Protein Family Classification with Neural Networks

Timothy K. Lee
Program in Biomedical Informatics
Stanford University
tklee@stanford.edu

Tuan Nguyen
Department of Statistics
Stanford University
tuanminh@stanford.edu

Abstract

Understanding protein function from amino acid sequence is a fundamental problem in biology. In this project, we explore how well we can represent biological function through examination of raw sequence alone. Using a large corpus of protein sequences and their annotated protein families, we learn dense vector representations for amino acid sequences using the co-occurrence statistics of short fragments. Then, using this representation, we experiment with several neural network architectures to train classifiers for protein family identification. We show good performance for a multi-class prediction problem with 589 protein family classes.

1 Introduction

Next-generation sequencing technologies generate large amounts of biological sequence information in the form of DNA/RNA sequences. From DNA sequences we also know the amino acid sequences of proteins, which are the fundamental molecules that perform most biological functions. The functionality of a protein is thus encoded in the amino acid sequence and understanding the sequence-function relationship is a major challenge in bioinformatics. Investigating protein functional often involves structural studies (crystallography) or biochemical studies, which require time consuming efforts. Protein families are defined to group together proteins that share similar function, and the aim of our project is to predict protein family from raw sequence. We focus on training informative vector representations for protein sequences and investigate various neural network models for the task of predicting a protein's family.

2 Background/Related Work

Traditionally, analyzing protein sequences involves searching for common motifs or through phylogenetic comparison against known protein sequences. As such, classifying a new protein sequence relies heavily on feature engineering using prior knowledge and experts for annotation. Previous work by Asgari et al. [4] demonstrated that word2vec vectors [9] representing trigrams of amino acids could be trained on large amounts of protein sequence data. The resulting vector representation maintained known biological relationships and were successfully used as features for protein family classification.

Neural Networks (NNs) models have achieved state-of-the-art performance on language modeling tasks in recent years and are now seeing adoption for biological problems. For example, Alipanahi et al. [3] have used convolutional neural networks to accurately predict the binding affinity of proteins to different DNA or RNA sequences or to predict splice specificity for different RNA sequences. Zhou et al. [13] have also used convolutional neural networks to predict epigenetic and chromosomal features from DNA sequences and recurrent architectures have been applied to this problem [11]. We will apply these language models to the task of protein family classification.

Total # of sequences	550,960
Sequence length	10-35000
Total # of families	10345
Total # families with $>$ 200 sequences	589
# of sequences used for classification	317460

Table 1: Uniprot dataset of annotated protein sequences.

```
MAFSAEDVLKEYDRRRRMEALLLSLYYP...   Pox_VLTF3
MSIIGATRLQNDKSDTYSAGPCYAGGCS...  Pox_G9-A16
MQNPLPEVMSPEHDKRTTTPMSKEANKF...        US22
```

Figure 1: Examples of protein sequences and their families

3 Approach

3.1 Dataset

Our dataset consists of annotated protein sequences (Fig. 1) from the Universal Protein Resource (UniProt) database [1] , with 550,960 protein sequences across 10,345 families in total. Each sequence is a string of characters of length varying from 10 to 35,000. Of all the protein families in the dataset, we select only those with more than 200 examples. This results in 589 families and 317,460 sequences used in the training and testing (Table 1).

To train models for protein family classification, we limited ourselves to sequences of less than 1000 overlapping trigrams (Fig. 2). The data was then split into training/validation/test folds at a 70/15/15 ratio preserving class stratification.

3.2 Global Vector (GloVe) for Amino Acid Sequence Representation

Previous work using the word2vec skipgram model was able to generate trigram representations that reproduced known physical relationships and were useful for protein classification. We evaluate whether Global Vectors for Word Represetation (GloVe) can generate improved representations using the full co-occurrence statistics available in our corpus.

To create a distributed representation of our protein sequences, we represent each sequence as a series of trigrams (a block of 3 amino acids) and create a distributed representation of each trigram using GloVe.

3.3 Baseline Classifier (Support Vector Machine)

For our baseline, we used a classification model described in Asgari et al. where a protein sequence is represented by the sum of all its trigram representations. We used a radial basis kernel SVC with penalty parameter $C = 10$, $\gamma = 10^{-3}$, using a one-vs-one multi-classification scheme. The hyperparameters were selected using grid search. The model took approximately 7 hours to train using the SVC package in scikit-learn [10].

```
MVE-RLG-IAV-EDS...;    VER-LGI-AVE-DSP...;    ERL-GIA-VED-SPK...
```

Figure 2: Examples of shifted non-overlapping trigrams from the first sequence in Fig. 1

3.4 Neural Network Models

We used overlapping trigrams in sequence as the inputs to the neural networks and initialized our inputs with our GloVe embeddings and allowed them to be trained.

3.4.1 Gated Recurrent Neural Networks (GRU)

Gated Recurrent Neural Networks [6] extend recurrent neural networks (RNNs) by using gated recurrent units (GRUs, [5]). The goal of these units is to overcome the limitations of RNNs in capturing effects over different time scales and overcome issues with vanishing gradient over long sequences. GRUs consist of two additional gates, an update gate and a reset gate. Using the new input, the new memory content is calculated with the reset gate determining how much of the previous hidden state is used before the nonlinear activation. Then, the update gate determines how much of the new memory content is used with the previous hidden state to determine the new hidden state. Together, these two gates allow long or short term dependencies to be expressed.

3.4.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory [7] models are a variant of RNN. In RNN, learning takes place over a sequence of steps, where inputs to a step comprise also of the output of the previous step's hidden layer. This enables the network to capture information from the past that can inform future predictions. LSTM adds "gates" that control the degree of influence of information from the current step and from the previous steps, as well as filtering out the parts of memory that are less important in making a prediction. This mechanism allows more flexible control over memory [7].

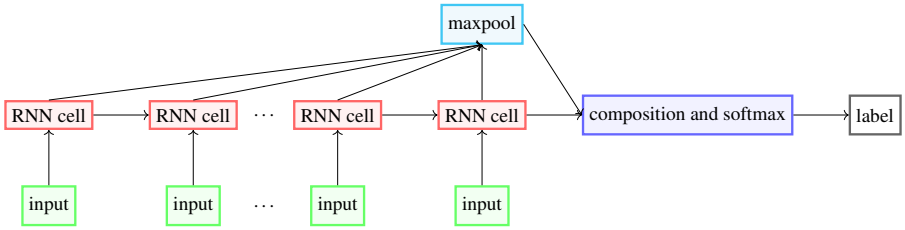


Figure 3: Recurrent neural network schematic.

3.4.3 Bidirectional LSTM (biLSTM)

BiLSTM is an extension of LSTM, in which an additional recurrence starts from the last timestep of the forward recurrence and proceeds backward to the first timestep of the forward recurrence. The information in the "future" steps thus can be captured and aids predictions making at earlier timesteps [12].

3.4.4 Convolutional Neural Network (CNN)

Convolutional neural networks have gained popularity for their success in computer vision problems [8], but can also be applied to sequence data. Convolution filters of a fixed size are scanned across a sequence to produce activation across that sequence. Since sequence data can vary in length, we maxpool over the entire sequence to find the maximal activation of a filter across a sequence. The maximal activation of many convolutional filters used as a fully connected layer to the output classes. For this study, we used three filter sizes (3, 9, 27), with 128 filters each.

3.4.5 Evaluation method

Protein family labeling is a multi-class classification problem, so we used F1 score,

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

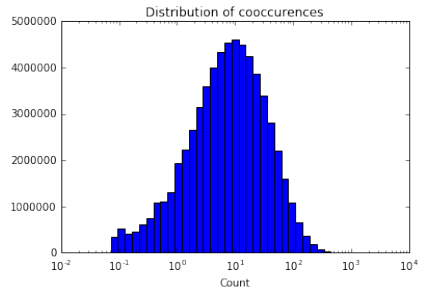
to evaluate the models' performance.

4 Experiments

4.1 Training

4.1.1 GloVe Embeddings

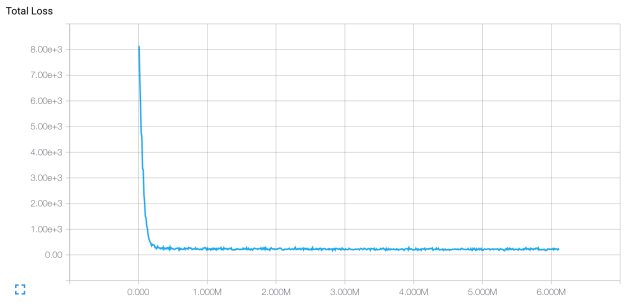
We computed the trigram co-occurrence matrix for all sequences using a symmetric window of 12 on each side. We considered non-overlapping trigrams in all three shifts resulting from shifting the reference location by one. Co-occurrences in a window were weighted by the inverse distance to the window center. This resulted in 10,311 trigram represented and 62 million non-zero entries in the cooccurrence matrix. Based off of the histogram of co-coccurrence counts (Fig. 4a), we selected a co-coccurence cap of 20 for the GloVe model. To train the GloVe embedding, we implemented the GloVe model in Tensorflow [2] and trained the embeddings using the hyperparameters specified in Table 4b using a GTX 980 Ti. The GloVe model quickly converged during training (Fig. 4c), but some progress was made even in later iterations. To see whether the embeddings separated well, we visualized the embedded representations with t-SNE (Fig. 4d) using the first 25 principal components.



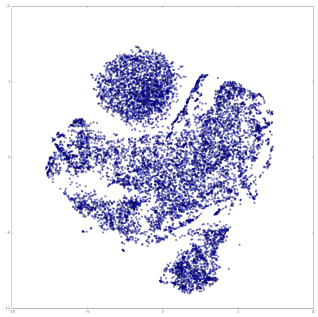
(a) Trigram co-occurrence counts in the UniProt dataset.

Max co-occurrence (x_{max})	20
Learning rate	0.0001
Batch size	512
Embedding size	100
Number of Epochs	50

(b) GloVe Training Parameters



(c) GloVe Loss



(d) t-SNE visualization of GloVe embeddings

Figure 4: GloVe training

4.1.2 Neural Network Model Training

In our initial LSTM recurrent network implementation, we used only the final hidden state's output to calculate logit scores for the SoftMax. When we trained these models, the loss would not appreciably decrease below 6, which is close to that of random guessing ($-\ln(\frac{1}{589})$), and we realized we could not overfit even small models. Using a more complex biLSTM structure, we had more initial success and we were able to achieve test F1 scores of about 0.5 on the full dataset. We drastically improved our models by maxpooling over the entire sequence of hidden states and our final models concatenated the maxpool and final hidden state. With the maxpooled output, we could overfit smaller datasets easily and on the full dataset our models were able to reach validation F1 scores above 0.9 within the first 7 epochs.

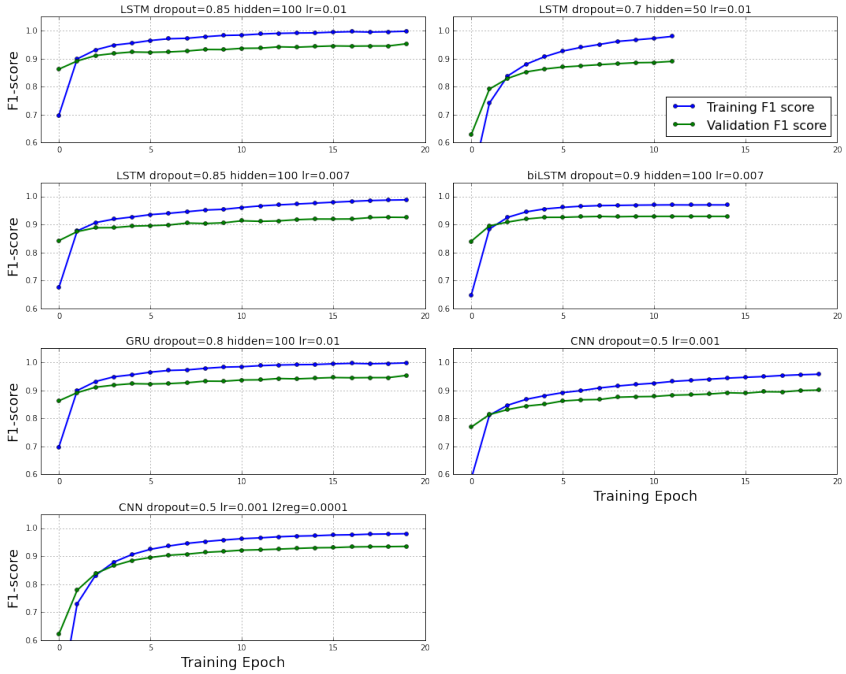


Figure 5: Performance during training for neural network models (Green: Validation set score, Blue: Training set score).

Our LSTM implementation achieved near perfect accuracy on the training set, so we attempted to decrease the hidden state size as a way to generalize the model further. During training we saw that the gap between training and validation was higher, so we attempted to decrease the dropout probability instead, which showed only a slight improvement.

For the biLSTM model, we first tried the averaged and stacked outputs of the final hidden layers of the forward and backward nets as input to the softmax. Similar to LSTM models, we observed significantly better results using maxpooling. Specifically, we applied maxpooling over all hidden layers' outputs of the forward net and backward net, respectively, and stacked the maxpooled outputs before feeding into the softmax layer. The biLSTM model was able to reach a validation F1 score above 0.9 within the first 3 epochs. Since we achieved near perfect accuracy on the training set with single direction LSTM models, it suggested we did not need the additional complexity of the biLSTM, which took nearly three times longer to train per epoch.

We attempted to train a gated recurrent neural network using similar parameters and found that it achieved the best results of our models. It had slightly better performance than the LSTM and biLSTM models, but this might be due to the particular instantiation of hyperparameters. However, the GRU model was training 10% faster than the LSTM models.

For the convolution networks, we saw a gain from using l2 regularization on all filters and weights, although it is unclear why since our training score is higher with regularization. From the loss histories, it seems that without regularization the training loss didn't stabilize even after 20 epochs.

We found dropout to be quite effective. For all the models, using only one hidden layer was enough to achieve sufficient model complexity. We experimented with a number of learning rates. We found that learning rate of 0.01 worked best for most cases. No significant improvement from random initialization of the weights was observed.

4.2 Results

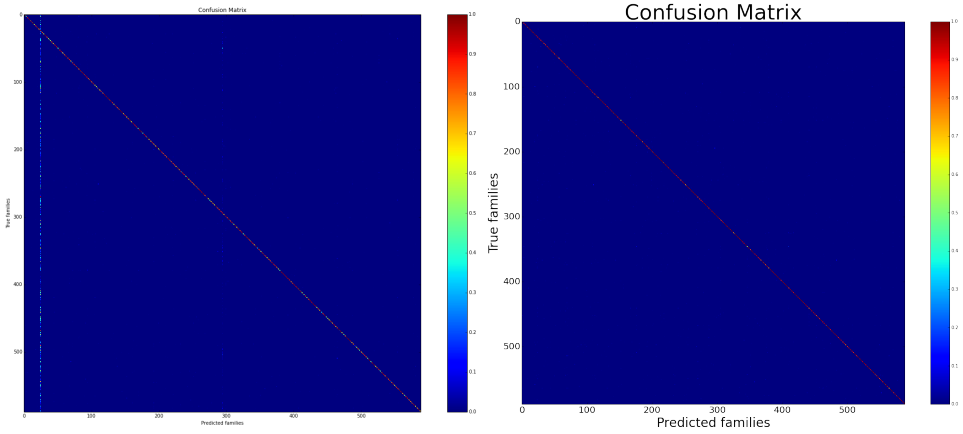
Our results suggest that protein families can be accurately predicted from amino acid sequences. In our subproblem consisting of 589 protein classes, it seems that more complex models (such

as the bidirectional LSTM) are not required for high accuracy. All of the neural network models outperform the SVM baseline, which does not take trigram order into account. Our best performing model was the GRU, but it is also likely we could achieve similar results with the other neural network architectures with hyperparameter tuning. Since we could achieve near perfect training performance most neural network models, it seems we should not see much improvement with more complex model such as biLSTM. With few exceptions, our GRU performed better than the

Model	# hidden units	lr	l2reg	dropout	Val. F1	Test F1
SVM	-	-	-	-	-	0.87876
LSTM	100	0.01	0	0.85	0.926192	0.92515
LSTM	50	0.007	0	0.85	0.890175	0.888509
LSTM	100	0.01	0	0.70	0.925665	0.922225
biLSTM	100	0.007	0	0.9	0.928740	0.927899
GRU	**100**	**0.01**	**0**	**0.8**	**0.953141**	**0.948452**
CNN	384	0.001	0	0.5	0.9006	0.897853
CNN	384	0.001	0.0001	0.5	0.934	0.934

Table 2: Test F1 scores for different models and hyperparameters

SVM baseline and did not significantly underperform for any given class. Examining the confusion matrcies, we see that the SVM performs very badly on certain classes, which is not evident with the GRU (Fig. 6). To directly compare the per class performance, we plotted the F1 scores for each



(a) SVM confusion matrix

(b) GRU confusion matrix

Figure 6: Classification Performance SVM vs GRU

family for both classifiers (Fig. 7). The GRU classifier performs better overall and there do not seem to be classes where the GRU performs poorly where the SVM does well.

To visualize our results, we used tSNE to examine the embeddings generated as outputs from the recurrent networks (before computing logits). From the t-SNE embeddings (Fig. 8), we see that class examples are often clustered tightly together. Ideally, similar protein classes should be close together in the embedding space (such that biological meaning is somehow represented), but on visual inspection no clear pattern of nearby classes exist.

5 Conclusions and Future Research

We have experimented with a number of neural network architectures to train a classifier for the task of protein family identification. We demonstrate that neural network classifiers give superior performance to our tuned SVM baseline on the same Uniprot dataset; in particular GRU outperformed our SVM baseline by almost 7%. In comparison to the SVM in Asgari et al. [4], which was

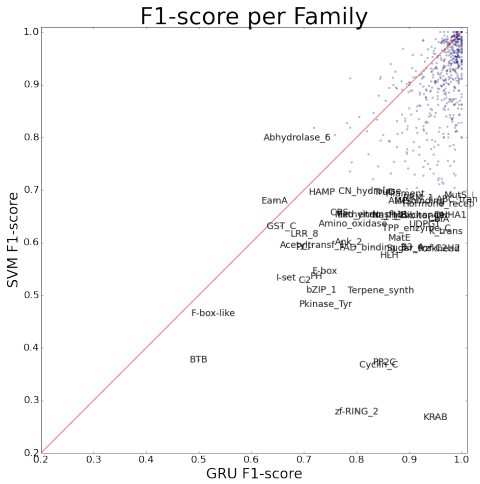
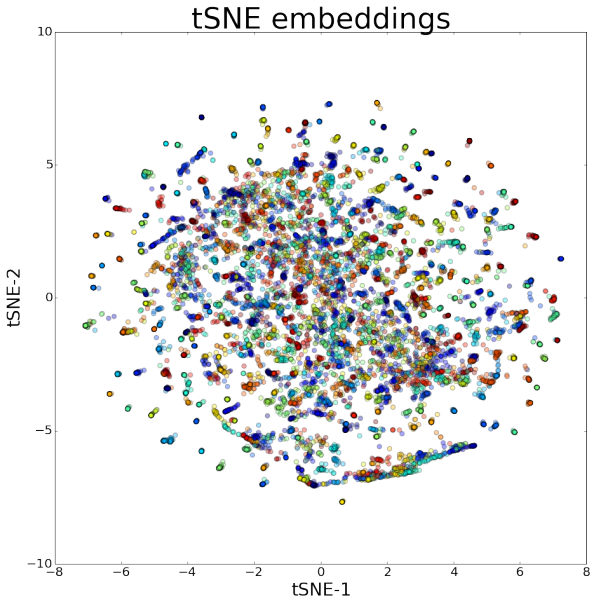


Figure 7: GRU vs SVM per class F1-score

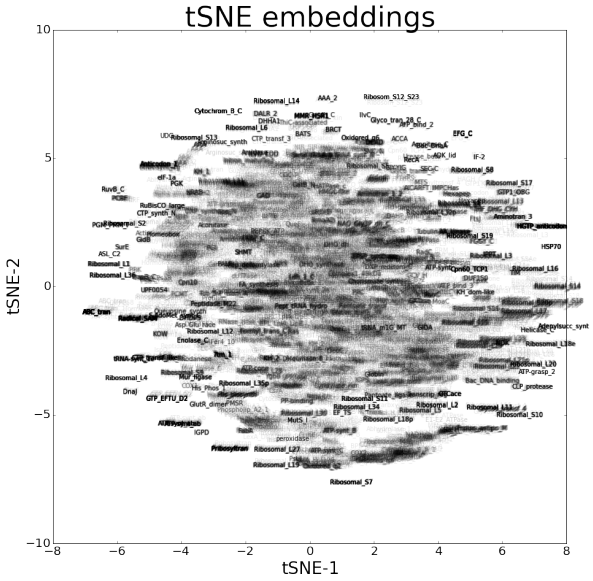
trained many single-class classifiers, the multi-class protein is more difficult and our neural network architectures perform well.

There are a number of directions in which we could expand this project. One would be to experiment with different lengths of the n-grams used to train the GloVe embeddings for the protein sequence.

Another direction would be to investigate if there exists any relationship between the biological properties of a protein family captured by our models. It is not clear from our analysis if the representation the trigrams or our sequences maintained biological meaning similar to how semantic and syntatic meaning is preserved in natural language models. Further investigation is required to examine both the GloVe embeddings and sequence representation to determine if higher order function is being used by the classifiers. It is possible that we are achieving good performance without biological representation and that we might require additional features if those representations were the focus.



(a) GRU tSNE embeddings (color coded)



(b) GRU tSNE embeddings (class labels)

Figure 8: t-SNE embeddings of test sequences

References

[1] Universal protein resource (uniprot) database. `http://www.uniprot.org/downloads`. Accessed: 2015-05-05.

[2] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[3] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 2015.

[4] Ehsaneddin Asgari and Mohammad RK Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 10(11):e0141287, 2015.

[5] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[6] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *arXiv preprint arXiv:1502.02367*, 2015.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Daniel Quang and Xiaohui Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *bioRxiv*, page 032821, 2015.

[12] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on 45.11 (1997): 2673-2681.2.*, 1997.

[13] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.