Annotating protein secondary structure from sequence

Aaron Cravens
Department of Bioengineering
Stanford University
Stanford, CA
`acravens@stanford.edu`

Christopher Probert
Departments of Genetics and Computer Science
Stanford University
Stanford, CA
`cprobert@stanford.edu`

Abstract

Proteins are linear chain biomolecules composed of 20 different amino acids which form secondary structures such as helixes and loops [8]. Here, we use deep recurrent memory networks and deep convectional networks for predicting secondary structure annotations on sequences from the Uniprot database. We compare different network architectures and memory models, and discuss possible improvements. Our results on several tasks are significantly better than current methods that use SVMs and feed-forward neural networks.

1 Introduction

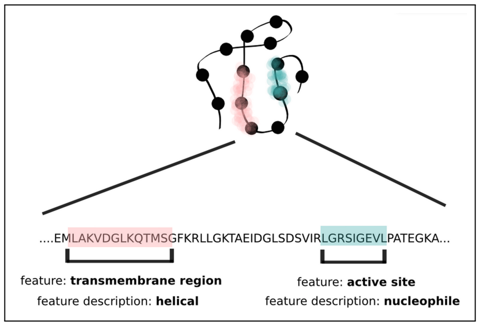
....EMLAKVDGLKQTMSGFKRLLGKTAEIDGLSDSVIRLGRSIGEVLPATEGKA...
feature: **transmembrane region**
feature description: **helical**
feature: **active site**
feature description: **nucleophile**

Figure 1: Example of secondary structure annotations.

Protein feature annotation is classically approached in several ways. Comparative analysis and k-mer based methods have long dominated the field and recently, for structural analysis, hand-crafted features such as FEATURE have proliferated. With current approaches, each classification problem often has a single highly specialized model. Thus, some of the challenges in protein annotation are are akin to the complexities in hand crafted feature representations in image classification and NLP just a decade ago - labor intensive feature generation and a proliferation of highly specialized expert models to achieve good results on specialized tasks.

Inspired by the success of deep learning in these two fields we applied several models to our classification problem to better understand the strengths of these classes of models applied to protein feature classification. While we limited ourselves to two feature types and did not attempt to annotate descriptions, we hope that these models can be extended to multilabel classification task as shown in figure 1.

2 Background and related work

2.1 SVMs

Support vector machines (SVMs) used on k-mer representations have long been a common approach for protein sequence classification. Using kernels (particularly spectrum kernels) improves the performance of such models. Previous results show strong performance on transmembrane-region prediction (.99 AUC on per-sequence tasks), and around .85 AUC on alpha-helix and sheet per-sequence tasks [7, 13]

2.2 Feed-forward neural networks

Some of the first major real-world successes of neural networks came from applications to protein secondary structure prediction. Decision boundaries between different classes are typically non-linear in the feature space, meaning that neural networks with two or more layers have an advantage over linear classifiers because they can generate non-linear decision boundaries. There are several different formulations of this problem, but results for feed-forward neural nets are often comparable to SVM results [10]. Other neural net models are trained for full structure prediction, and have lower accuracy (but the task is much harder).

2.3 HMMs

Hidden Markov state Models (HMMs) are commonly used for many biological sequence tasks, and show strong performance in the case of protein secondary structure annotation. Several of the most commonly models, including the models used to annotate some subset of the Uniprot database, are HMMs [1, 3].

2.4 RNNs and LSTMs

Several published approaches use recurrent neural nets and a subset include memory components, including LSTMs [2, 6, 11]. The most recent paper ([2]), uses an LSTM that is fed directly to the output layer with no additional units, and does not use learned word vector representations. These models were also evaluated on the per-residue task (rather than the per-sequence task in this project), so the results are not directly comparable. Still, these works introduce LSTMs as a powerful model for protein sequence classification, and we see them as a building block for our investigation of LSTMs and other RMNs for these tasks.

2.5 CNNs

Convolutional Neural Networks are in wide spread use in image processing. The power of these models is the convolutional architecture which relies upon feature recognition and extraction at each level in the network. Several recent papers have applied CNN architectures to learn protein secondary structure but have not approached the more general task of feature annotation [12, 14]. Both papers improved upon state-of-the-art accuracies for specific task using sequence information. These results suggest that a CNN architecture can extract meaningful features from sequence information.

3 Approach

Building on current works in the field, we sought to establish our own baseline results using a k-mer based classifier with linear decision rules

3.1 Recurrent networks

Recurrent neural networks operate on sequences [9]. At a given timestep t, the hidden state of the previous timestep $h_{(t)}$ is provided as input into the function to calculate the hidden state $h_{(t)}$, in addition to the sequence input at the current timestep, x_t. The following equations formalize this model:

$$h_{(t)} = Wf(h_{(t-1)}) + W^{(hx)}x_t$$
$$\hat{y}_{(t)} = W^{(S)}f(h_{(t)})$$

Additionally, it is possible to introduce more layers to the RNN, both before or after $h_{(t)}$ of a vanilla RNN [2]. The additional layers can be recurrent, in which case they take input from previous timesteps. Another extension includes taking input from subsequent timesteps, which are termed ”forward-backward RNNs”. Here, we have used a deep backwards RNN, which takes input from 20 previous states. The hidden state formula for this model with $D = 20$ is:

$$h_{(t)} = W^{(hx)}x_t + \sum_{d=1}^{D} W^{(d)}f(h_{(t-d)})$$

We truncated backpropagation through time to a number of steps equal to the model's depth parameter, so errors from one timestep only propagate 20 timesteps back. Additionally, we applied 50% dropout to the input vector x, and 50% dropout to the hidden state $h_{(t)}$. No regularization was applied to the model parameters; we found that this did not affect performance.

Table 1: RNN architecture

Layer	**Activation**	**Size**
Feature embedding	-	21x10
RNN, depth 20	sigmoid	10x128
Output	softmax	128x2

3.2 Recurrent memory networks

Recurrent memory networks are a variant of recurrent nets which generalize the tranmission of information from one state to the next. The two main flavors of RMNs are gated recurrent units (GRUs) and long short term memories (LSTMs). In addition to the same hidden state structure as vanilla RNNs, they use learned weight parameters W to adjust how information is propogated between previous layers and the current layer.

In the context of protein secondary structure annotation, we expect RMNs to perform favorably because information from previous states is unevenly distributed. For example, in an alpha-helix structure, the residue at position $(t-4)$ falls directly above the residue at position t in a 3-dimensional structure [8]. This means that the side-chain of residue $(t-4)$ has much stronger interactions with residue t than other nearby residues. While a deep RNN could learn this structure - for example, by increasing weights in $W^{(4)}$ - we expect the ability to form and control specific memories of preceding residues to be important for learning these types of structured temporal relationships [11].

3.2.1 GRU

Gated recurrent units are a different form of activation function for recurrent nets. Rather than accept the previous hidden states for use in calculating the current hidden state, gated recurrent units store memories of hidden states based on forget and remember gates. In the following notation, σ is used to represent the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$. Figure 2 provides a graphical overview of the GRU model.

$$z_{(t)} = \sigma(W^{(z)}x_{(t)} + U^{(z)}h_{(t1)}) \qquad \text{Update gate}$$
$$r_{(t)} = \sigma(W^{(r)}x_{(t)} + U^{(r)}h_{(t1)}) \qquad \text{Reset gate}$$
$$\tilde{h}_{(t)} = tanh(r_{(t)} \cdot Uh_{(t1)} + Wx_{(t)}) \qquad \text{New memory}$$
$$h_{(t)} = (1z_{(t)}) \cdot \tilde{h}_{(t)} + z_{(t)} \cdot h_{(t1)} \qquad \text{Hidden state}$$

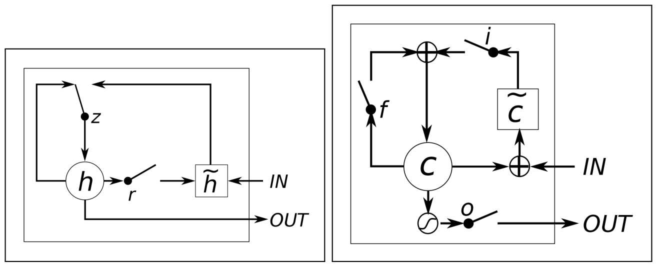

Figure 2: Overview of GRU (left) and LSTM (right) memories. [2]

The goal of the GRU is to produce a new memory $h_{(t)}$. Internally, this is accomplished through the generation of a memory, $\tilde{h}_{(t)}$ The input to the memory non-linearity includes linear transforms of both the previous hidden state $h_{(t-1)}$, as well as the current feature $x_{(t)}$. The mixture of the previous and current inputs is controlled by the update gate $z_{(t)}$ and the reset gate $r_{(t)}$ [2].

Table 2 provides the layers and dimensions used in the experiments of this study. The GRU component of the net is followed by a fully connected ReLU layer, which was found to improve performance (possibly by reducing the softmax input dimension).

Table 2: GRU architecture

Layer	Activation	Size
Feature embedding	-	21x10
GRU	outer: sigmoid, inner: hard sigmoid	10x256
Fully connected	relu	256x128
Output	softmax	128x2

3.2.2 LSTM

LSTMs are another memory activation unit that can be embedded in recurrent nets. Figure 2 provides a grapical overview of the LSTM architecture.

$$\begin{aligned}
i_{(t)} &= \sigma(W^{(i)}x_{(t)} + U^{(i)}h_{(t-1)}) && \text{Input gate}\\
f_{(t)} &= \sigma(W^{(f)}x_{(t)} + U^{(f)}h_{(t-1)}) && \text{Forget gate}\\
o_{(t)} &= \sigma(W^{(o)}x_{(t)} + U^{(o)}h_{(t-1)}) && \text{Output gate}\\
\tilde{c}_{(t)} &= tanh(W^{(c)}x_{(t)} + U^{(c)}h_{(t-1)}) && \text{New memory cell}\\
c_{(t)} &= f_{(t)} \cdot \tilde{c}_{(t-1)} + i_{(t)} \cdot \tilde{c}_{(t)} && \text{Final memory cell}\\
h_{(t)} &= o_{(t)} \cdot tanh(c_{(t)}) && \text{Hidden layer}
\end{aligned}$$

Rather than the transmission between states only being embedded in the hidden state $h_(t)$, LSTMs have a separate representation $c_{(t)}$ which is also transmitted. This significantly increases the number of parameters, but allows information to be propagated internally without it needing to be expressed in the hidden state passed to the next layer at timepoint t. The hidden state at time t is formed as a combination of the memory at time t, $c_{(t)}$, as well as the components normally associated with the hidden state in an RNN, $h_{(t-1)}$ and $x_{(t)}$ [6].

Similarly to GRUs, the memory generation process is controlled by a series of gates. The input gate $i_{(t)}$ is used to learn the importance of the input $x_{(t)}$ at time t. The controls the contribution of this input to the new memory. The forget gate $f_{(t)}$ controls whether the memory $c_{(t-1)}$ from the previous timestep is used in calculating the new memory at timestep t. Finally, the output gate $o_{(t)}$ controls the relative contribution of the new memory to the output hidden state, as opposed to basing the output hidden state on the input element $x_{(t)}$ directly [5].

Table 3: LSTM architecture

Layer	Activation	Size
Feature embedding	-	21x10
LSTM	outer: sigmoid, inner: hard sigmoid	10x256
Fully connected	relu	256x128
Output	softmax	128x2

Table 3 provides an overview of the LSTM architecture used in our models. Features are embedded in a 10 dimensional vector learned on the fly. The 256 output dimension LSTM is followed by a fully connected ReLU layer with output dimension 128 before the output softmax layer. Similarly to GRUs, we found that adding the fully connected layer after the memory unit improved model performance.

3.3 Convolutional neural networks

Convolutional Neural Networks are models which rely upon position invariant filters which scan across input data to generate higher order maps of feature and feature compositions. The power of these models lays in the parameter savings achieved through use of convolution compared to affine layers. The defining function of this class of network computes the input to the next layer weighted by the filter components from the previous layer. In the case of string processing the shown two dimensional convolution is simplified to one dimension.

$$y_{ij}^{\ell} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} x_{(i+a)(j+b)}^{\ell-1}$$

Table 4: Convolutional network architecture

Layer	Activation	Size
Feature embedding	-	21x10
convolution	sigmoid	6x15
convolution-pool	sigmoid	5x30
fully connected	-	300x2
output	softmax	1x2

4 Experiment

4.1 Datasets

Protein sequence information is readily available in online databases. We selected UniProt Knowledgebase as a high quality manually curated source of approximately 400K protein sequences. The length distribution of protein sequences in UniProt falls primarily between 50 and 1200 amino acids. We used these as bounds to what sequences were analyzed. Feature characteristics of UniProt are illustrated in figure 1. Protein sequence features can consist of thousands of different categories and description classes. We selected two high frequency features - transmembrane regions and alpha-helice regions - as feature types to train binary classifiers over. We applied each model to these tasks, and show performance by model.

4.2 Baseline results

To compare the performance of our neural network models, we first built simpler models using random forests on k-mer vector representations. K-mer vectors are a standard representation used for featurizing linear molecules, and are the representation used in several current SVM-based secondary structure annotation methods.

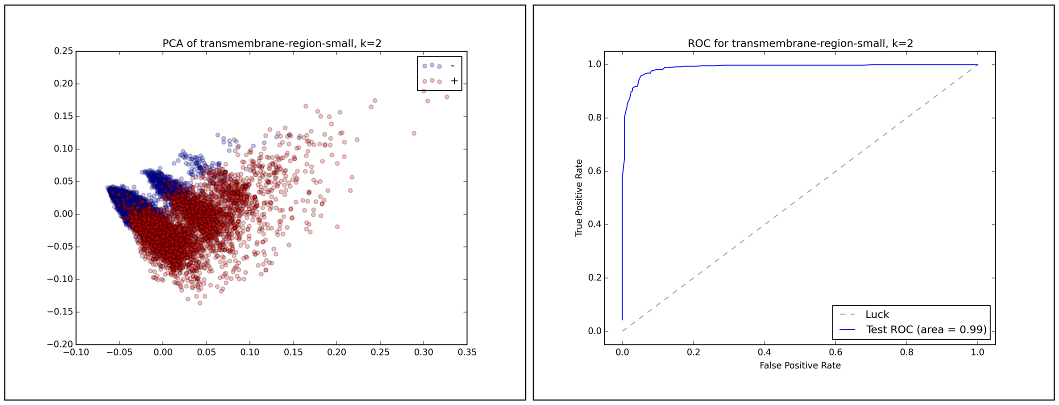

Figure 3: Baseline performance on transmembrane-region task.

We first used the k-mer vector random forest model on the transmembrane-region task. Figure 3 shows a PCA of k-mer vectors for 200K features (left), and the ROC curve for a random forest with 200 predictors of unlimited depth trained on this data (right). The AUC is 0.99, showing that the model performs well on this task, achieving almost perfect accuracy on the balanced dataset.

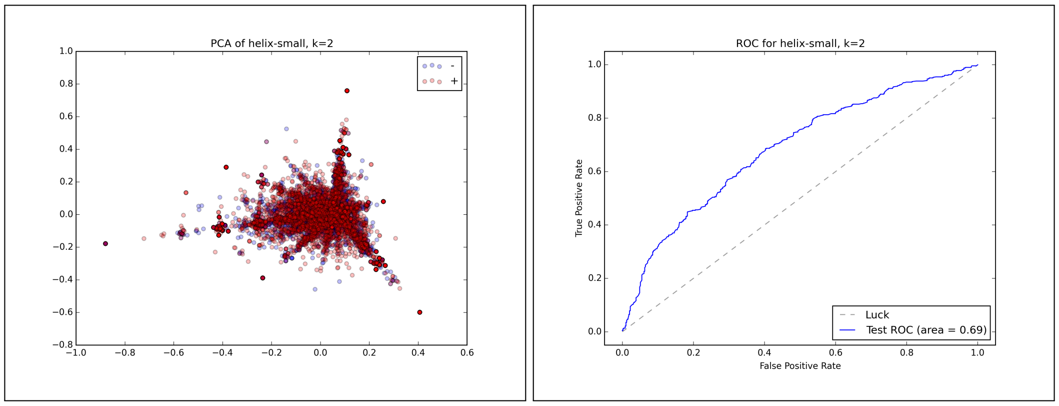

Figure 4: Baseline performance on alpha helix task.

We then used the same model on the alpha helix task. Figure 4 shows a PCA of k-mer vectors for 200,000 alpha helix features (left), and the ROC curve for a random forest with 200 predictors of unlimited depth trained on this data (right). The AUC is 0.69, showing that the model performs significantly worse on this task than the transmembrane-region task.

4.3 Recurrent neural network results

The first recurrent model we tried was a simple recurrent neural network (RNN). Figure 5 shows a ROC plot (left), and a loss function per-epoch plot (right). The AUC is 0.99, showing that the model performs well on this task, achieving almost perfect accuracy on the balanced dataset.

4.4 Gated recurrent unit results

The first recurrent memory model we tried was a gated recurrent unit (GRU). Figure 6 shows a ROC plot (left), and a loss function per-epoch plot (right). The AUC is 0.99, showing that the model performs well on this task, achieving almost perfect accuracy on the balanced dataset.

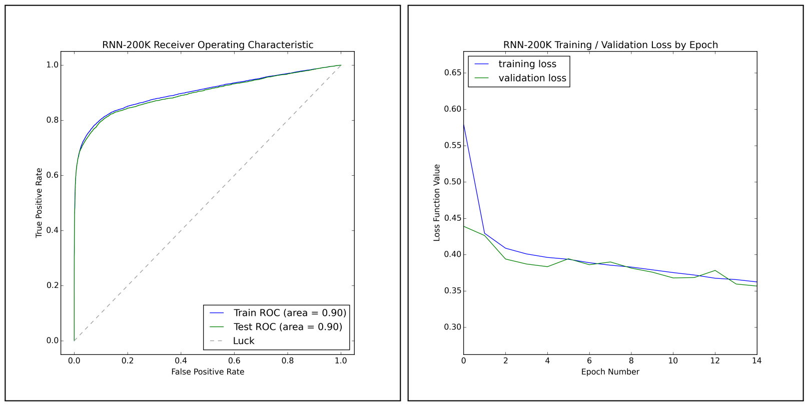

Figure 5: RNN transmembrane-region model performance.

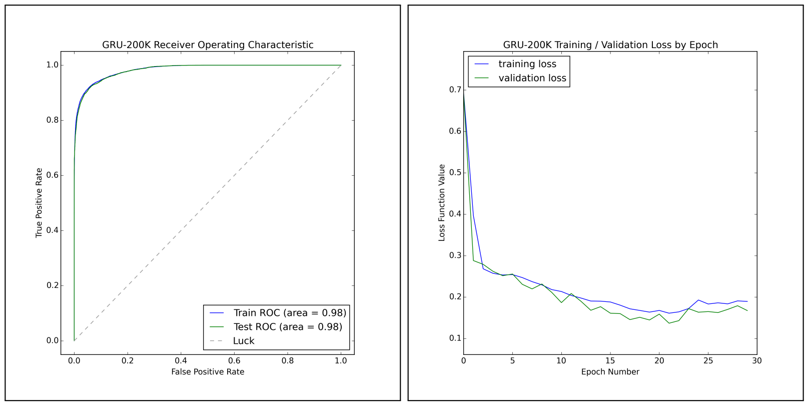

Figure 6: GRU transmembrane-region model performance.

4.5 Long short term memory network results

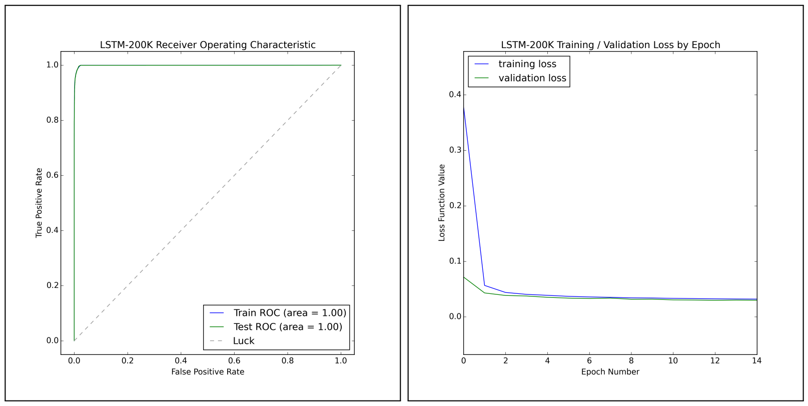

Figure 7: LSTM model performance.

The first recurrent model we tried was a simple recurrent neural network (RNN). Figure 7 shows a ROC plot (left), and a loss function per-epoch plot (right). The AUC is 0.99, showing that the model performs well on this task, achieving almost perfect accuracy on the balanced dataset.

We also trained the LSTM model on the harder helix task, where it also showed strong results (figure 8). The AUC is 0.99, which is significantly better than the baseline AUC (0.69).

For space we have omitted results on the helix task for the GRU and RNN models, which showed AUCs around 0.7-0.8. This is better than baseline, but not close in performance to the nearly perfect LSTM results.

An interesting artifact is present in the loss function plot for the LSTM on helix tasks (figure 8). The plot shows the typical fast learning within the first 2-3 epochs, but then the loss increases periodically around 4 epochs, and then again around 9 epochs. It returns to its global minima, but the sudden spikes were not observed in other models. In some training regimes this is a basis for early stopping, but we decided to train for a fixed number of epochs rather than introduce an additional hyperparameter. We note that the final model dev accuracy (at epoch 20) is better than then dev accuracy at epoch 3 (the first minima), suggesting that the model does continue to improve after these first spikes in loss.

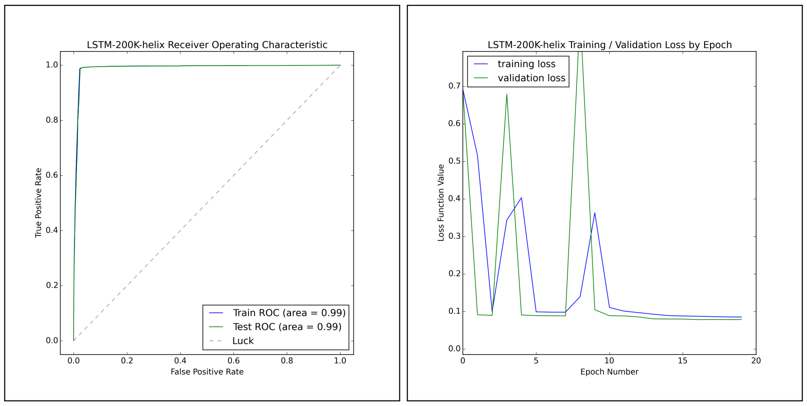

Figure 8: LSTM helix task model performance.

4.6 Convolutional network results

The first convolutional neural network was based off of a recent paper in NLP using dynamic max-K pooling to process sentences of varying length. This model was implemented in Theano but did not train well due to un-supported gradient passing for the version of max-K implemented.

We then implemented a series of convolutional architectures of varying filter size based around the architecture shown in figure 4. Filters in layer one were varied as 5, 10, 15 and at layer two as 10, 20, 30. The best performing model had the most filters, was trained using vanilla SGD and had a weak regularization of 1E-5. Results for TM-domain and helix prediction are shown in figures 10 and 11 respectively. The TM-domain task outperformed our standard substantially as we expected when evaluated on a random background. The absence of model over-fitting in this task indicates that we could implement a larger model and expect additional improvement.

We implemented the architecture in figure 4 without further hyperparameter optimization training our helix data-set of 100k examples on a random protein background. The performance was substantially better than the benchmark result, and similarly to results with TM-domain, we observed no overfitting indicating that a larger model should improve further.

5 Conclusion

5.1 Performance on transmembrane and helix tasks

The LSTM model described in table 3 achieved near-perfect precision and recall $> 99\%$ on transmembrane-region and helix tasks against both shuffled same-class and database-wide backgrounds. The RNN, GRU, and convolutional nets performed well on the transmembrane task, but not as well on the more difficult helix task.

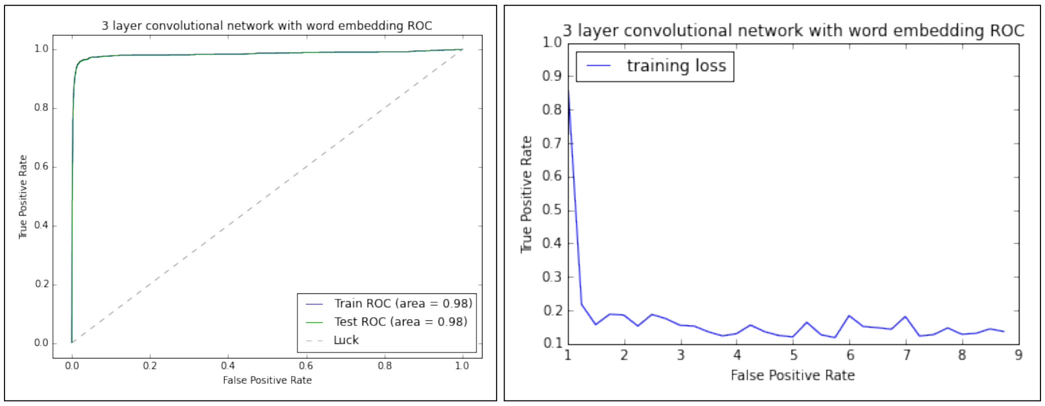

Figure 9: CNN transmembrane-region performance

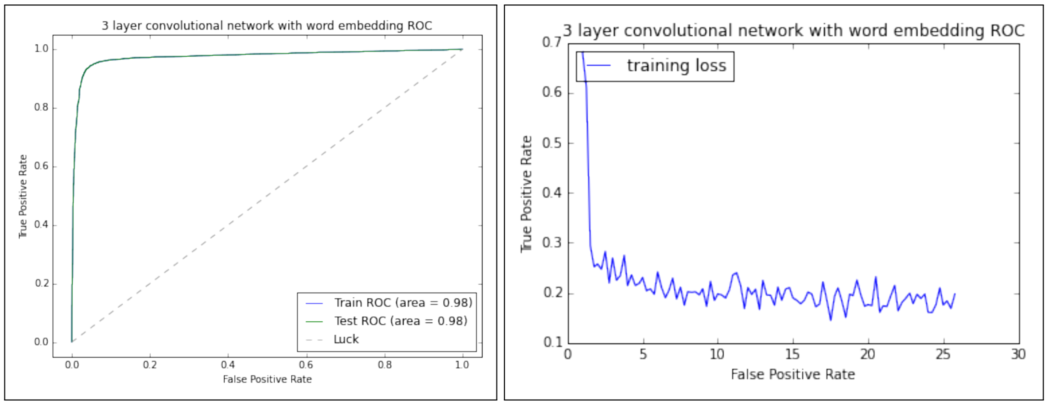

Figure 10: CNN helix task performance

5.2 Training characteristics and initialization

To eliminate the need to use a learning rate hyperparameter, we used Adagrad [4] for all recurrent. In general, our RNNs and RMNs began learning from the first epoch, and arrived within +/- 1% of the final loss value within 5 epochs.

We experimented with a deeper variant of the LSTM (by adding a second LSTM layer and another fully connected layer). In this model, we observed a peculiar learning behavior, where the model only began decreasing loss after 10-12 epochs. We recently received feedback that this may be due to a bad choice of initialization. In general we used Glorot uniform initialization with default parameters.

Another artifact we noticed during LSTM training was the spikes seen in the loss curve of figure 8. We don't have a strong explanation for these spikes, but future improvements could try different initialization and/or early stopping. We could also try different learning algorthims, such as AdaDelta or regular SGD with a higher decay rate.

5.3 Regularization

5.3.1 Weight-based

We did not apply weight-based regularization to any of the recurrent models presented here. We found that adding L-1 or L-2 regularization did not improve model performance, and several LSTM and RNN reference papers [5, 6] contained models without weight-based regularization.

5.3.2 Dropout

We extensively applied dropout regularization throughout our models, and found that it improved dev performance during training and reduced overfitting the training data compared to test data. We use relatively large hidden dimensions and word vector representations, which likely made dropout important to prevent overfitting (e.g. memorizing training data). We generally applied 0.2 - 0.5 dropout to each layer, counting the $h_{(t)}$ output of memory units as a single layer (we did not apply dropout to internal vectors of memory units, such as $c_{(t)}$ of an LSTM).

5.4 Future work

5.4.1 Analysis of current results

Our analysis of our current models was limited by time constraints. We wish to better understand the weights our models learn: are all features important in each layer? How do different residues $x_{(t)}$ in a sequence affect, for example, the activation of the forget gate $f_{(t)}$ in an LSTM? Which residues in which positions have the largest effect on the posterior distribution generated from the softmax output layer? Intuitively, do these correspond to residue/position combinations that are likely to have significant impacts on the given secondary structure?

5.4.2 Analysis of dataset

While we performed careful analysis of the label distribution in the training dataset (Uniprot), we did not analyze the sequence diversity of the dataset. For example, how many unique k-mers does a particular label contain? Does sequence redundancy affect results, positively or negatively, in either the 1-vs-all or 1-vs-shuffled tasks?

5.4.3 Extension to multitask and per-base models

In this paper, we present results only from 1-vs-all and 1-vs-shuffled tasks on single labels. We wish to extend these methods to a multitask setting with per-base labels, so we can predict multiple labels for each base at once. We have preliminary results for our networks on a 5-way, per-base multitask setting where performance is directly comparable to in the 1-vs-all task, but did not include them here for time and space constraints.

5.4.4 Different model architectures

Relative to the possible model and hyperparameter spaces, our sampling of different model configurations was very sparse. For example, on the full dataset we tried only two different networks with LSTMs, one of which had more than twice the number of layers than the other. To improve scaling, we would likely need access to improved computing resources, and/or move our model training to GPU (currently our models use Theano, but are set to CPU execution).

Acknowledgements

We thank Richard Socher, Oana Ursu, and Anshul Kundaje for discussion.

C. Probert submits this work as a joint final project with CS373 (Machine Learning for Genomics).

References

[1] Asai, K., Hayamizu, S., and Handa, K. Prediction of protein secondary structure by the hidden markov model. *Computer applications in the biosciences: CABIOS 9*, 2 (1993), 141–146.

[2] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[3] Cole, C., Barber, J. D., and Barton, G. J. The jpred 3 secondary structure prediction server. *Nucleic acids research 36*, suppl 2 (2008), W197–W201.

[4] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research 12* (2011), 2121–2159.

[5] Gers, F. A., Schmidhuber, J., and Cummins, F. Learning to forget: Continual prediction with lstm. *Neural computation 12*, 10 (2000), 2451–2471.

[6] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069* (2015).

[7] Guo, J., Chen, H., Sun, Z., and Lin, Y. A novel method for protein secondary structure prediction using dual-layer svm and profiles. *PROTEINS: Structure, Function, and Bioinformatics 54*, 4 (2004), 738–743.

[8] Kabsch, W., and Sander, C. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers 22*, 12 (1983), 2577–2637.

[9] Pineda, F. J. Generalization of back-propagation to recurrent neural networks. *Physical review letters 59*, 19 (1987), 2229.

[10] Qian, N., and Sejnowski, T. J. Predicting the secondary structure of globular proteins using neural network models. *Journal of molecular biology 202*, 4 (1988), 865–884.

[11] Sønderby, S. K., and Winther, O. Protein secondary structure prediction with long short term memory networks. *arXiv preprint arXiv:1412.7828* (2014).

[12] Spencer, M., Eickholt, J., and Cheng, J. A deep learning network approach to ab initio protein secondary structure prediction.

[13] Ward, J. J., McGuffin, L. J., Buxton, B. F., and Jones, D. T. Secondary structure prediction with support vector machines. *Bioinformatics 19*, 13 (2003), 1650–1655.

[14] Zhou, J., and Troyanskaya, O. G. Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. *arXiv preprint arXiv:1403.1347* (2014).