A Family of Feed-Forward Models for Protein Sequence Classification

Sam Blasiak, Huzefa Rangwala, and Kathryn B. Laskey

George Mason University

Abstract. Advances in sequencing have greatly outpaced experimental methods for determining a protein's structure and function. As a result, biologists increasingly rely on computational techniques to infer these properties of proteins from sequence information alone. We present a sequence classification framework that differs from the common SVM/kernel-based approach. We introduce a type of artificial neural network which we term the Subsequence Network (SN) that incorporates structural models over sequences in its lowest layer. These structural models, which we call Sequence Scoring Models (SSM), are similar to Hidden Markov Models and act as a mechanism to extract relevant features from sequences. In contrast to SVM/kernel methods, which only allow learning of linear discrimination weights, our feed-forward structure allows linear weights to be learned in conjunction with sequence-level features using standard optimization techniques.

1 Introduction

Advances in sequencing have greatly outpaced experimental methods for determining a protein's structure as well as its role within the complex network of interactions taking place inside living organisms. As a result, biologists increasingly rely on computational techniques to infer structural and functional properties of proteins from sequence information alone.

Popular and successful approaches for protein classification employ Support Vector Machines (SVM) [9,10,11,17,15,2]. Performance of SVM-based classifiers is highly dependent on the kernel function, which can be difficult to specify and to interpret. Kernel functions often have free parameters that must be set either through cross validation or heuristics. Further, ad hoc techniques are often employed to normalize pre-computed kernels so that the algorithm can learn larger margins between classes.

We present a sequence classification framework that differs from the SVM/kernel-based approach. We construct a type of neural network called a Subsequence Network (SN) that incorporates structural models over subsequences. These structural models, called Sequence Scoring Models (SSMs), are similar to Hidden Markov Models and act as a mechanism to extract relevant features from sequences. Our feed-forward structure allows standard optimization techniques to be used for learning linear discrimination weights in conjunction with sequence-level features. We compare our algorithm against state of the art kernel methods on a set of canonical datasets for structural and functional protein sequence classification.

2 Background

2.1 Support Vector Machines and Kernels

Support Vector Machines (SVMs) are linear classifiers; they assume that an input space, $\mathcal{X}$, can be partitioned by a hyperplane so that positive examples lie on one side of the plane and negative examples on the other. SVMs can capture nonlinear boundaries by mapping data into a transformed space, $\varphi : \mathcal{X} \rightarrow \mathcal{X}'$, where $\mathcal{X}$ is the original input space and $\mathcal{X}'$ is the transformed input space. Instead of computing this mapping directly, we can substitute the inner product between training examples in the transformed space, $\langle \varphi(x_i), \varphi(x_j) \rangle$, with a kernel function, $K(x_i, x_j)$, where $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ and $x_i, x_j \in \mathcal{X}$ [19].

String kernels extend the SVM to problem domains of variable-length sequences and also allow prior knowledge over a particular problem domain to be incorporated into the classifier. Examples of string kernels include the following: The *spectrum kernel* [9] computes, for a pair of sequences, x_i and x_j, the count of subsequences of length k that are present in both sequences. The *mismatch kernel* [10] can be best described as a fuzzy version of the spectrum kernel. For two sequences, x_i and x_j, the mismatch kernel computes the number of subsequences of length k across x_i and x_j that contain at most m mismatches. The *local alignment kernel* (LA-kernel)[17] computes the sum over all possible alignment scores between two sequences. Alignment scores generalize edit distance and score pairs of individual amino acids using a predefined distance matrix, commonly the BLOSUM62 matrix [4]. Profile kernels [15,7] are semi-supervised methods that augment training and test sequences with unlabeled sequences in the Protein Data Bank (PDB).

2.2 Hidden Markov Models

The Hidden Markov Model (HMM) [14] defines a probability distribution over sequences. The HMM assumes: (i) Each symbol in the sequence was generated from a mixture distribution; the mixture components are referred to as hidden states. (ii) The Markov property holds over hidden states i.e., the hidden state generating the current observation depends on the past only through the hidden state of the previous observation.

The joint probability of a sequence, $x_{1:T}$ of length T, and a set of hidden states, $z_{1:T}$, under an HMM is given as follows:

$$p(x_{1:T}, z_{1:T}) = \prod_{t=1}^{T} p(z_t|z_{t-1})p(x_t|z_t) = \prod_{t=1}^{T} \theta^{trans}_{z_{t-1},z_t} \theta^{emit}_{x_t,z_t}, \tag{1}$$

where θ^{trans} is a set of transition probabilities and θ^{emit} is a set of emission probabilities. Detailed descriptions of parameters are given in Table 1. Converting transitions from adjacent hidden states over the length of the sequence to counts of emissions and transitions gives

$$p(x_{1:T}, z_{1:T}) = \prod_{k,k'} \left(\theta^{trans}_{k,k'}\right)^{n^{trans}_{k,k'}} \prod_{k,m} \left(\theta^{emit}_{k,m}\right)^{n^{emit}_{k,m}} \tag{2}$$

Table 1. Description of HMM parameters

K	the number of HMM hidden states
M	the number of possible amino acids
N	the number of protein sequences
T_n	the length of the n^{th} sequence
$\mathbf{x}_n$	is the n^{th} amino acid sequence, $x_{n,t}$ is the t^{th} symbol in the n^{th} sequence
y_n	indicates the category associated with the n^{th} sequence. $y_n \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of all categories
$\mathbf{z}_n$	is the n^{th} sequence of hidden states, $z_{n,t}$ is the value of the hidden state for the n^{th} sequence at position t. $\mathbf{z}_n \in \mathcal{Z}$, where $\mathcal{Z}$ the set of all possible hidden state sequences.
$\theta^{trans}_{k,k'}$	the probability of hidden state k occuring at position t when hidden state k' appears at position $t+1$
$w^{tran}_{k,k'}$	defined as $\log \theta^{trans}_{k,k'}$
$\theta^{emit}_{k,m}$	the probability of emitting symbol m at position t when hidden state at position t is k
$w^{emit}_{k,m}$	defined as $\log \theta^{emit}_{k,m}$
$\mathbf{w}$	defined as $\left[w^{trans}_{1,:} \dots w^{trans}_{K,:} w^{emit}_{1,:} \dots w^{emit}_{K,:}\right]^\top$, a vector comprising both the transition and emission weights - the subscripted ":" is matlab notation for the vector over the relevant index.
$n^{emit}_{k,m}$	the number of times hidden state k occurs in conjunction with observed symbol m
$n^{trans}_{k,k'}$	the number of times hidden state k occurs before hidden state k'

In the logarithm, the joint probability of a sequence and associated hidden states under the HMM is a linear function:

$$\log p(x_{1:T}, z_{1:T}) = \sum_{k,k'} n^{trans}_{k,k'} \log \theta^{trans}_{k,k'} + \sum_{k,m} n^{emit}_{k,m} \log \theta^{trans}_{k,m} \tag{3}$$
$$\overset{\text{def}}{=} \sum_{k,k'} n^{trans}_{k,k'} w^{trans}_{k,k'} + \sum_{k,m} n^{emit}_{k,m} w^{emit}_{k,m}$$

where we define $w \overset{def}{=} \log \theta$ for both emissions and transitions. HMMs are probability distributions and thus require that $\sum_{\mathcal{X},\mathcal{Z}} p(\mathbf{x}, \mathbf{z}) = 1$, where $\mathcal{X}$ indicates the set of all possible sequences with alphabet size M, and $\mathcal{Z}$ indicates the set of all hidden states assignments for a sequence $x_{1:T}$. This constraint is satisfied as long as $\sum_{k'} \theta^{trans}_{k,k'} = 1$ and $\sum_m \theta^{emit}_{k,m} = 1$.

It is often useful to find the maximum probability assignment of values to hidden states, i.e., $\arg\max_{z_{1:T}} p(x_{1:T}, z_{1:T})$. The maximum can be computed efficiently by distributing the addition operator over max function to create the following recurrence:

$$\max_{z_{1:t}} \log p(x_{1:t}, z_{1:t}) = \max_{z_t} \left[\left(\max_{z_{1:t-1}} \log p(x_{1:t-1}, z_{1:t-1}) \right) + \log p(z_t | z_{t-1}) + \log p(x_t | z_t) \right] \tag{4}$$

The algorithm that uses this recurrence to compute the maximum over $z_{1:T}$ is known as the Viterbi algorithm [14].

2.3 Neural Networks

A feed-forward artificial neural network (ANN) is a nonlinear classifier or regression function where the input to output transformation is a composition of differentiable functions: $f^{(H)}(\dots(f^{(1)}(x))\dots)$. We assume a dataset $\{(x_n, y_n)\}_{n=1}^N$, $x \in \mathcal{X}$, $y \in \mathcal{Y}$ where x_n is an input vector associated with an output y_n. We denote the vector valued output of each layer as $\mathbf{f}^{(h)}$, which we call "layer h" of the neural network. The top layer of the network (layer H) is compared against the true output, y, using a loss function $\ell(\mathbf{f}^{(H)}, y)$. In a standard ANN, $\mathbf{f}^{(h)}_i$, the i^{th} element of the vector $\mathbf{f}^{(h)}$, is computed by passing a linear combination of the values from the previous layer through a squashing function (usually the hyperbolic tangent function). Each $\mathbf{f}_i^{(h)}$ is computed independently of $\mathbf{f}_j^{(h)}$, $i \neq j$ given values from layer $h-1$:

$$\mathbf{f}^{(h)} = \left[f_1^{(h)} \left(\mathbf{f}^{(h-1)} \right), \ldots, f_{n_h}^{(h)} \left(\mathbf{f}^{(h-1)} \right) \right]^\top , \tag{5}$$

where n_h is the number of elements in layer h and $f_i^{(h)}$ denotes the composition of squashing and linear functions used to compute the i^{th} element of $\mathbf{f}^{(h)}$.

Convolutional Neural Networks (CNNs) [8] are inspired by neural connections in the human visual cortex. In CNNs, lower levels of the network respond to local portions of the input. For instance, the Time Delay Neural Network (TDNN) [20] is a type of CNN used in speech recognition. In the TDNN, the first hidden layer of the network is computed from a set of overlapping windows of an input sequence i.e., $\mathbf{f}^{(1)}$ is computed from an input sequence $x_{1:T}$:

$$\begin{aligned}\mathbf{f}^{(1)} = \Big[& f_1^{(1)} \left(x_{1:n_{cnv}} \right), f_1^{(1)} \left(x_{2:n_{cnv}+1} \right), \ldots, f_1^{(1)} \left(x_{T-n_{cnv}:T} \right), \ldots, \\ & f_{n_h}^{(1)} \left(x_{1:n_{cnv}} \right), f_{n_h}^{(1)} \left(x_{2:n_{cnv}+1} \right), \ldots, f_{n_h}^{(1)} \left(x_{T-n_{cnv}:T} \right) \Big]^\top \end{aligned} \tag{6}$$

where n_{h-1} indicates the number of elements in the vector $\mathbf{f}^{(h-1)}$ and n_{cnv} is the size of the "convolutional window" (the number of elements from the input, x, which contribute to produce the value of layer 1). We use Matlab slice notation, $x_{i_1:i_2}$, to indicate a sub-vector of the input sequence starting at index i_1 and ending at index i_2.

3 Sequence Classification with Subsequence Networks

Our family of feed-forward classification models are convolutional neural networks that assume a protein's structural or functional category can be predicted by the presence of a set of subsequences. We call these models Subsequence Networks (SN). In a Subsequence Network, the convolutional layer learns the degree to which a subsequence is present in a protein sequence. The degree of presence of a subsequence acts as a feature, which can be input to a linear classification layer, allowing combinations of these subsequence features to be detected.

Convolutional units in the Subsequence Network are structured like HMMs, except that we relax the constraint that the output of each unit defines a probability distribution over sequences. That is, we perform unconstrained optimization with respect to $w_{k,k'}^{trans} \overset{def}{=} \log \theta_{k,k'}^{trans}$ and $w_{k,m}^{emit} \overset{def}{=} \log \theta_{k,m}^{emit}$ rather than enforcing the constraint that each θ vector sums to one. We refer to these unnormalized models as "Sequence Scoring Models" (SSM). Figure 1 shows a diagram of our Subsequence Network using an SSM convolutional layer.

3.1 Pair-SSMs

The Pair-SSM is an unnormalized version of the Pair HMM [5]. Pair HMMs probabilistically extend the concept of edit distance by assigning probabilities

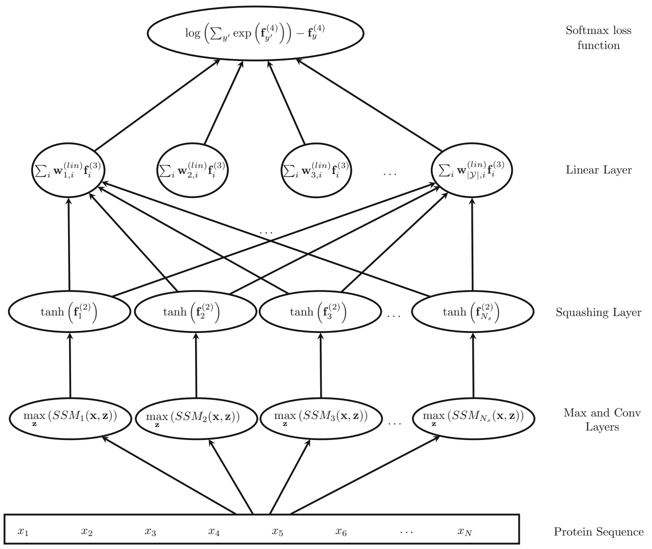

$\log\left(\sum_{y'}\exp\left(\mathbf{f}^{(4)}_{y'}\right)\right) - \mathbf{f}^{(4)}_y$
Softmax loss function
$\sum_i \mathbf{w}^{(lin)}_{1,i}\mathbf{f}^{(3)}_i$
$\sum_i \mathbf{w}^{(lin)}_{2,i}\mathbf{f}^{(3)}_i$
$\sum_i \mathbf{w}^{(lin)}_{3,i}\mathbf{f}^{(3)}_i$
$\sum_i \mathbf{w}^{(lin)}_{|\mathcal{Y}|,i}\mathbf{f}^{(3)}_i$
Linear Layer
$\tanh\left(\mathbf{f}^{(2)}_1\right)$
$\tanh\left(\mathbf{f}^{(2)}_2\right)$
$\tanh\left(\mathbf{f}^{(2)}_3\right)$
$\tanh\left(\mathbf{f}^{(2)}_{N_s}\right)$
Squashing Layer
$\max_{\mathbf{z}}(SSM_1(\mathbf{x},\mathbf{z}))$
$\max_{\mathbf{z}}(SSM_2(\mathbf{x},\mathbf{z}))$
$\max_{\mathbf{z}}(SSM_3(\mathbf{x},\mathbf{z}))$
$\max_{\mathbf{z}}(SSM_{N_s}(\mathbf{x},\mathbf{z}))$
Max and Conv Layers
x_1 x_2 x_3 x_4 x_5 x_6 … x_N
Protein Sequence

Fig. 1. A diagram illustrating a Subsequence Network being applied to an input sequence. In the bottom row of the network, the maximum of the scores from each SSM are taken over the input sequence. The Conv layer is defined by a score from an SSM. In the second row, a squashing function is applied to the maximum SSM scores. The third row computes the distance of these squashed scores from hyperplanes used to define boundaries between sequence categories. Finally, the loss function compares the category given by the hyperplane to the true sequence category.

to a symmetric set of insertions, deletions, and substitutions that allows one sequence from a pair to be created from the other.

The log probability of a pair of sequences, $\mathbf{x}_i$ and $\mathbf{x}_j$, in the Pair HMM can be given by a linear model in the log of the distribution parameters:

$$\log p(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}) = \sum_{k,k'} n^{trans}_{k,k'} w^{trans}_{k,k'} + \sum_{m,m'} n^{emit}_{m,m'} w^{emit}_{m,m'} \tag{7}$$

where $n^{emit}_{m,m'}$ indicates the number of times we substituted an amino acid, m, from sequence $\mathbf{x}_i$ with m' from sequence $\mathbf{x}_j$, and $w^{emit}_{m,m'}$ is the associated cost of this substitution. The expression in Equation 7 differs from the probability of the standard HMM (Equation 3) in that we replace counts of emissions from a hidden state by counts of substitutions of amino acid m from sequence i with amino acid m' from sequence j i.e., $n^{emit}_{k,m}$ becomes $n^{emit}_{m,m'}$ and $w^{emit}_{k,m}$ becomes $w^{emit}_{m,m'}$. The Pair-SSM includes three types of hidden states: In an *Insert* hidden state, the model emits a symbol from sequence $\mathbf{x}_i$. In a *Delete* hidden state the model emits a symbol from sequence $\mathbf{x}_j$. In a *Match* hidden state, the model emits a symbol from both sequences. To allow the model to capture relevant subsequences, we add additional hidden states I_{start} and I_{end}. These hidden states emit symbols of either $\mathbf{x}_i$ or $\mathbf{x}_j$ from a background distribution, allowing the main portion of the Pair-SSM to emit symbols from the relevant subsequence. We show a diagram of hidden state transitions in Figure 2a.

If used in the convolutional layer of a Subsequence Network, the Pair-SSM extracts features using a similar technique as the LA-kernel. In this type of network, subsequences that best match an input sequence are selected from within each training set sequence by the Pair-SSMs. This view is closely related to the empirical kernel map [18,17]. In the empirical kernel map, a feature vector associated with an unknown sequence is given by a vector of kernel evaluations over the training set i.e., we map the sequence x to the vector $[K(x_1, x), \ldots, K(x_N, x)]^\top$, where $K(\cdot, \cdot)$ is the kernel function and $\{x_n\}$, $n \in [1 \ldots N]$ is the set of training sequences. In the first layer of empirical kernel map, a feature vector is computed from an input sequence by evaluating a fixed kernel function on the input sequence paired with each training set sequence. This set of values are then combined linearly using weights, w_n, to produce an overall score for the query sequence: $s(x) = \sum_n w_n K(x_n, x)$. As in all linear classifiers, we classify the query sequence, x, as a member of the positive class if $s(x) > 0$ and as a member of the negative class otherwise. In SVM/kernel classification, kernel evaluations for all pairs of sequences are computed independently. Then, given $K(x_i, x_j)$, $\forall i, j$, the SVM learning algorithm solves a (convex) quadratic program to compute the linear weights, w_n.

Although optimization over the Subsequence Network with a Pair-SSM convolutional layer is tractable, it is not yet practical without distributing computation over multiple processors. For each SGD epoch we must compute the Viterbi paths over N^2 pairs of sequences, x_i and x_j, at a cost of $O(|x_i| \times |x_j|)$ (where N is the number of training sequences and $|x|$ is the length of a sequence).

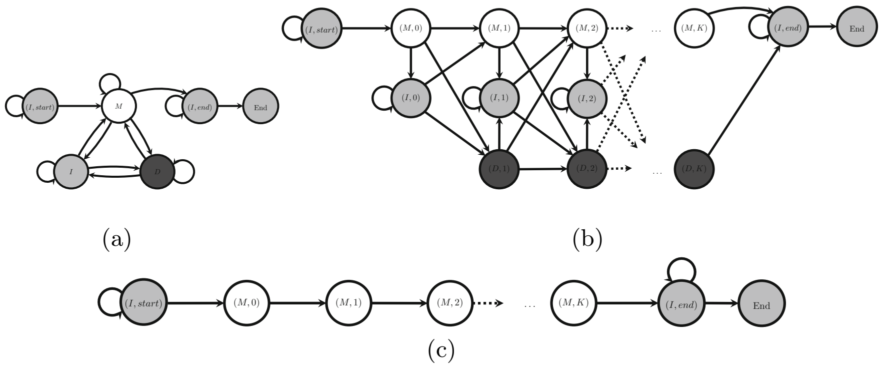


Fig. 2. A diagram of the deterministic finite-state automaton associated with **(a)** the Pair-SSM **(b)** the Local SSM (L-SSM) and **(c)** the Simplified Local SSM (SL-SSM). Match states are indicated with a white background, Insert states with a light-gray background, and Delete states with a dark-gray background.

3.2 Local SSM (L-SSM)

The Local SSM (Figure 2b) is an unnormalized version of the Profile HMM (pHMM) adapted to model a single subsequence within an observed sequence. Profile HMMs [6] are a variation of the standard HMM commonly used for modeling biological sequences. They are left-to-right, non-ergodic HMMs [14] that represent sequences in relation to an archetypal sequence encoded in the emission distributions of the pHMM's hidden states. Profile HMMs use three types of hidden states: (1) *Match (M)* states encode individual symbols of the archetypal sequence, (2) *Insert (I)* states allow additional symbols to be inserted between matched symbols, and (3) *Delete (D)* states allow matched symbols to be skipped. Hidden states of the archetypal sequence are expressed as pairs of symbols (s,k), where $s \in \{M,I,D\}$ indicates a Match (M), Insert (I), or Delete (D), paired with a base state, $k \in [1 \dots K]$, which can be thought of as indexing a symbol in the archetypal sequence. The form of the archetypal sequence is encoded by the emission distributions from each of the K match states. In the local version of the pHMM, which models a single subsequence within the observed sequence, I_{start} and I_{end} are special insert states that allow portions of the sequence before the archetypal sequence to be skipped. We fix transition and emission probabilities from I_{start} and I_{end}, allowing these to be ignored during optimization. In addition, we explicitly include an *End* state to mark the end of the observed sequence. We must include the *End* state because without transition from I_{end} to *End*, the model favors archetypal subsequences positioned near the beginning of the observed sequence. As in the standard pHMM, in our L-SSM, emissions occur only from Match and Insert states.

3.3 Simplified Local SSM (SL-SSM)

Like the L-SSM, the SL-SSM models relevant subsequences within a set of sequences. The SL-SSM (Figure 2c) simplifies the L-SSM by removing Insert and Delete states from the model. This change in the model results in contiguous subsequences of match states. This structural change speeds inference i.e., the highest scoring set of hidden states can be efficiently computed by sliding the window of K match states over a sequence and returning the position with the highest probability. We use the same parametrization of hidden states as in the L-SSM: $\{I_{start}, (M,1), \dots, (M,K), I_{end}, \text{END}\}$, where the pair (M,k), indicates a hidden state that emits the k^{th} symbol of the archetypal sequence. Transitions from the Match state (M,k), $k < K$ to $(M,k+1)$ and transitions from the state (M,K) to the state I_{end} occur with probability one.

The L-SSM or SL-SSM convolutional layer in the Subsequence Network can be interpreted as a simplification of the SN with a Pair-SSM convolutional layer. This simplification is motivated by two assumptions about the domain of protein sequences: (i) the set of protein sequences lies on a lower-dimensional manifold within the sequence space and (ii) the basis given by the training set spans the manifold of our domain and is redundant. With these assumptions it becomes reasonable to simplify the Pair-SSM convolutional layer by creating a model with a lower-dimensionality basis independent of the training examples. For our model, we choose this basis to be a fixed set of L-SSMs or SL-SSMs.

When we replace Pair-SSMs with (S)L-SSMs, additional computational efficiencies become possible because the (S)L-SSM allows us to store only the locally relevant pattern rather than an entire sequence. Computing the score of a sequence under an (S)L-SSMs where hidden states are restricted to small, fixed lengths therefore requires less computation time than evaluating the Pair-SSM between pairs of sequences. A disadvantage of these simplifications is that new parameters are added to the model. In particular, the number of (S)L-SSMs and the number of hidden states for each (S)L-SSM must be specified in advance. In the results section, we show that these simplifications not only maintain much of the accuracy of the Pair-SSM layer, but they are also robust to variations in parameter choices.

3.4 Subsequence Network Objective Function

The objective function for our model includes a loss for each sequence in the dataset and a regularization term that penalizes large parameter values:

$$F(\mathbf{x}) = \sum_n \ell\left(\mathbf{x}_n, y_n; \mathbf{w}^{all}\right) + \frac{\lambda}{2}||\mathbf{w}^{all}||^2 \tag{8}$$

where $\mathbf{w}^{all}$ is an agglomeration of all of the SSM weight vectors in the model (the composition of $\mathbf{w}^{all}$ varies depending on which type of SSM is used for the convolutional layer) and linear combination weights associated with each SSM; $\ell(x, y; \mathbf{w}^{all})$ is a loss function. The λ term determines the trade off between the loss and magnitude of the weights.

A loss function compares the output of a CNN with the label of a single protein sequence. In our model, we use the a softmax loss, shown in the first row of Table 3. The output of the network is given by

$$\mathbf{f}^{(4)} \overset{def}{=} f^{(4)}\left(f^{(3)}\left(f^{(2)}\left(f^{(1)}\left(\mathbf{x}_n\right)\right)\right)\right) \overset{def}{=} \text{Lin}\left(\text{Tanh}\left(\text{Max}\left(\text{Conv}\left(\mathbf{x}_n\right)\right)\right)\right) \tag{9}$$

where *Conv* is a convolutional layer containing multiple convolutional units, *Max* computes the maximum over the responses of each convolutional unit, *Tanh* is the hyperbolic tangent function and maps values in the range $(-\infty, \infty)$ to $(-1, 1)$, *Lin* is a linear layer. As in Section 2.3, we denote the output from hidden layer h as $\mathbf{f}^{(h)}$. Table 3 gives the full form of each layer of the network, and Table 2 gives a description of network parameters.

If the number of hidden states for each SSM, $|\mathcal{Z}^{(i)}|$, across an individual network is the same $\left(|\mathcal{Z}^{(i)}| = |\mathcal{Z}^{(j)}| \quad \forall i, j\right)$ layer $\mathbf{f}^{(1)}$ becomes a matrix of size $N_s \times |\mathcal{Z}|$, where N_s is the number of SSMs in the convolutional layer. In the L-SSM and

Table 2. Subsequence Network parameters

Parameter	Definition
N_s	number of SSMs in the convolutional layer
$\mathbf{f}^{(h)}$	the vector of values that comprise hidden layer h
$\mathbf{w}^{all}$	a combined vector of all SSM and linear weights, $\left[\left(\mathbf{w}^{(lin)}\right)^\top, \left(\mathbf{w}^{(1)}\right)^\top, \ldots, \left(\mathbf{w}^{(N_s)}\right)^\top\right]^\top$
$\mathbf{w}^{(lin)}$	linear weights that define linear boundaries between dataset categories, $\mathbf{w}^{(lin)} = \left[w_{1,1}^{(lin)}, \ldots, w_{1,N_s}^{(lin)}, \ldots, w_{\lvert\mathcal{Y}\rvert,1}^{(lin)}, \ldots, w_{\lvert\mathcal{Y}\rvert,N_s}^{(lin)}\right]$
$\mathbf{w}^{(i)}$	a vector of transition and emission weights for the i^{th} SSM, $\mathbf{w}^{(i)} = \left[\left(w_{1,:}^{trans}\right)^{(i)}, \ldots, \left(w_{K,:}^{trans}\right)^{(i)}, \left(w_{1,:}^{emit}\right)^{(i)}, \ldots, \left(w_{K,:}^{emit}\right)^{(i)}\right]^\top$ where K is the number of hidden states in the SSM
$\mathcal{Z}^{(i)}$	the set of hidden states for the i^{th} SSM

Table 3. The table above describes the composition of each layer in the Subsequence Network and gives an expression for the Jacobian with respect to the layer's input. The values of each layer are given by the vector $\mathbf{f}^{(h)}$ for hidden layer h. The Jacobian of the first layer (Conv) with respect to the input is not used during inference.

Layer	Definition	Jacobian with respect to $\mathbf{f}^{(h)}$
$\ell\left(\mathbf{f}^{(4)}, y\right)$	$\log \sum_i \exp\left(\mathbf{f}_i^{(4)}\right) - \mathbf{f}_y^{(4)}$	$\begin{bmatrix} \frac{\exp\left(\mathbf{f}_1^{(4)}\right)}{\sum_i \exp\left(\mathbf{f}_i^{(4)}\right)}, \dots, \frac{\exp\left(\mathbf{f}_{\lvert\mathcal{Y}\rvert}^{(4)}\right)}{\sum_i \exp\left(\mathbf{f}_i^{(4)}\right)} \\ \vdots \\ \frac{\exp\left(\mathbf{f}_1^{(4)}\right)}{\sum_i \exp\left(\mathbf{f}_i^{(4)}\right)}, \dots, \frac{\exp\left(\mathbf{f}_{\lvert\mathcal{Y}\rvert}^{(4)}\right)}{\sum_i \exp\left(\mathbf{f}_i^{(4)}\right)} \end{bmatrix} - I$
$\mathbf{f}^{(4)} \equiv \mathrm{Lin}\left(\mathbf{f}^{(3)}\right)$	$\left[\sum_{i=1}^{N_s} w_{1,i}^{(lin)} \mathbf{f}_i^{(2)}, \dots, \sum_{i=1}^{N_s} w_{\lvert\mathcal{Y}\rvert,i}^{(lin)} \mathbf{f}_i^{(2)}\right]^\top$	$\begin{bmatrix} w_{1,1}^{(lin)}, \dots, w_{\lvert\mathcal{Y}\rvert,1}^{(lin)} \\ \vdots \\ w_{1,N_s}^{(lin)}, \dots, w_{\lvert\mathcal{Y}\rvert,N_s}^{(lin)} \end{bmatrix}$
$\mathbf{f}^{(3)} \equiv \mathrm{Tanh}(\mathbf{f}^{(2)})$	$\left[\tanh\left(\mathbf{f}_1^{(2)}\right), \dots, \tanh\left(\mathbf{f}_{N_s}^{(2)}\right)\right]^\top$	$\left[d\tanh\left(\mathbf{f}_1^{(2)}\right), \dots, d\tanh\left(\mathbf{f}_{N_s}^{(2)}\right)\right]^\top I$
$\mathbf{f}^{(2)} \equiv \mathrm{Max}\left(\mathbf{f}^{(1)}\right)$	$\begin{bmatrix} \max\left(\left[f_{1,1}^{(1)}, \dots, f_{1,\lvert\mathbf{Z}^{(1)}\rvert}^{(1)}\right]\right) \\ \vdots \\ \max\left(\left[f_{N_s,1}^{(1)}, \dots, f_{N_s,\lvert\mathbf{Z}^{(N_s)}\rvert}^{(1)}\right]\right) \end{bmatrix}$	$\begin{bmatrix} \mathbb{I}\left(f_{1,1}^{(1)} = \max\left(f_{1,:}^{(1)}\right)\right), \dots, \mathbb{I}\left(f_{N_s,1}^{(1)} = \max\left(f_{N_s,:}^{(1)}\right)\right) \\ \vdots \\ \mathbb{I}\left(f_{1,\lvert\mathcal{Z}^{(1)}\rvert}^{(1)} = \max\left(f_{1,:}^{(1)}\right)\right), \dots, \mathbb{I}\left(f_{N_s,\lvert\mathcal{Z}^{(1)}\rvert}^{(1)} = \max\left(f_{N_s,:}^{(1)}\right)\right) \end{bmatrix}$
$\mathbf{f}^{(1)} \equiv \mathrm{Conv}(x)$	$\begin{bmatrix} \mathrm{SSM}_1\left(x, \mathbf{z}_1^{(1)}\right), \dots, \mathrm{SSM}_1\left(x, \mathbf{z}_{\lvert\mathcal{Z}^{(1)}\rvert}^{(1)}\right) \\ \vdots \\ \mathrm{SSM}_{N_s}\left(x, \mathbf{z}_1^{(N_s)}\right), \dots, \mathrm{SSM}_{N_s}\left(x, \mathbf{z}_{\lvert\mathcal{Z}^{(N_s)}\rvert}^{(N_s)}\right) \end{bmatrix}$	

Pair-SSMs, $|\mathcal{Z}|$ is exponential in the size of the input sequence. We make the feed-forward and backpropagation steps for the network tractable by computing the composition Max(Conv(x)) directly using the Viterbi algorithm [14]. The locations of the non-zero indicator functions in the Jacobian of the Max layer are then given by the Viterbi path [14] through the SSM.

Our Subsequence Network incorporates the hyperbolic tangent squashing function, $\tanh(x) = \frac{e^{-2x}+1}{e^{-2x}-1}$. We denote the derivative of this function with respect to the hyperbolic tangent input as $d\tanh(x) = 1 - \tanh^2(x)$. In the Max layer, the function $\max(\boldsymbol{v})$ returns the largest scalar element of the vector $\boldsymbol{v}$.

3.5 Training Subsequence Networks

Training is performed using the stochastic gradient descent (SGD) algorithm. In SGD, the gradient of the objective is evaluated for each training example. The gradient is then scaled by a learning rate and subtracted from the current set of parameters to obtain a new set of parameters. This procedure contrasts with batch gradient learning where the gradient is computed for the entire set of training examples. We compute gradients using the backpropagation procedure [16]. Our model includes a locally non-smooth Max function, causing the the gradient of the objective to be undefined at the non-smooth points. To deal with this potential issue, we skip the gradient update in these non-smooth areas [3].

SGD updates take the form

$$\mathbf{w}_t^{all} \leftarrow \mathbf{w}_{t-1}^{all} - \eta_t \frac{\partial F(\mathbf{x}_n)}{\partial \mathbf{w}^{all}} \tag{10}$$

where $F(\mathbf{x}_n) = \ell\left(\mathbf{x}_n, y_n; \mathbf{w}^{all}\right) + \frac{\lambda}{2}\left|\left|\mathbf{w}^{all}\right|\right|^2$ is the objective for a single sequence, t indicates the iteration number in the SGD algorithm, $\mathbf{w}_t^{all}$ indicates the value of the weights at the t^{th} iteration, and η_t is the learning rate at iteration t and has the form $\eta_t = \eta_0 (1 + \lambda \eta_0 t)^{-1}$, where λ is the regularization parameter.

The gradient with respect to the linear weights is given by

$$\frac{\partial F(x_n)}{\partial w_{yi}^{(lin)}} = -\frac{\partial \ell \left(\mathbf{x}_n, y_n; \mathbf{w}^{all} \right)}{\partial \mathbf{f}_y^{(4)}} \mathbf{f}_i^{(3)} \tag{11}$$

This leads to an update where w_{yi} (the linear weight associated with the i^{th} SSM and category y) is increased if the current training example matches the weight's category and decreased otherwise. The change in the weight's value is proportional to, $\mathbf{f}_i^{(3)}$, the squashed response of the i^{th} SSM. The expression for $\frac{\partial \ell \left(\mathbf{x}_n, y_n; \mathbf{w}^{all} \right)}{\partial \mathbf{f}_y^{(4)}}$ is given in Table 3.

The gradient with respect to the SSM weights is given by

$$\frac{\partial F(x_n)}{\partial \mathbf{w}^{(i)}} = \frac{\partial \ell \left(x_n, y_n \right)}{\partial \mathbf{f}_i^{(3)}} \frac{\partial \mathbf{f}_i^{(3)}}{\partial \mathbf{w}^{(i)}} \tag{12}$$

where

$$\frac{\partial \ell}{\partial \mathbf{f}_i^{(3)}} = \left(\sum_{y \in \mathcal{Y}} \frac{\exp \left(\sum_{i=1}^{N_s} w_{yi}^{(lin)} \mathbf{f}_i^{(3)} \right) w_{yi}^{(lin)}}{\sum_{y' \in \mathcal{Y}} \exp \left(\sum_{i=1}^{N_s} w_{y'i}^{(lin)} \mathbf{f}_i^{(3)} \right)} - w_{y_n i}^{(lin)} \right) \tag{13}$$

The gradient of $\mathbf{f}_i^{(3)}$ with respect to the SSM weights, $\mathbf{w}^{(i)}$, is given by $\frac{\partial \mathbf{f}_i^{(3)}}{\partial \mathbf{w}^{(i)}} = d\tanh \left(\text{SSM}_i \left(x_n \right) \right) \mathbf{n}_{\mathbf{z}_{max}}^{(i)}$ where $\mathbf{n}_{z_{max}}^{(i)}$ is a vector of counts of emissions and transitions associated with the set of hidden states that maximizes the value of $\text{SSM}_i \left(\mathbf{x}_n, \mathbf{z} \right)$ and $d\tanh$ is the derivative of the $\tanh$ function with respect to its input. As in the HMM, $\mathbf{z}_{max}$ for the SSM can be computed efficiently with the Viterbi algorithm [14]. Gradient steps therefore change $\mathbf{w}^{(i)}$ in proportion to the counts of emissions and transitions in the highest scoring set of hidden states. The factor of proportionality is $\frac{\partial \ell}{\partial \mathbf{f}_i^{(3)}}$. This factor can be viewed as a measure of how much the i^{th} linear weight of the ground truth class, $w_{y_n i}^{(lin)}$, differs from its expected value.

4 Experiments

We perform classification experiments on four protein datasets. Of these datasets, two are derived from the Structural Classification of Proteins (SCOP) [12] version 1.53. SCOP is a database that categorizes proteins with known structure into a hierarchy with levels denoted by class, fold, superfamily, and family, from broadest level to the most narrow respectively. The first structural dataset [11], denoted by SF, defines 54 fixed superfamily partitions. The second dataset [15], FD, consists of 23 predefined partitions at the fold level. Both of the SCOP datasets were constructed so that no overlap between lower levels in the hierarchy occurs between training and test sets.

The other protein datasets divide sequences into functional, rather than structural, categories. The enzyme classification dataset [13], which we refer to as EC, contains sequences from six enzyme categories and a set of non-enzymes for a total of 7 one-versus-rest datasets. The fourth dataset [13] categorizes proteins

Table 4. Datasets Sizes - # Train indicates the average number of sequences in the training set over all categories, # Test indicates the average number of test set sequences, and # Categories indicates the number of one-versus-rest classification problems defined by the dataset

Dataset	# Train	# Test	# Categories
SF	2948	1366	54
FD	2196	2155	23
EC	379	110	7
GO	115	57	23

by Gene Ontology. We refer to this dataset as GO. Information about the protein datasets is given in Table 4.

4.1 Comparative Classifiers

We compared the three SVM string kernels to our Subsequence Network. The BLAST kernel was computed by performing a BLAST [1] database search on each sequence. If another sequence from the training set was returned by the BLAST search, then we set the corresponding Kernel value to the returned E-value. The mismatch kernel was described in Section 2.1 and has two parameters. We denote a mismatch evaluation by Mismatch(k, m), where k is the subsequence length and m is the number of allowable mismatches. The LA-Kernel was also described in Section 2.1. For all experiments, the LA-Kernel's temperature parameter, β, was set to 0.2.

4.2 Models and Parameters

We compared three variations of our Subsequence Network. In the first variation, "Pair-SSM," the convolutional layer consisted of Pair-SSMs associated with each training sequence in the model. Similarly, the "L-SSM" and "SL-SSM" variations use L-SSMs and SL-SSMs in the convolutional layer respectively.

For the Pair-SSM network, we initialized pairwise weights using a scaled version of the BLOSUM62 matrix [4] and ran inference for 5 epochs on the FD dataset and 10 epochs on the EC and GO datasets. We set the precision parameter associated with the Gaussian regularizer to $\lambda = .005$. We set multiplicative factor in the learning rate (Section 3.5) $\eta_0 = .1$ for the linear weights. For the Pair-SSM parameters, we set $\eta_0 = \frac{.1}{10 \times (\#\text{Train})}$, where # Train is the number of training set sequences. To allow training of the Pair-SSM model to take place in a reasonable amount of time, we distributed gradient computations of SSMs in the convolutional layer within each backpropagation step over 50 machines[1].

Choices of parameters were the same for the L-SSM and SL-SSM networks: We used 96 SSMs in the convolutional layer and set K (the number of states

[1] Training the (S)L-SSM networks was significantly faster than the Pair-SSM network. To give a rough comparison, SL-SSM network training with the parameters described was faster than computing the Mismatch(5,2) kernel.

for each SSM) to 11. We set the precision parameter to $\lambda = .005$ for both SSM weights and linear weights, and we set $\eta_0 = .1$. For each experiment, we ran inference for 30 epochs.

The weight vector for these models were set by generating subsequences, x, of length K uniformly and at random. For position k in the subsequence, weight w_{kx_k} was set to $\frac{1}{K}$ and weights w_{km}, $m \neq x_k$ was set to $-\frac{1}{K}$.

To compensate for unbalanced numbers of positive and negative examples, we oversampled the positive training set so that the same number of positive and negative examples were presented to the SGD trainer during each epoch. In the (S)L-SSM networks, we found that initializing the linear weights so that half of the SSMs were associated with the positive class and the other half associated with the negative class improved performance of our algorithm.

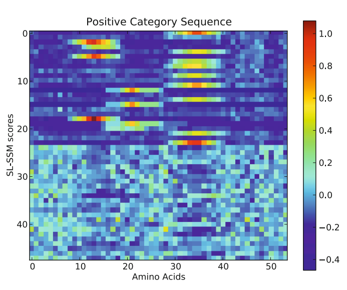
4.3 Evaluation Metrics

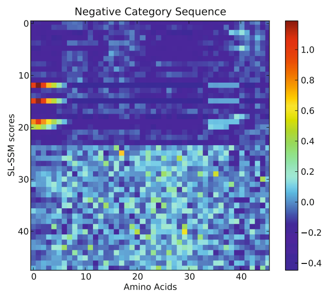
We measured the performance of our algorithm by computing the average ROC scores for each of the one-versus-rest classification problems defined by our datasets. ROC, also known as Area Under the Curve (AUC), is defined as the area under the receiver operating characteristic (ROC) curve. The ROC curve plots the percentage of true positives against the percentage of false positives.We also report ROC$_{50}$ and ROC$_{10\%}$ which are the area under the ROC curve excluding all but the top 50 negative examples or 10 percent of the negative examples respectively. To compare the performance of different models, we performed the Wilcoxson signed rank tests at a 5% significance level using each one-versus-rest category. We report ROC results based on the algorithm's scoring of sequences on the test sets.

4.4 Synthetic Experiments

We constructed a synthetic dataset to verify that our network can detect the relevant subsequence features that we propose will lead to good protein sequence classification performance. Specifically, we generated 1000 sequences with lengths generated from a Poisson distribution with a mean of 50 symbols. Each sequence contained between one and three fixed relevant subsequences, with the positive class containing all three subsequences and the negative class containing either one or two sequences of any type. The relevant subsequences were arranged in random order within the sequence. After placement of the relevant sequences, noise was added - we replaced each every relevant subsequence amino acid with a random amino acid with 10% probability. Amino acids outside relevant subsequences were generated from a uniform multinomial distribution.

Figure 3 shows responses of the lowest layer in the SL-SSM network on two example sequences from the synthetic dataset. Strong responses in portions of the sequences that contain relevant subsequences indicate that our model is able to effectively learn features that discriminate well for a dataset where categories are determined based on the presence of subsequences. The SL-SSM achieves an ROC of .9998 and ROC$_{50}$ of .997 on the test portion of the synthetic dataset, showing that after detecting relevant subsequences, our model has the ability





(a) (b)

Fig. 3. The figures above show responses from each of the 48 unnormalized SL-SSMs over each length 7 subsequence for a sequence generated from the positive class (a) and the negative class (b). The positive example contains all three relevant subsequences while the negative example contains only one relevant sequence. The first 24 SL-SSMs (top half both figures) were constrained to be associated with the positive category, while the last 24 were constrained to be associated with the negative category. The heat maps show that sets of positive SL-SSMs have adapted to each of the three relevant subsequences in the synthetic dataset - both the three relevant subsequences in the positive example and the one relevant subsequence in the negative example were detected by subsets of the first 24 SL-SSMs. In contrast, SL-SSMs associated with the negative category learn a background distributions of symbols.

to effectively classify sequences generated according to a relevant subsequences assumption on the dataset.

4.5 Parameter Adjustment

We adjusted the parameters of (S)L-SSM models by comparing both the number of SSMs in each network and the number of hidden states for the SSMs (we used the same number of hidden states for all SSMs) in experiments on the FD dataset. Table 5 shows results from these comparisons. Within a relatively wide range

Table 5. Average ROC results for different settings of the SL-SSM network on the FD dataset. ROCs were averaged over ten independent trials initialized with random pattern weights. When varying the number of SL-SSM hidden states in (a), 96 SL-SSMs were used in the network. In (b), 11 hidden states were used for each SL-SSM when varying the number of SL-SSMs.

Hidden States	ROC	ROC$_{50}$
7	.801	.146
9	.813	.145
11	.815	.153
13	.815	.153
15	.807	.163

(a)

# SL-SSMs	ROC	ROC$_{50}$
64	.812	.144
80	.812	.156
96	.814	.153
112	.816	.145
128	.816	.147

(b)

of parameter settings, the performance of the model stays roughly the same, showing that, although our method may require some adjustment using cross validation, micromanagement of parameter settings is not critical to maintaining acceptable performance. For the results shown in Table 6, we selected parameter settings for the other experiments ($K = 11$ with 96 convolutional units) that performed best on FD.

4.6 Protein Classification Experiments

We compared four subsequence models used in the convolutional layer of our Subsequence Network. The lower rows of Table 6a and b show results for the L-SSM and the SL-SSM. These indicate that the simpler model, the SL-SSM outperforms the L-SSM. Superior performance of the SL-SSM results from unstable inference in the L-SSM when attempting to learn insert transition weights. If these weights grow above zero for the positive class, then the model tends to explain every protein sequence using long sequences of insert states. For this reason, we fix the insert transition weights in the L-SSM to small negative values ($-\frac{1}{2K}$ where K is the number of Match states in the L-SSM), but this fix affects the flexibility of the model. On the FD dataset, the Pair-SSM outperforms our other models in both ROC and ROC$_{50}$. However, on the functional datasets, both L-SSM and SL-SSM outperform the Pair-SSM. These results indicate that the simpler (S)L-SSM assumptions may be better models of protein structure in certain cases.

Table 6 also compares our subsequence networks to SVM/kernel methods from the literature. Compared to the LA-kernel on the FD dataset, the Pair-SSM model is statistically equivalent in both ROC and ROC$_{50}$ measurements. Compared to the Mismatch kernel on the FD dataset, the Pair-SSM model performs better in ROC but is equivalent for ROC$_{50}$. We note, however, that the β parameter used for the LA-kernel is the best of many settings on both the SF and FD training/test set split. Due to this extensive adjustment on the FD dataset, it is likely that the LA-kernel overfits. In contrast, Pair-SSM network performance is only weakly dependent on parameter settings (the regularization parameter) and we did not perform extensive adjustment of these values, so overfitting likely to be less problematic for our Pair-SSM on the FD test set. The Pair-SSM is equivalent to both the LA-kernel and Mismatch kernel on the EC dataset in both ROC and ROC$_{10\%}$. On the GO dataset, the the Pair-SSM is outperformed by the LA-kernel in both ROC and ROC$_{10\%}$ but is equivalent to the Mismatch kernel.

In ROC, the LA-kernel outperforms the L-SSM and SL-SSM models on the SF and GO datasets but is statistically equivalent for the FD and EC datasets. In ROC$_{50}$, the LA-kernel outperforms the L-SSM and SL-SSM on both the SF and FD datasets at a 5% significance level. In ROC$_{10\%}$, both of our methods outperform the LA-kernel on the EC dataset. On the GO dataset, the LA-kernel's performance is statistically equivalent to the SL-SSM but outperforms the L-SSM. Compared to the Mismatch(5,2) kernel, both the L-SSM and SL-SSM models have equivalent performance in SF and FD in both ROC and ROC$_{50}$. Our algorithms outperform the Mismatch kernel in ROC$_{10\%}$ on both of the functional datasets.

Table 6. ROC results for the FD and SF datasets (a) and the EC and GO datasets (b). Because our model is non-convex, we report means and standard deviations of ROCs from multiple starting points in the SSM weight space. Ten trials were averaged for both the L-SSM and SL-SSM models for both structural and functional datasets. Due to the length of P-SSM network's runtime, we report results from only a single trial.

Dataset	SF		FD	
Method	ROC	ROC_{50}	ROC	ROC_{50}
BLAST	.756	.341	.603	.100
Mismatch (5,1)	.872	.403	.802	.146
Mismatch (5,2)	.888	.479	.782	.188
LA-Kernel($\beta = .2$)	.926	.663	.805	.216
Pair-SSM	-	-	0.826	0.168
L-SSM	0.901 ± 0.003	0.456 ± 0.013	0.804 ± 0.006	0.141 ± 0.010
SL-SSM	0.903 ± 0.003	0.503 ± 0.012	0.815 ± 0.009	0.153 ± 0.008

(a)

Dataset	EC		GO	
Method	ROC	$ROC_{10\%}$	ROC	$ROC_{10\%}$
Mismatch (4,1)	.580	.007	.740	.238
Mismatch (5,2)	.581	.008	.747	.245
LA-Kernel($\beta = .2$)	.574	.004	.801	.339
Pair-SSM	.544	.021	.737	.264
L-SSM	0.587 ± 0.018	0.081 ± 0.011	0.736 ± 0.004	0.280 ± 0.010
SL-SSM	0.583 ± 0.017	0.092 ± 0.024	0.731 ± 0.011	0.306 ± 0.018

(b)

For the EC dataset, none of the algorithms perform particularly well. It is possible that both the size of the dataset and weak correlations between sequence and function cause subsequence-based approaches to fail. A class of methods based on a different set of assumptions may be necessary to achieve strong functional classification performance.

5 Conclusions

The empirical kernel map applied in conjunction with SVM classifiers is strongly related to feed-forward models like convolutional neural networks. Based on this relationship, we show how to construct a family of models, which we call Subsequence Networks, where kernel parameters can be learned in conjunction with linear classification boundaries. Our Subsequence Networks operate differently from state-of-the-art protein sequence classification models yet can achieve comparable performance. We hope that Subsequence Networks can shift the focus in biological sequence classification from increasingly fine-tuned kernel methods toward developing structures with self-tuning abilities.

Our networks also contribute to existing neural network literature by extending the convolutional layer to a maximization over latent parameter spaces in standard sequence models. The effectiveness of this framework for protein sequence classification shows that it has potential in other classification domains.

A straightforward and potentially useful modification to our networks involves adapting them to take Psi-BLAST profiles, rather than sequences, as input. This shift to a semi-supervised structure has the potential to improve classification performance to the same degree as it has for kernel methods [15].

References

1. Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped blast and psi-blast: a new generation of protein database search programs. Nucleic acids research 25(17), 3389–3402 (1997)
2. Blasiak, S., Rangwala, H.: A hidden markov model variant for sequence classification. In: International Joint Conference on Artificial Intelligence (2011)
3. Bottou, L.: Online algorithms and stochastic approximations. Online Learning and Neural Networks (1998)
4. Durbin, R.: Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge Univ. Pr. (1998)
5. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis. Cambridge University Press, Cambridge (2002)
6. Eddy, S.R.: Profile hidden markov models. Bioinformatics 14(9), 755 (1998)
7. Kuang, R., Ie, E., Wang, K., Siddiqi, M., Freund, Y., Leslie, C.: Profile-based string kernels for remote homology detection and motif extraction. In: 2004 IEEE Proceedings of Computational Systems Bioinformatics Conference, CSB 2004, pp. 152–160 (2004)
8. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
9. Leslie, C., Eskin, E., Noble, W.S.: The spectrum kernel: a string kernel for svm protein classification. In: Pacific Symposium on Biocomputing, Hawaii, USA, vol. 575, pp. 564–575 (2002)
10. Leslie, C.S., Eskin, E., Cohen, A., Weston, J., Noble, W.S.: Mismatch string kernels for discriminative protein classification. Bioinformatics 20(4), 467 (2004)
11. Liao, L., Noble, W.S.: Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships. Journal of computational biology 10(6), 857–868 (2003)
12. Murzin, A.G., Brenner, S.E., Hubbard, T., Chothia, C.: SCOP: a structural classification of proteins database for the investigation of sequences and structures. Journal of molecular biology 247(4), 536–540 (1995)
13. Qiu, J., Hue, M., Ben-Hur, A., Vert, J.P., Noble, W.S.: A structural alignment kernel for protein structures. Bioinformatics 23(9), 1090–1098 (2007)
14. Rabiner, L., Juang, B.: An introduction to hidden markov models. IEEE ASSP Magazine 3(1), 4–16 (1986)
15. Rangwala, H., Karypis, G.: Profile-based direct kernels for remote homology detection and fold recognition. Bioinformatics 21(23), 4239 (2005)
16. Rumelhart, D.E., Hintont, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature 323(6088), 533–536 (1986)
17. Saigo, H., Vert, J.P., Ueda, N., Akutsu, T.: Protein homology detection using string alignment kernels. Bioinformatics 20(11), 1682–1689 (2004)
18. Schölkopf, B., Mika, S., Burges, C.J.C., Knirsch, P., Müller, K.R., Rätsch, G., Smola, A.J.: Input space versus feature space in kernel-based methods. IEEE Transactions on Neural Networks 10(5), 1000–1017 (1999)
19. Smola, A.J., Schölkopf, B.: Learning with kernels. Citeseer (1998)
20. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.J.: Phoneme recognition using time-delay neural networks. IEEE Transactions on Acoustics, Speech and Signal Processing 37(3), 328–339 (1989)