that for the signal peptide detection network, the sequential cascaded network was unable to fare any better than the improvements gained by the simple recurrent neural network. However, in the mitochondrial targeting peptide task, it provides a consistent improvement in sensitivity, but a variable total error. If we examine the Logo plots

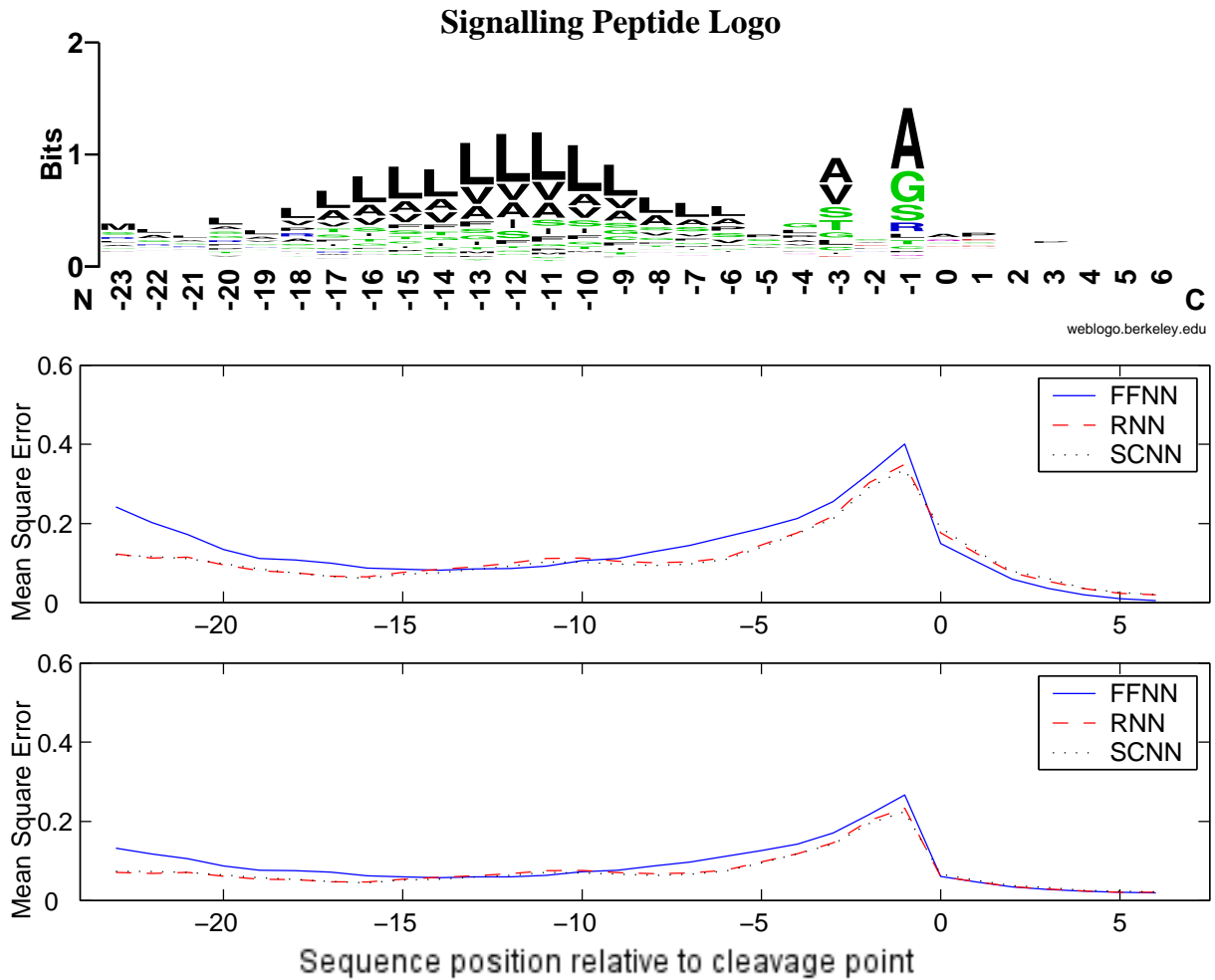


Figure 6: Non-plant signal peptides - Logo aligned with sequence prediction network errors. The top graph shows a sequence aligned logo for non-plant signal peptides. The sequences are aligned by the cleavage points, and the size of letters depicts proportion of sequences with the specific amino acid at each position. The two graphs below the logo show error within the networks that have been trained to recognise sequences of this type. The output vectors of the networks have been similarly aligned with the cleavage points of the signal peptides, or the start of the sequence for negative examples. The middle graph shows sensitivity alone: errors on positive testing instances only. The lower graph shows total error across the entire testing set.

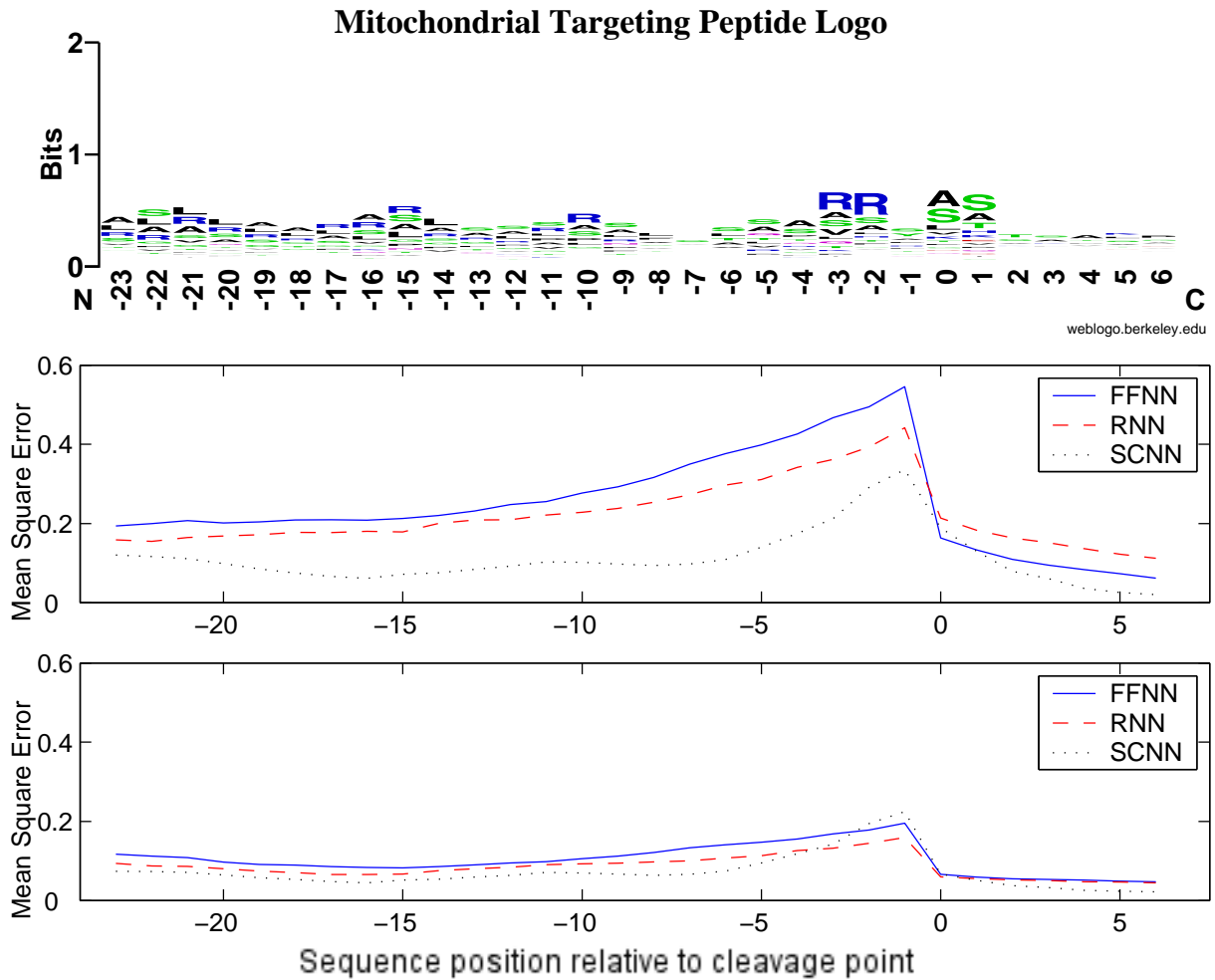


Figure 7: Non-plant mitochondrial targeting peptides - Logo aligned with sequence prediction network errors. The top graph shows a sequence aligned logo for non-plant mitochondrial targeting peptides. The sequences are aligned by the cleavage points, and the size of letters depicts proportion of sequences with the specific amino acid at each position. The two graphs below the logo show error within the networks that have been trained to recognise sequences of this type. The output vectors of the networks have been similarly aligned with the cleavage points of the targeting peptides, or the start of the sequence for negative examples. The middle graph shows sensitivity alone: errors on positive testing instances only. The lower graph shows total error across the entire testing set.

for these two task it becomes obvious that the signal peptide task has a much clearer, reasonably fixed position motif, whereas the mitochondrial targeting peptide has far more ambiguous features for the networks to exploit. The sensitivity plot would tend to indicate that the sequential cascaded architecture has been able to locate subtler patterns within the peptide. However, the total error plot demonstrates that it is more likely that, in the absence of obvious patterns, the individual networks are finding subtle patterns of different kinds. This is a clear indication of bias in action, the architecture of the networks allows them to find and use sequence features of different kinds.

3.3.2 Plant proteins

We performed the same analysis on the plant data set and found that for signal peptides the results were much the same. The only distinct difference in the plots was an absence of a pronounced upward trend in the error for feed forward networks on the N-terminal side of the hydrophobic stretch.

The plant data error plots for the mitochondrial targeting detection networks were likewise not dissimilar from those seen above for non-plant data. The only distinct difference was the absence of a pronounced spike in total error around the cleavage point for the simple recurrent architecture.

In Figure 8 we see an analysis of the errors within the chloroplast targeting peptide detection network. Sensitivity is at its worst for all networks as we approach the cleavage point. Even though both of the recurrent architectures exhibit a slightly higher sensitivity error after the cleavage point, their total error remains below that of the feed forward architecture over the whole window. The sensitivity error of the sequential cascaded network is the worst of all three networks, however its overall error is the best. Indicating that it has sacrificed sensitivity to positive discriminating patterns in order to improve overall performance. This provides further indication that the sequential cascaded network

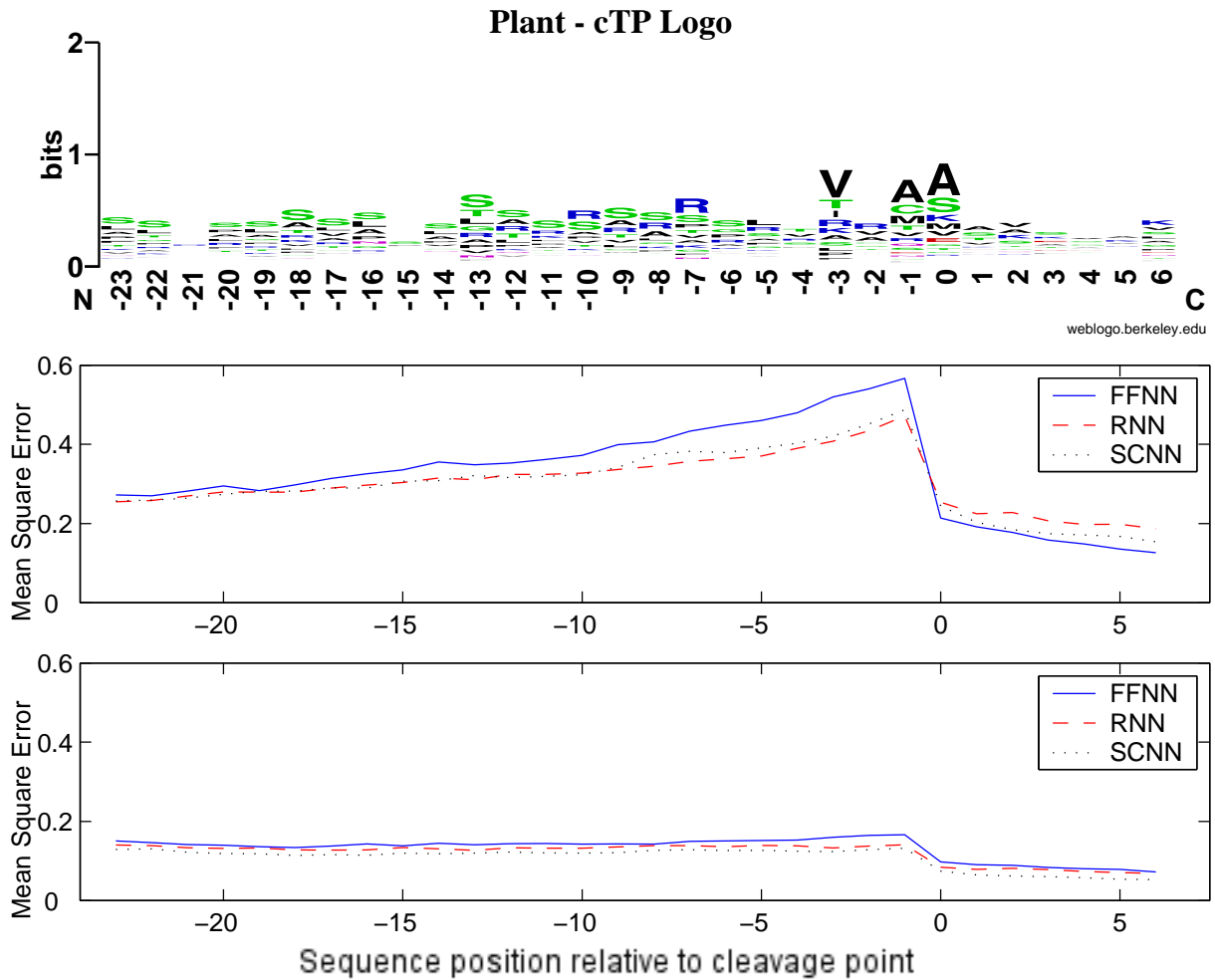


Figure 8: Plant chloroplast targeting peptides - Logo aligned with sequence prediction network errors. The top graph shows a sequence aligned logo for plant chloroplast targeting peptides. The sequences are aligned by the cleavage points, and the size of letters depicts proportion of sequences with the specific amino acid at each position. The two graphs below the logo show error within the networks that have been trained to recognise sequences of this type. The output vectors of the networks have been similarly aligned with the cleavage points of the signal peptides, or the start of the sequence for negative examples. The middle graph shows sensitivity alone: errors on positive testing instances only. The lower graph shows total error across the entire testing set.

Network	Targeting Peptide Task		
	non-plant SP	non-plant mTP	plant cTP
FFNN	0.0877 (0.0074)	0.1025 (0.0030)	0.1318 (0.0651)
RNN	0.0728 (0.0064)	0.0849 (0.0050)	0.1206 (0.0187)
SCNN	0.0720 (0.0081)	0.0922 (0.0119)	0.1082 (0.0208)

Table 3: The average error across the window for the three network architectures (standard deviation across the five runs in brackets)

is a promising architecture for classification and prediction of sequences with ambiguous features.

3.4 Overall results

In Table 3 we see the mean squared errors for each of the networks, averaged across the 30 residue window shown in the previous figures. The general trend is that overall the recurrent architectures provide superior performance. However, on the more difficult tasks the difference between the performance of the recurrent architectures diverges, indicating that they each have differing sensitivity to the requisite patterns.

4 Conclusion

When a problem space is large and sparse (long sequences consisting of a large set of symbols) with a relatively few number of examples in the training set, it is essential for a machine learning algorithm to receive some guidance to “meaningful” patterns. In an biological sequence space, meaningful patterns are constrained by the physical system in which the molecules operate. For example there are length constraints on membrane spanning regions which exclude vast numbers of sequences from a realistic search [10]. By being aware of the manner in which architectural bias can influence search we can design classifiers that deliberately constrain their search to realistic portions of the problem space.

The state space of the network *architecture* reflects the presence of a bias. According to our results, motifs in sequential data that are relatively short, contain some wildcards, deletes and are able to shift their position are more accessible to the recurrent architectures than to the feed forward network. Furthermore, the recurrent network seems to cope with domains which are extremely high-dimensional – the distinct organisation of sequences at the state layer is almost unaffected with an increase of input dimensionality.

The architectural bias study is supported by the examination of testing error for networks trained on the difficult task of recognising amino acids that belong to targeting peptides. First and foremost, these error plots corroborate the point that recurrent architectures are biased toward recognition of features within sequential data. There is a general improvement in accuracy offered by both the recurrent architectures.

The second most striking result from the applied simulations, is that as the features within the target sequence become more ambiguous, then the choice of particular architecture becomes more critical. For example, the peptide with the strongest motif is the signal peptide in non-plant data. When trained on this task both recurrent architectures produced almost identical results. However, the mitochondrial and chloroplast targeting peptides have much less distinct sequence features and we can see from the error plots that all architectures have attuned themselves to different areas within the sequence. In some cases the additional context sensitivity of the sequential cascaded network is of great benefit, in others it seems to be a hindrance.

We conclude that although there have been a small number of applications of recurrent neural networks in the bioinformatics field [4, 16], there are good theoretical [19, 9, 18] and now empirical results that indicate that they will provide improved access to the kinds of patterns that are biologically significant. Furthermore, our protein localisation case study has demonstrated that when the problem involves a reasonable amount of motif conservation, then the particular recurrent architecture seems unimportant. However,

when the important features become more ambiguous then the subtleties of the architecture provide crucial differences in the information that is extracted from the training set. We offer the general heuristic that for difficult problems it is worth exploring the space of recurrent architectures to find a suitable bias.

5 Methods and materials

5.1 Data: Architectural bias simulation

For each simulation we generated 500 sequences each of a pre-specified length L .

$S_j = [s_{j,1}, s_{j,2}, \dots, s_{j,L}]$ denotes the sequence consisting of elements $s_{j,i} \in \Lambda^U$. Λ^U is the set of U symbols, for example $\Lambda^2 = \{\mathbf{A}, \mathbf{B}\}$.

A motif is defined as a sequence $M = [m_1, m_2, \dots, m_N]$ (where N is the length, $N \leq L$). Each $m_i \in \Lambda^{U*}$, where Λ^{U*} is the set of symbols Λ^U plus the wildcard symbol ϵ , matching all symbols in Λ^U .

Furthermore each symbol m_i could be specified as being deletable using a separate delete sequence of identical length to the motif sequence $D = [d_1, d_2, \dots, d_N]$, where each $d_i \in (0, 1)$. On each simulation a specific number of these delete positions were set to 1 while the others were set to zero. The presence of a 1 in some d_i indicates that the corresponding motif symbol m_i may be deleted.

Further to the specification of the motif, each configuration allowed for some movement in the location of the pattern within a sequence. We introduced a parameter a to indicate the amount of allowable shift for the motif, a was typically varied uniformly between 0 and 6 over the simulation.

Each element in a sequence is encoded for presentation to the networks. We use unique one-hot codes (one bit on, rest is off) for all symbols in Λ^U (always resulting in codes that have U bits). This choice of input encoding imparts no bias because the distance between

all symbols is identical, hence the machine can infer no information about the problem space from the encoding.

Each of the bias simulations on synthetic involved a combination of some subset of the following parameters,

- The size of the alphabet (values: 4, 20)
- The length of the sequence (values: 50, 60, 75, 100, 150, 200, 300)
- The size of the motif in proportion to the sequence length (values: $\frac{1}{3}$, $\frac{1}{5}$, $\frac{1}{8}$)
- The proportion of the motif that contains wildcards (values: $\frac{1}{5}$, $\frac{1}{8}$, $\frac{1}{10}$, $\frac{1}{20}$)
- The amount of positional shift allowed (values: 0, 1, 2, 3, 4, 5, 6)
- The number of allowed deletes in the motif (values: 0, 1, 2, 3, 4)

Each and every parameter configuration was tested using 10 differently initialised networks. For configurations that are neutral with respect to some parameters, outcomes for all values of such parameters were collected and the average result is reported.

5.2 Methods: Architectural bias simulations

The architectural bias is estimated using K -means clustering of the hidden node activations after each sequence has been presented to the networks. To do so we need to specify K : the number of codebook vectors. Experimentation showed that as expected the entropy decreased as K increased. We settled on using the value 10 because it provided ample opportunity for separation of the data without generating trivially low entropy results.

K -means is allowed to converge to stable codebook vectors denoted by $V = [\mathbf{v}_1, \dots, \mathbf{v}_K]$. Running the network on a sequence $S_j = [s_{j,1}, \dots, s_{j,L}]$, the state is mapped to a group index, $C(\mathbf{x}_j) \in \{1, 2, \dots, K\}$, where C selects the closest codebook vector by Euclidean distance.

The same procedure is applied to all sequences in $S_j \in \mathcal{S}$. Each group will then contain a subset of the sequences, of which some contain the motif and some do not. For the set of sequences within group k , the number of positives is denoted by p_k and the number of negatives is denoted by n_k . The entropy for each group is denoted by E_k .

$$E_k = -(p_k/(p_k + n_k)) \cdot \log_2(p_k/(p_k + n_k)) - (n_k/(p_k + n_k)) \cdot \log_2(n_k/(p_k + n_k)). \quad (1)$$

The entropy measures the homogeneity within the group, considering the proportion of sequences containing the motif and those that do not. Lower value indicates higher homogeneity, more transparent access to the motif. We report the average entropy over all groups $\sum_k E_k/K$.

5.3 Networks

In general we refer to the neural networks as producing an output vector S_j when presented with a particular sequence as an input vector x_j .

$$\mathbf{w}(S_j) = \mathbf{x}_j \quad (2)$$

However, aside from the architectural differences that influence this function, the two categories of network process the input sequence in a different fashion. A feed forward network accepts its input in one time step according to the size of its input window L . This may be the entire sequence or a subset thereof.

$$\mathbf{x}_j = \mathbf{f}(\phi(s_{j,1})|\dots|\phi(s_{j,L})) \quad (3)$$

where $|$ signifies concatenation of vectors.

The recurrent networks, by comparison, use a smaller input window and process the

sequence by moving this window across the sequence until all of the input has been exposed to the network. The network works its way across the sequence iteratively, continually modifying its internal state in light of the current input and the current state of the network. So if we say that the input sequence can be broken into N windows W then the processing can be defined as taking one window at a time, working recursively from 1 to N (the second index on the inputs in Equation 4).

$$\mathbf{x}_j = \mathbf{g}(\phi(W_{j,1}), \mathbf{g}(\phi(s_{W,2}), \mathbf{g}(\phi(W_{j,3}), \dots))) \quad (4)$$

A null vector terminates the recursion in Equation 4.

The dynamics of the feed forward network can be described with the following formalisations. The network consists of three layers and two mapping functions, one from the input to state (or hidden) nodes and one from the state nodes to the output. In each case the function is identical but has an independent set of weights and biases determined during the training process. This feed forward mapping is standardly described in Equation 5

$$\mathbf{f}(\mathbf{x}) = \sigma(W_F \cdot \mathbf{x} + \mathbf{b}) \quad (5)$$

Where W_F is the weight matrix and \mathbf{b} is the set of biases. The output function is given by σ , a sigmoidal output function (for hidden nodes we use the logistic function, for output nodes we use the softmax function).

The dynamics of the simple recurrent neural networks is identical except for the addition of terms on the input-to-state mapping function, described by Equation 6.

$$\mathbf{g}(\mathbf{x}, \mathbf{z}) = \sigma(W_F \cdot \mathbf{x} + W_R \cdot \mathbf{z} + \mathbf{b}) \quad (6)$$

In Equation 6 \mathbf{z} depicts the activation of the state nodes from the previous time step and W_R is the state-to-state weight matrix for the recurrent connections. Similarly the dynamics of the sequential cascaded recurrent network are modified only at the input-to-state mapping function. They involve addition of a further weight matrix that governs second order combinations of the current input with the state activations, described in Equation 7.

$$\mathbf{g}(\mathbf{x}, \mathbf{z}) = \sigma(W_F \cdot \mathbf{x} + W_R \cdot \mathbf{z} + \Omega_R \cdot \mathbf{\Pi}_{\mathbf{xz}} + \mathbf{b}) \quad (7)$$

In Equation 7 $\mathbf{\Pi}_{\mathbf{xz}}$ is a column vector consisting of all second order combinations of an element of the current input with an element of the state vector, i.e. of length $|\mathbf{x}| \cdot |\mathbf{z}|$. Ω_R is the input-to-state second order weight matrix.

For the architectural bias simulations, all network architectures were initialised with weights randomly drawn from a uniform distribution $[-0.5, 0.5]$.

5.4 Details of the subcellular localisation study

The networks used for detecting residues in targeting sequences are defined as above. The bi-directional recurrent network is a pair of standard uni-directional recurrent networks, one operating from the N-terminal flank and one from the C-terminal flank, each accumulating a state working towards the central residue. In the final step the amino acid found at the central position is encoded into the middle input layer, prior to this point this layer has

null input (see Figure 5).

In all cases, we use the softmax output function and the negative log-likelihood error function. All networks are trained using backpropagation and for the recurrent networks the error is “unfolded” through the sequence both upstream and downstream as described by Baldi *et al.* [4]. For practical reasons the error flow is truncated after 5 steps. For both feed forward and recurrent networks, the learning rate (η) is fixed to 0.01, all weight values randomly initialised with a Gaussian distribution around 0.0 (variance 0.1). By monitoring errors throughout learning, slow convergence and minor fluctuations were noted. However, the consistency of results reported below denies the presence of major learning issues.

Each simulation was evaluated by 5-fold cross-validation: The data set is divided into five subsets (of approximately equal size). Four are used for training the system, the remaining subset is used for testing. The procedure is repeated with randomly initialised networks and the data subsets are shuffled so that each of the five subsets appears as a test set exactly once. Consequently, the five systems are only tested on unseen sequences. The score we use is the aggregate result for all five test sets (over the five systems). All five-fold cross-validated simulations are then repeated five times to ensure that final scores are significant. The reported results are all averages over the five repeats.

Where we report the total testing error in the aligned sequence plots, the negative test cases are aligned by the following procedure. If the sequence has a targeting peptide of a different class, then it is aligned by the cleavage point of this targeting peptide. If the sequence has no targeting peptide at all, then it is aligned at the start of the sequence.

The plant data set consists of 940 proteins (368 mitochondrial, 141 chloroplast, 269 signal peptides and 162 nucleus and cytosolic, referred to as “other”). The non-plant set consists of 2738 proteins (371 mitochondrial, 715 signal peptides and 1652 nucleus and cytosolic).

References

- [1] S. F. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.
- [2] T. Bailey, M. E. Baker, C. P. Elkan, and W. N. Grundy. MEME, MAST, and Meta-MEME: New tools for motif discovery in protein sequences. In J. T. L. Wang, B. A. Shapiro, and D. Shasha, editors, *Pattern Discovery in Biomolecular Data: Tools, Techniques, and Applications*, pages 30–54. Oxford University Press, 1999.
- [3] P. Baldi and S. Brunak. *Bioinformatics : the machine learning approach, Second Edition*. MIT Press, Cambridge, Mass, 2001.
- [4] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15:937–946, 1999.
- [5] M. Christiansen and N. Chater. Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23:157–205, 1999.
- [6] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.
- [7] O. Emanuelsson. Predicting protein subcellular localisation from amino acid sequence information. *Briefings in Bioinformatics*, 3(4):361–376, 2002.
- [8] O. Emanuelsson, H. Nielsen, S. Brunak, and G. von Heijne. Predicting subcellular localization of proteins based on their n-terminal amino acid sequence,. *Journal of Molecular Biology*, 300(4):1005–1016, 2000.
- [9] B. Hammer and P. Tino. Recurrent neural networks with small weights implement definite memory machines. *Neural Comp.*, 15(8):1897–1929, 2003.

- [10] R. Janulczyk and M. Rasmussen. Improved pattern for genome-based screening identifies novel cell wall-attached proteins in gram-positive bacteria. *Infection and Immunity*, 69(6):4019–4026, 2001.
- [11] L. Kall, A. Krogh, and E. L. L. Sonnhammer. A combined transmembrane topology and signal peptide prediction method. *Journal of Molecular Biology*, 338(5):1027–1036, 2004.
- [12] J. F. Kolen. Recurrent networks: State machines or iterated function systems? In M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman, and A. S. Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 203–210, Hillsdale, NJ, 1994. Erlbaum Associates.
- [13] B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002.
- [14] T. M. Mitchell. The need for biases in learning generalisations. In Shavlik and Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1980. Originally published as a Rutgers Report.
- [15] J. B. Pollack. The induction of dynamical recognizers. *Machine Learning*, 7:227, 1991.
- [16] G. Pollastri, D. Przybylski, B. Rost, and P. Baldi. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins*, 47:228–235, 2002.
- [17] T. D. Schneider and R. M. Stephens. Sequence logos: a new way to display consensus sequences. *Nucleic Acids Research*, 18(20):6097–6100, 1990.
- [18] P. Tino, M. Cernansky, and L. Benuskova. Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, 15(1):6–15, 2004.

- [19] P. Tino and B. Hammer. Architectural bias in recurrent neural networks: Fractal analysis. *Neural Comp.*, 15(8):1931–1957, 2003.
- [20] A. Vullo and P. Frasconi. A recursive connectionist approach for predicting disulfide connectivity in proteins. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 67–71. ACM Press, 2003.
- [21] E. J. B. Williams, C. Pal, and L. D. Hurst. The molecular evolution of signal peptides. *Gene*, 253(2):313–322, 2000.