DS-GA 1006 Capstone, Fall 2017

Sequential Deep Learning Methods for Protein Function Prediction

Akash Kadel[1], Brenton Arnaboldi[1], and Daniel Amaranto[1]

[1]MS in Data Science, New York University

Abstract

We attempted the task of protein function prediction on a simplified input set of human and yeast protein sequences. We compared the following deep learning models: FastText for text classification, a Recursive Neural Net (RNN) with Gated Recurrent Unit (GRU), a RNN with Long Short Term Memory (LSTM) network, and a Convolutional Neural Net (CNN). The relative results for each model differed depending on which evaluation metric was considered. As with any algorithm for protein prediction, sparsity and imbalanced classes were an impediment to model performance. We discovered a correlation between the extent of class imbalance in the input data and the performance of our models as assessed by F1-score. Despite challenges in improving performance, these simplified models might be integrated into larger models with more diverse inputs in order to augment their overall performance.

1 Research Question

Protein sequences are being identified at a very fast pace, while deeper understanding of what the identified sequences actually do is acquired much more slowly. A large number of characteristics of a given protein are of interest to scientists, such as structure, protein-protein interaction, and determining various functions. Since confirming those characteristics requires empirical validation that takes much longer than merely identifying a sequence, we can expect that the pace of sequence identification will continue to be faster than that of any of the other protein-related discoveries. Therefore one interesting question in microbiology today would be whether machine learning can be used to predict the functions of sequences that haven't been encountered before. This task has been met with varying degrees of success in a variety of studies. The Critical Assessment of Functional Annotations (CAFA) [12, 8], is a competition among models for this task that

has been conducted twice so far. Its competing models have involved computational techniques that consider a wide variety of features for functional prediction, including functional domains, protein interactions, mass spectrometry, gene interactions, clinical data, natural language processing, and many more [8], but there is room for improvement. Since many models that are relatively successful at this task take a very large number of inputs and combine them in complex ways, a more narrow question in the realm of protein function prediction is whether or not there are simpler, more efficient inputs and model structures that could improve performance without amplifying complexity. We propose strictly using sequence information in text format for our input set.

2 Data

Our data consisted of 12475 human protein sequences divided into train, validation, and test sets of sizes 9751, 3871, and 1647, respectively. We also have yeast protein sequences divided into train, validation, and test sets of sizes 3447, 963, and 206, respectively. The human protein sequences collectively predict 147 different GO molecular function terms (GO terms are described in Section 2.1) and the yeast protein sequences collectively predict 26 GO molecular function terms. 20 functions were present in the sequences of both the human and yeast sets.

In human data the average length of protein sequences was 563 amino acids, but there were two outliers of length 35000 and 14000. Those sequences were truncated to 5000 to alleviate their outsize effect on memory requirements. In the yeast training sequences the average length was 532 amino acids and the maximum length was 4910. Given that these were variable length sequences, we set the first dimension of each trained batch to the maximum length of the sequence in that batch, and padded all shorter sequences with 0s.

In total there were 22 different amino acids in all of the human sequences and 20 amino acids in the yeast sequences. In each training example, the letters that were contained in each string were assigned to one of 22 numbers. We then created a randomly initialized embedding vector for each individual amino acid.

2.1 Outputs: GO Terms

With respect to labeling, a taxonomy of protein functions has been developed by the Gene Ontology Consortium.

The structure of GO functions can be described in terms of a graph (Figure 1), where each GO term is a node, and the relationships between

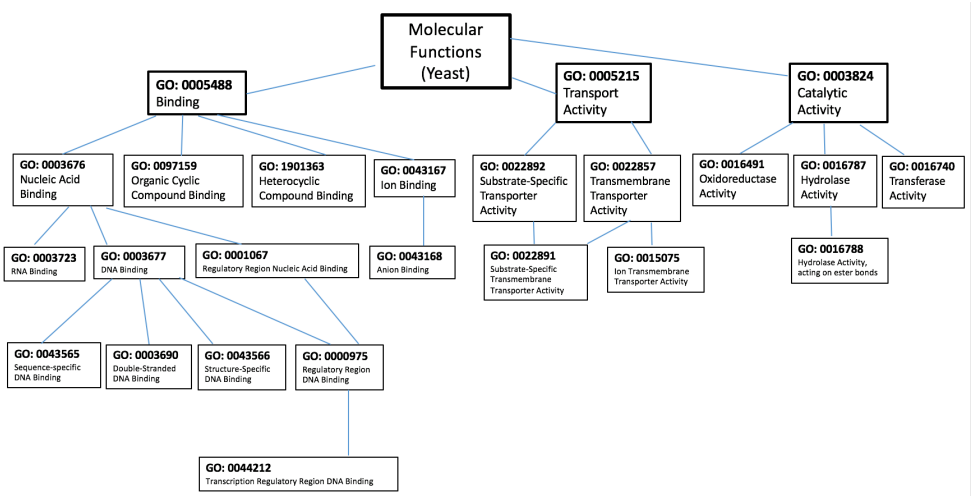


Figure 1: Ontology tree for 22 of 26 yeast functions in dataset. Most GO labels fall under three main buckets: 1) Binding, 2) Transport, or 3) Catalytic.

the terms are edges between the nodes. GO is loosely hierarchical, with 'child' terms being more specialized than their 'parent' terms, but unlike a strict hierarchy, a term may have more than one parent term (note that the parent/child model does not hold true for all types of relation). For example, the biological process term hexose biosynthetic process has two parents, hexose metabolic process and monosaccharide biosynthetic process. This is because biosynthetic process is a subtype of metabolic process and a hexose is a subtype of monosaccharide.

In Figure 1 above, the hierarchy for a subset of the Yeast functional terms are shown, in which relations are represented by edges between the squares; the terms get more specialized going down the graph, with the general terms — molecular function in this example — at the top of the graph. Terms may have more than one parent, and they may be connected to parent terms via different relations. Ideally, a model will predict every appropriate label within a branch. That is, if the model predicts a particular label, it should also predict every function that lies higher up the hierarchy as well. Our analysis will show that the models we use are mostly sensitive to the degree of class imbalance in the label set, and don't necessary predict labels along the hierarchies successfully.

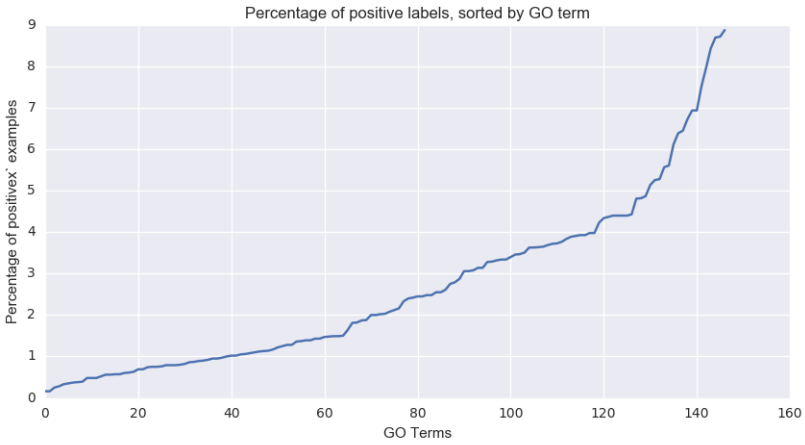


Figure 2: Ranked Percentage of Positive Predictions in 147 Human GO term Labels

2.2 Class Imbalance

As shown in Figure 2, the data was highly imbalanced, with the vast majority of labels in the training set being 0. For human data, the least imbalanced label still had only positive labels in 9% of the sequences. Class imbalance is a typical problem in protein prediction tasks, and we found it to have an impact on performance, as will be discussed.

2.3 Train, Validation, Test Set Split

The selection of train, validation, and testing data is temporally determined and aligns with the timeline by which discoveries of functions are made empirically. At time t_0, a set of protein sequences is identified and their respective annotations (if present) are recorded. At a later time t_1 the annotations of those same protein sequences are captured and used to set the train, validation, and test sets as follows: proteins that have had no changes to their annotations become the train set. Proteins that originally had some annotations and then gained more annotations between t_0 and t_1 become the validation set. Proteins that had no annotations at all at t_0 and gained some annotations in t_1 become the test set. Finally, the entire set of annotations is summed and the labels are limited to functions that have between 10 and 1000 total instances. This step eliminates extremely common and extremely uncommon functions. Because of this final filtration, there are some proteins in the set that have no functional predictions at all.

3 Methodology

3.1 Model experimentation

There are a variety of methods that could be promising for the task of using sequence data alone to predicted GO functions. Since our input data is strings of text wherein each letter represents a specific amino acid, we have focused on different deep learning methods that have been successful at text classification tasks[9, 11, 13, 2]. As applied to text classification, the methods below convert words or characters into vector representations, and the overall sequence of representations in an input sentence or paragraph correspond to some label for the given text entry (for example, product reviews and movie reviews can have "positive", "neutral", or "negative" labels, or some algorithms have been successful at identifying specific languages or even particular authors from an input sequence. We base our choices of models below upon this same premise. It is possible that the particular sequence of amino acids, each of which can be represented by a vector, is sufficient to train a model to predict function.

This approach admittedly simplifies the highly dynamic nature of proteins. Protein sequence does in fact determine its shape, and its shape essentially defines its functions. However, it doesn't follow that the sequence alone will provide enough information for these algorithms to connect our inputs to the functional outputs. Moreover, certain small variations in amino acids, or a re-ordering of sections in an amino acid chain can potentially have a significant effect on function.

3.1.1 FastText

FastText is a simple and very fast algorithm that takes variable length sequences for text classification tasks. In some tasks its performance is similar to embeddings trained with deep learning methods; however, as with many text based models, it requires a considerable amount of training data in order to be effective. As stated above, this model begins with a look-up table matrix of representations for each textual component (in this case one of the 22 amino acid letters). For each input, the representations are averaged and then fed into a linear classifier [9]. In our implementation, we pass the averages through a rectified linear unit (ReLU) before passing to a sigmoid function for the final prediction.

3.1.2 RNN

Additionally, we include experiments with Recurrent Neural Networks as they suggest promising results with variable sized input and are capable of learning long-range dependencies [7]. We tested an RNN because we believe

that the ordering of amino acids might be significant, and LSTMs do a particularly good job of retaining information from earlier in a sequence. One very pressing issue with using vanilla RNN's is the vanishing gradient problem, due to which we are also training on LSTM's and GRU's as well. Both of them combat the vanishing gradient problem and share a lot of properties amongst themselves. According to empirical evaluations [10], there isn't a clear winner among both of them, which motivates us to try to experiment with both LSTM's and GRU's. The detailed motivation of using LSTM and GRU is described below

3.2 LSTM & GRU's Architecture:

What makes a normal RNN impractical is that it has to process and update every cell in the hidden layer. This is almost impossible given the complexity of RNN. The gated Recurrent units tries to circumvent this by using two other gates, u_i (indicates whether the hidden layer's value will change or not) and r_i (indicates whether hidden layer's value will update or not) gates. The RNN update function can be now written as:

$\tilde{h}_t = g(W\phi(x_i) + U(r \odot h_{t-1}))$, and

$h_t = (1 - u) \odot h_{t-1} + u \odot \tilde{h}_t$

where, h_t is the updated gate at time t,
g is non linear function like tanh $\phi(x_t)$ is the feature representation of the input $\odot$ is the element wise dot product W and U are the weight matrices. The above equation uses two extra gates which control the vanishing gradient problem. The equations used to tune the values of u_i and r_i are

$r = \sigma(W_r\phi(x_t) + U_r h_{t-1})$

$u = \sigma(W_u\phi(x_t) + U_u h_{t-1})$

Similar to GRU's, the LSTM uses the concept of revealing only some part of its hidden state by using two gates: input (i) and forget (f). So now the update state h_t is given by,

$h_t = o \odot tanh(c_t)$

$c_t = f \odot c_{t-1} + i \odot \tilde{c}_t$

the input and forget gates are updated in the same manner as reset and update gates for GRU's:

$$f = \sigma(W_f \phi(x_t) + U_f h_{t-1})$$

$$i = \sigma(W_i \phi(x_t) + U_i h_{t-1})$$

Chung et al. [4] explain how LSTM and GRU variants of RNN can outperform vanilla RNN's with practically any input setting. Hence, we believe its theoretical justified to implement them.

3.3 Convolutional Neural Networks (CNN)

Convolutional Neural Networks are known for their ability to extract local information very effectively by convolving a small kernel around the sequence representation. This can efficiently help us to segregate the sub functions inside the molecular functions which the sequences represent (which would be difficult to predict using RNN's due to overfitting, as they extract both short and long term dependencies without generalization).

We implemented CNN with different kernel sizes, channels, and strides to compare its performance on protein sequences. CNN seemed to perform well which can be seen from the images in the results section. This shows that, even if the sequences are long (or short), there are few small-sized patterns in the sequences which help the algorithm determine the GO terms of the sequence.

3.4 Loss

Each of our models used the same loss function. Based on maximum entropy, it determines a one-versus-all loss based between an 2-dimensional set of input predictions and set of target predictions of the same dimensions[1]. For each sample in the minibatch:

$$loss(x, y) = -\sum_i \left(y_i log(\frac{1}{1+exp(-x_i)}) + (1 - y_i) log(\frac{exp(-x_i)}{1+exp(-x_i)} \right)$$

To implement stochastic gradient descent we created batches of size 50, resulting in batch matrices with dimensions of 50 x maximum sequence length in batch x embedding dimensions.

3.5 Training and Hyperparameter Selection

All of our models were run separately on human sequences, yeast sequences, and an aggregation of human and yeast sequences. Moreover, we compared performance of the sequences alone against the performance when kmers were included. All models ran on graphical processing units (GPUs). In choosing the best version of each model, we optimized the following hyperparameters according to a random search strategy [3]: embedding dimensions, hidden size, L2 penalty, and learning rate. In addition to the weight

decay penalty, we implemented early stopping as an additional regularization method. For our final analysis, we selected models with lowest validation loss on the validation set.

3.6 Evaluation

Our primary metrics were i) area under the receiver operator characteristic curve (AUC) and ii) F1 score. Given the variation of our models by evaluation metric and the difficulty in interpreting the models in relation to these metrics, we analyzed how each metric responded to our models' outputs. (For more details, please refer to the Appendix). The most important takeaway from this analysis was the insight regarding F-score as a useful metric in protein function prediction.

F1 score is a widely accepted benchmark for CAFA submissions. However, F1 score may not be an perfectly ideal metric for this task because it places equal weight on precision and recall. Given the sparsity of the data set, it may make more sense to prioritize recall over precision. **Recall is more important than precision when a false negative is more costly than a false positive.** (Note that the formulas for recall and precision are $\frac{TruePositives}{TruePositives+FalseNegatives}$ and $\frac{TruePositives}{TruePositives+FalsePositives}$, respectively). For this task, a false negative would be failing to predict a Go Term for a given protein, while a false positive would be an incorrect prediction for a Go Term-protein pair. False positives are probably tolerable because biologists can double-check the model's work and catch incorrect predictions. However, false negatives might be very detrimental if it means that Go Terms for a protein are completely missed.

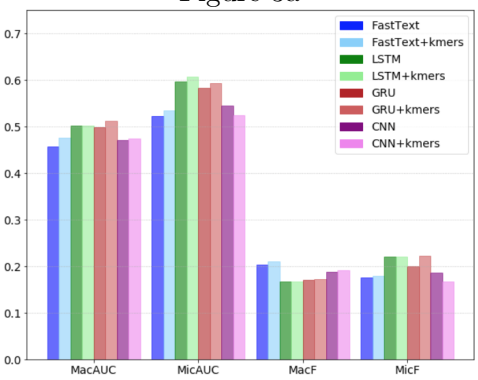
For this project, we follow convention and use F1 score. In the future, it might make sense to experiment with variants of F-score that place more weight on recall. One example might be F2 score, which equals $\frac{5*precision*recall}{4*precision+recall}$.

4 Results

Our results are shown in Figures 3a 3b. It's difficult to tell which model performs best when looking at the averaged results because relative performance changes depending on which metric is used. For yeast proteins, LSTM and GRU appear to perform slightly better in terms of AUC, but F-score results make the picture less clear. Similarly, no obvious patterns in model-to-model comparisons appear in the human proteins plot.

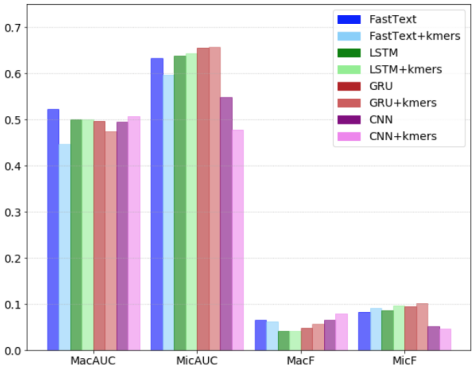
Nevertheless, there are some interesting takeaways. The average F-scores for yeast proteins (clustered around 0.20) are significantly higher than the average F-scores for human proteins (clustered between 0.05-0.10). We probably found better F1 scores for yeast proteins because the yeast GO terms

Figure 3a



(a) Results on Yeast Proteins

Figure 3b



(b) Results on Human Proteins

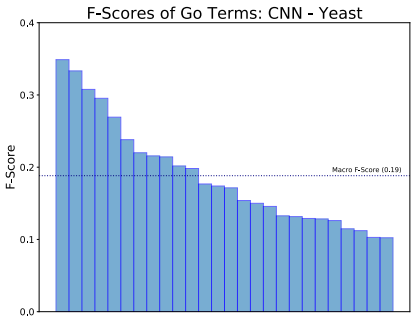
in our dataset tended to have a higher percentage of positive labels relative to most human GO terms.

Another pattern is that the Micro AUC scores are considerably higher than the Macro AUC scores for both human and yeast proteins. At first, we attributed this trend to the idea that individual GO terms with many positive labels may also have higher AUCs. However, when we compared the AUCs and positive label frequencies for each individual GO term, we found no significant correlation between a GO term's AUC and its "coverage ratio" (the number of positive labels). On a side note, we did find significant correlation between a GO term's coverage ratio and F1 score; we will discuss in more detail below. While the first explanation did not hold true, a second explanation for why Micro AUC was higher than Macro AUC is that the predicted outputs for some GO Terms were consistently higher than the predicted outputs for other GO Terms. For reasons described in the Appendix (under the "Micro vs. Macro-Level Evaluation" subsection), evaluating micro-level metrics for a matrix with columns that never overlap (e.g. one column has values between [0.4-0.5], another column's values are between [0.2-0.3]) can be misleading.

Performance per GO Term

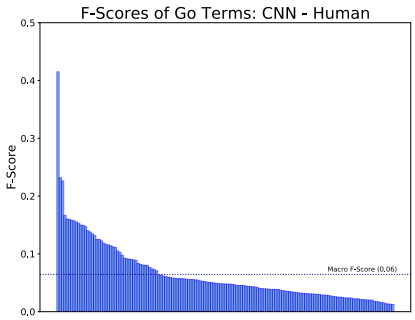
We also visualized performance for individual GO terms. Even though we trained our models in a multi-class setting – instead of training a binary classifier for each function – we still observe considerable variation across different GO terms. Figures 4a and 4b rank the performance of the CNN by F1-score for all GO terms in humans and in yeast. The F-scores for yeast proteins range from roughly 0.1 to 0.4, and for humans the F-scores range from 0.01 to 0.4. While only 20% of human GO terms have F-scores greater than 0.1, all yeast GO terms exceed this mark. As expected based on the summary charts from the previous section, the F-scores for individual yeast

Figure 4a



(a) Individual F-Scores on Yeast Functions by CNN Model

Figure 4b



(b) Individual F-Scores on Human Functions by CNN Model

GO terms were higher.

From figures 4a-4b, the question arises of whether there are significant differences between GO terms for which the model performed well and those that it predicted with a relatively low F-score. As such, we analyzed whether a GO term's F1 score was correlated with its "coverage" (percent of positive examples). The percentage of positive labels for each GO term ranges from 0.2% to 9.1% in humans and from 2.2% to 20.9% in yeast.

Figure 5 shows that there is significant correlation between the sparsity of the label and the ability of the model to assign it a relatively high F-score. In other words, GO terms with more positive labels tended to have higher F-scores. All correlations in Figure 5 have p-values below 0.01. As noted before, there was no significant correlation between a GO term's AUC and its coverage. Perhaps this is the case because AUC is more robust to changes in class distribution.

Since it seems that the certainty with which we can view model performance is somewhat correlated to the sparsity of labels, it could be useful for researchers who are looking for functional predictions to also know the label sparsity of a given function. That is, if we know that a certain label is relatively well populated in the data, then we can also be more certain that predictions about that label are reliable. Conversely, knowing that a certain label is not highly represented in the data would reduce our confidence in predictions for that label.

Performance by Function Type: Yeast Ontology

We also examined performance on GO terms based on their position in the hierarchy. Specifically, we analyzed a group of GO terms across three levels in the "Binding" branch. As shown in Table 1, GO terms closer to the top of the tree tend to have higher F-scores than their child nodes. This is not

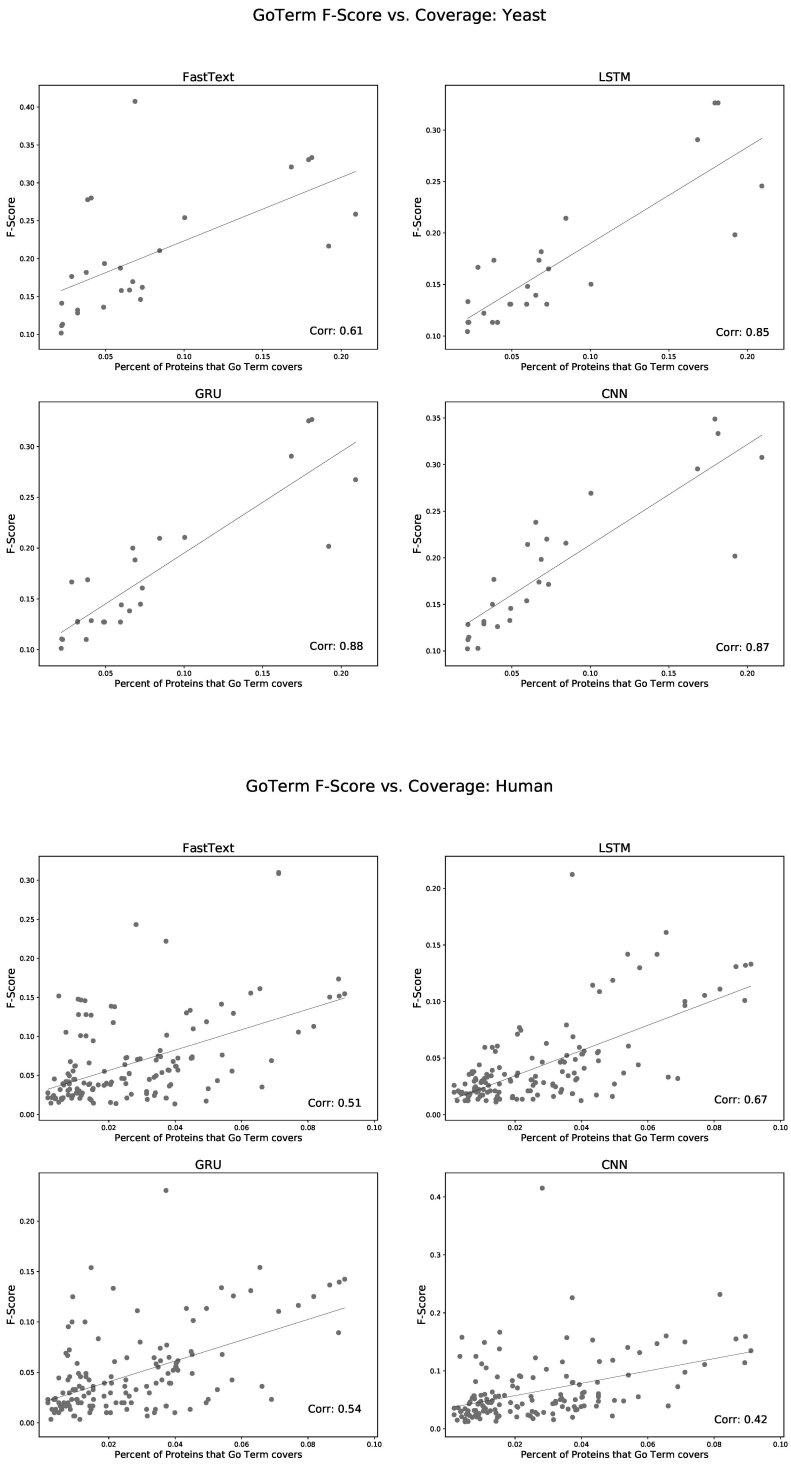


Figure 5: Correlation Between Coverage and F-score

	Positive Class Percentage	FastText	LSTM	GRU	CNN
Level 1					
Nucleic Acid Binding	16.8%	0.32	0.29	0.29	0.30
Level 2					
DNA Binding	6.9%	0.41	0.18	0.19	0.20
Level 3					
Reg. Region DNA Binding	2.2%	0.11	0.13	0.11	0.11
Double-Stranded DNA Binding	3.8%	0.18	0.11	0.11	0.15
Seq.Specific DNA Binding	4.1%	0.28	0.11	0.13	0.13

Table 1: F-Scores for each Go Term in branch of yeast ontology

surprising; upper-level functions have higher positive class percentages (by default), and in the previous section we found that F-scores are strongly correlated with positive class percentage. As such, one avenue for future research might be to determine the extent to which F-score differences between GO terms in the same branch can be attributed to i) mere differences in positive label coverage or ii) the hierarchical relationship between terms. Finally, we also measured the performance of the models on the three main functional categories in our set of yeast GO terms (Binding, Transport, and Catalytic). Of the 26 overall yeast labels, 12 fell under Binding, 5 under Transport, and 4 under Catalytic. We found that the models performed slightly worse for Transport functions than for Binding and Catalytic (see Appendix Figure 6). A more rigorous analysis might be useful, but from a preliminary standpoint, the F-scores for Transport functions might be lower because the positive label percentages for these GO terms was a bit lower (from 4%-8%).

5 Discussion and Next Steps

Using this simplified set of inputs, the models we built did not yield strong results, but their performance on each protein function label was related to how imbalanced the classes were for that label. It is possible that combining one of these models, particularly FastText, which trains quickly, could augment the performance of models that take a variety of inputs.

There are a number of additional steps that could be taken to assess these models on their own. Due to time constraints and competition for GPUs, we were not able to include homologs of our sequences. Including homologs would in essence amplify our training data and mitigate some of the issues we saw with imbalanced classes. With respect to the GRU and

LSTM models, we only applied unidirectional RNNs. Though they are more costly in terms of memory requirements, implementing bidirectional RNNs could improve performance of those models.

Finally, implementing an attention mechanism on the RNNs could be very promising. It is possible that large portions of these sequences are not meaningful to the classification task. Using attention could allow a neural net to improve its memory of what portions of the sequences are meaningful.

6 Acknowledgments

We are greatly appreciative to Dr. Richard Bonneau, Vladimir Gligorijević, Meet Barot, and Da Chen Emily Koo for their help with this research.

References

[1] Source code for torch.nn.modules.loss. http://pytorch.org/docs/master/modules/torch/nn/modules/loss.html#MultiLabelSoftMarginL Accessed: 2017-09-30.

[2] Berger, M. J. Large scale multi-label text classification with semantic word vectors.

[3] Bergstra, J., and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13* (2012), 281–305.

[4] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR abs/1412.3555* (2014).

[5] Fawcett, T. Introduction to roc analysis. 861–874.

[6] Gligorijević, V., Barot, M., and Bonneau, R. deepnf: Deep network fusion for protein function prediction. *bioRxiv* (2017).

[7] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Comput. 9*, 8 (Nov. 1997), 1735–1780.

[8] Jiang, Y., Oron, T. R., Clark, W. T., Bankapur, A. R., D'Andrea, D., Lepore, R., Funk, C. S., Kahanda, I., Verspoor, K. M., Ben-Hur, A., Koo, D. C. E., Penfold-Brown, D., Shasha, D., Youngs, N., Bonneau, R., Lin, A., Sahraeian, S. M. E., Martelli, P. L., Profiti, G., Casadio, R., Cao, R., Zhong, Z., Cheng, J., Altenhoff, A., Skunca, N., Dessimoz, C., Dogan, T., Hakala, K., Kaewphan, S., Mehryary,

F., Salakoski, T., Ginter, F., Fang, H., Smithers, B., Oates, M., Gough, J., Törönen, P., Koskinen, P., Holm, L., Chen, C.-T., Hsu, W.-L., Bryson, K., Cozzetto, D., Minneci, F., Jones, D. T., Chapman, S., BKC, D., Khan, I. K., Kihara, D., Ofer, D., Rappoport, N., Stern, A., Cibrian-Uhalte, E., Denny, P., Foulger, R. E., Hieta, R., Legge, D., Lovering, R. C., Magrane, M., Melidoni, A. N., Mutowo-Meullenet, P., Pichler, K., Shypitsyna, A., Li, B., Zakeri, P., ElShal, S., Tranchevent, L.-C., Das, S., Dawson, N. L., Lee, D., Lees, J. G., Sillitoe, I., Bhat, P., Nepusz, T., Romero, A. E., Sasidharan, R., Yang, H., Paccanaro, A., Gillis, J., Sedeño-Cortés, A. E., Pavlidis, P., Feng, S., Cejuela, J. M., Goldberg, T., Hamp, T., Richter, L., Salamov, A., Gabaldon, T., Marcet-Houben, M., Supek, F., Gong, Q., Ning, W., Zhou, Y., Tian, W., Falda, M., Fontana, P., Lavezzo, E., Toppo, S., Ferrari, C., Giollo, M., Piovesan, D., Tosatto, S. C., del Pozo, A., Fernández, J. M., Maietta, P., Valencia, A., Tress, M. L., Benso, A., Di Carlo, S., Politano, G., Savino, A., Rehman, H. U., Re, M., Mesiti, M., Valentini, G., Bargsten, J. W., van Dijk, A. D. J., Gemovic, B., Glisic, S., Perovic, V., Veljkovic, V., Veljkovic, N., Almeida-e Silva, D. C., Vencio, R. Z. N., Sharan, M., Vogel, J., Kansakar, L., Zhang, S., Vucetic, S., Wang, Z., Sternberg, M. J. E., Wass, M. N., Huntley, R. P., Martin, M. J., O'Donovan, C., Robinson, P. N., Moreau, Y., Tramontano, A., Babbitt, P. C., Brenner, S. E., Linial, M., Orengo, C. A., Rost, B., Greene, C. S., Mooney, S. D., Friedberg, I., and Radivojac, P. An expanded evaluation of protein function prediction methods shows an improvement in accuracy. *Genome Biology 17*, 1 (Sep 2016), 184.

[9] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. Bag of tricks for efficient text classification. *CoRR abs/1607.01759* (2016).

[10] Jozefowicz, R., Zaremba, W., and Sutskever, I. An empirical exploration of recurrent network architectures. *Journal of Machine Learning Research* (2015).

[11] Kim, Y. Convolutional neural networks for sentence classification. *CoRR abs/1408.5882* (2014).

[12] Radivojac, P., Clark, W. T., Oron, T. R., Schnoes, A. M., Wittkop, T., Sokolov, A., Graim, K., Funk, C., Verspoor, K., Ben-Hur, A., Pandey, G., Yunes, J. M., Talwalkar, A. S., Repo, S., Souza, M. L., Piovesan, D., Casadio, R., Wang, Z.,

Cheng, J., and Fang, H. A large-scale evaluation of computational protein function prediction. *Nature Methods 10*, 3 (2013), 221 – 227.

[13] Zhou, C., Sun, C., Liu, Z., and Lau, F. C. M. A C-LSTM neural network for text classification. *CoRR abs/1511.08630* (2015).

7 Appendix

The main metrics we use to evaluate model performance are i) area under the receiver operator characteristic curve (AUC) and ii) F1 score. Both of these metrics are commonly used in CAFA submissions. We also analyze both metrics at the micro and macro level.

F1 Score

F1-score is the harmonic mean of Precision and Recall, or $\frac{2*precision*recall}{precision+recall}$. In line with other CAFA submissions, we computed F_{max}, which is found by testing a series of thresholds in [0, 1] and seeing which threshold t maximizes F1 Score. To be more specific, a threshold t determines if a prediction should be considered positive or negative. For example, if $t = 0.2$, and our initial output vector is [0.22, 0.10, 0.90], our final predictions would be [1, 0, 1]. We used the thresholds generated by Python's scikit-learn package.

Area Under the Curve

AUC reflects the trade-off between True Positive Rate (TPR) and False Positive Rate (FPR). More intuitively, AUC can be interpreted as the probability that the model ranks a randomly-selected positive example ahead of a randomly-selected negative example [5]. For instance, if our model predicts the scores [0.5, 0.6, 0.1, 0.2] and the actual output is [1, 0, 0, 0], the AUC is 0.67 because the probability that the positive example (0.5) is ranked ahead of a randomly selected negative example (0.6, 0.1, or 0.2) is 2/3. Theoretically, a random classifier would have an AUC of 0.5. We record AUC because it is a useful metric for problems with imbalanced class sizes and has a well-defined baseline (0.5).

Micro vs. Macro-Level Evaluation

We compute micro- and macro-level statistics for both AUC and F1 score. Macro-level AUC is calculated by finding the AUC scores for each individual Go Term and then taking the mean of those values. Micro-level AUC is found by vectorizing the protein-function prediction matrix and binary label matrix into one-dimensional columns, and then computing the AUC from those two new vectors, similar to [6]. Macro- and Micro- F1 score are calculated in a similar manner.

Both micro- and macro-level metrics have potential pitfalls. Macro AUC, as an unweighted average, might be problematic if certain Go Terms have more positive examples than other Go Terms. For instance, let GoTerms X, Y, Z have AUCs of 0.8, 0.2, and 0.2. Then the Macro AUC would be (0.8+0.2+0.2)/3 = 0.4, well below a random classifier. However, what if Go Term X had 20 positive examples and Go Terms Y and Z had only one positive example each? In this case, computing the unweighted Macro AUC may be a suboptimal approach. Micro AUC is more useful for evaluating the global effectiveness of a model. However, there are also cases when Micro AUC can be problematic. This is especially true when the predicted outputs in one column (or Go Term) are consistently higher than the outputs in a different column. Consider the example below, with predictions on the left and labels to the right. Each column is a Go Term and each row is a protein:

$$\begin{bmatrix} 0.50 & 0.49 & 0.25 & 0.02 \\ 0.51 & 0.42 & 0.29 & 0.20 \\ 0.60 & 0.40 & 0.22 & 0.21 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After vectorizing these matrices, we find that the Micro AUC is 0.43, even though the model appears to do a good job of scoring positive examples higher than negative examples in each of the final three columns. When different columns have non-overlapping ranges of values, Micro AUC can be a problematic metric. During our experiments, we found many cases where the predicted output values for one column (Go Term) were higher than the values from a different column. For example, the output values from one Go Term might range from [0.2, 0.3], while those from another Go Term might range from [0.1, 0.2]. These differences could arise if one Go Term appears much more frequently than another Go Term. If Go Term A appears in 25% of proteins, but Go Term B appears in only 2% of examples, the model will probably learn that Go Term A is more common and thus produce higher prediction scores for entries in that column than entries in Go Term B's column.

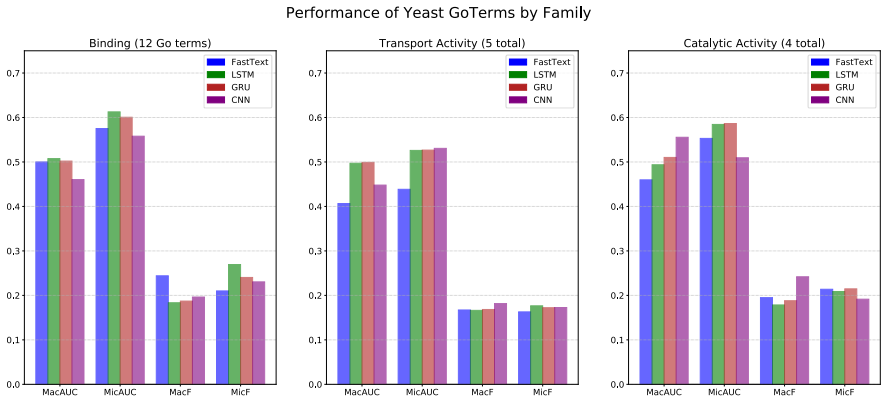


Figure 6: Evaluation Metrics Broken Down by GO Term Branch