Sequence analysis

# Sequence-based protein function prediction using convolutional neural networks

**Ali Abbas**

Department of Computer Science, University of Central Florida, Oviedo, FL 32765, United States.

**Abstract**

**Motivation:** Genome sequencing advances in recent years have resulted in huge amounts of protein sequence data. To make use of such data, we first need to map each protein sequence to relevant biological processes, cellular components, or molecular functions. This is known as Gene Ontology (GO) term annotation prediction problem. In other words, the problem is that given a protein sequence data as input, how to predict its GO term annotations. Manual labeling of data is very time consuming so we need to automate this process.
**Results:** Recently, Zuallaert *et al.* (2019), studied the effects of different encoding strategies on protein function prediction performance using a Convolutional Neural Network (CNN). I reimplemented a reduced scope of their work by selecting only one-hot encoded and ad-hoc trainable embeddings encoding strategies and dropping trigrams. I was unable to get the same results despite using the same parameters but got the same trends of seeing simpler encodings performing better. Next, in order to improve the results, I augmented the unigram one-hot encoded input with six chemical properties of the amino acids and got slightly better results than my first implementation, especially for BP and MF functions, which might be due to BP and MF having stronger correlation to chemical properties of amino acids compared to CC. Then, in order to further improve my results, I used the Tree-structured Parzen estimators (TPE) to search for best hyperparameters, which resulted in better performance but still not matching the original paper's results. Finally, I used a bottleneck model and was able to match the original paper's results for CC functions, and get to 1.2% distance from MF results. The final model's improved results could be due to introducing an information bottleneck before the last layer which causes the model to learn to generalize from limited data.
**Availability:** Source code: https://github.com/ali-i-abbas/protein_function_prediction
**Contact:** ali.abbas@knights.ucf.edu

## 1 Introduction

Protein function prediction has many important applications in understanding diseases and drug target discovery. Despite its importance, the process of annotating proteins with their function is very labour intensive and time consuming, therefore finding an automated solution will accelerate developments in this area.

Various methods have been proposed for predicting protein functions from their sequence data. One of the first methods that has been used successfully for prediction protein functions is based on sequence similarity which mainly uses BLAST. Hennig *et al.* (2003) developed the GOblet software that uses BLAST to search a database of proteins and assign the Gene Ontology (GO) terms of similar sequences to the unknown sequence. More recently deep learning has been successfully applied to protein function prediction. Kulmanov *et al.* (2018b) used ad-hoc trainable embeddings of trigrams as input for a Convolutional Neural Network (CNN). It also incorporates hierarchical GO information to improve the results. Zuallaert *et al.* (2019) used different encoding strategies for the input to a CNN and studied the biological significance of these encodings and its effects on prediction results.

Here, I reimplement unigram and bigram one-hot encoding and ad-hoc trainable embedding encoding schemes from (Zuallaert *et al.*, 2019) in conjunction with the same CNN model used by them. In order to improve the results of the implementation, I experimented with various approaches including: augmenting chemical properties of the amino acids with the unigram one-hot encoding, hyperparameter tuning with Tree-structured Parzen estimators (TPE), and a hybrid of previous implementations.

## 2 Methods

### 2.1 Dataset

I used the same dataset that was used by Zuallaert *et al.* (2019) and Kulmanov *et al.* (2018b) and can be accessed from (Kulmanov *et al.*, 2018a). Protein sequences from SwissProt dataset were filtered based on their annotation's experimental evidence code (EXP, IDA, IPI, IMP, IGI, IEP, TAS and IC), maximum sequence length (1002), and not having ambiguous amino acid codes (B, O, J, U, X, Z). Furthermore, a minimum number of annotations were required for each GO class (50 for MF or CC and 250 for BP). The final dataset has 589 MF classes with 25224 training samples and 6306 testing samples, 932 BP classes with 36380 training

Table 1. Baseline CNN model structure. Output shape is for MF model with unigram one-hot encoding

| Layer | Output Shape |
|---|---|
| Encoding | (1002, 21) |
| Conv + Relu (64 filters of size 9, dropout = 0.2) | (994, 64) |
| Max Pooling (size 3, stride 3) | (331, 64) |
| Conv + Relu (64 filters of size 7, dropout = 0.2) | (325, 64) |
| Max Pooling (size 3, stride 3) | (108, 64) |
| Conv + Relu (64 filters of size 7, dropout = 0.2) | (102, 64) |
| Max Pooling (size 3, stride 3) | (34, 64) |
| K Max Pooling (K = 10) | (10, 64) |
| Flatten | (640) |
| Fully Connected (32 units) | (32) |
| Sigmoid Output (One for each GO class) | (589) |

samples and 9096 testing samples, and 439 CC classes with 35546 training samples and 8887 testing samples

## 2.2 Input representation

Following (Zuallaert *et al.*, 2019) the input to the model is the n-gram of the sequence of amino acid (AA) codes, that then is encoded using one of the following approaches:

- **One-hot encoding:** for each term of the n-gram, assign a vector consisting of all zeros, except for a one at the position reserved for that term.
- **Ad-hoc trainable embedding:** initializes a random vector of size v for each term of the n-gram and v is learned along with other network parameters during training

## 2.3 Baseline comparison method

The model developed by (Zuallaert *et al.*, 2019) is used for comparing the results. It's a CNN that consists of one encoding layer, followed by layers of convolution, Rectified Linear Unit (ReLU), drop out, and max pooling, that are repeated 3 times. Next, there's a dynamic K-max pooling layer, that is used to produce a fixed output size regardless of the input size by selecting k largest values in each channel while preserving their original order. Finally, there's a fully connected layer with a sigmoid output for each GO class. Binary cross entropy loss and Adam optimizer with learning rate 0.001, batch size of 64, and 15 epochs were used for training the models. 87.5% of the dataset was used for training and the remaining 12.5% for validation. Table 1 shows the baseline model structure with parameters taken from (Zuallaert *et al.*, 2019). It also shows the output dimensions for a model trained with MF data that uses the unigram one-hot encoding.

## 2.4 Amino acid chemical properties augmentation method

The function of a protein depends on its amino acids' chemical properties, so by augmenting the sequence data with these properties, the input representation should provide more useful information that can be used in the function prediction task. The following chemical properties were selected (Szalkai and Grolmusz, 2018):

1. Expected charge: (1: positive, -1: negative, 0: neutral)
2. Hydrophobicity: from -4.5 to 4.5
3. Polar: yes or no (1 or 0)
4. Aromatic compound: yes or no (1 or 0)
5. Has a hydroxyl group: yes or no (1 or 0)
6. Has a sulfur atom: yes or no (1 or 0)

Table 2. TPE CNN model structure. Output shape is for MF model with unigram one-hot encoding augmented with chemical properties of amino acids

| Layer | Output Shape |
|---|---|
| Encoding | (1002, 27) |
| Conv + Relu (512 filters of size 13, dropout = 0.1) | (990, 512) |
| Max Pooling (size 5, stride 4) | (247, 512) |
| K Max Pooling (K = 15) | (15, 512) |
| Flatten | (7680) |
| Fully Connected (512 units) | (512) |
| Sigmoid Output (One for each GO class) | (589) |

These can easily be obtained from the amino acid letters. The one-hot encoded unigram (20 dimension) is augmented with these 6 numbers, so that the input will be a 26 dimension vector (27 dimensions if the one-hot encoding of the null data is included) for each amino acid in the sequence. The model used here is the same as the baseline model with the same parameters shown in Table 1 with the exception of the input which is augmented with chemical properties of the amino acids.

## 2.5 Model found by hyperparameter optimization method

Hyperparameters have a significant impact on the performance of CNN models. There are several methods of automatically searching for best hyperparameters, including grid search which tries all combinations of parameters, random search which randomly selects combinations of parameters, and Tree-structured Parzen Estimators (TPE) which constructs models with the given parameters and based on the performance of these models, selects the next parameters for model construction, therefore efficiently searching the parameter space. It constructs a tree of estimators based on the hyperparameters (Bergstra *et al.*, 2011).

I selected the following ranges of values for the following parameters to be searched by TPE:

1. Convolutional Layer:
   - number of layers = [1, 2, 3, 4]
   - number of filters = [32, 64, 128, 256, 512]
   - kernel sizes = [3, 5, 7, 9, 11, 13]
   - max pooling size = [2, 3, 4, 5, 7, 9, 11]
   - strides = [1, 2, 3, 4, 5]
   - dropout rate = [0, 0.1, 0.2, 0.3, 0.4, 0.5]
2. K Max Pooling:
   - k = [1, 3, 6, 8, 10, 15, 20, 40, 80]
3. Fully Connected Layer:
   - number of layers = [1, 2]
   - number of units = [32, 64, 128, 256, 512, 1024, 2048]
   - dropout rate (not included in last layer) = [0, 0.1, 0.2, 0.3, 0.4, 0.5]

I ran the TPE with 10 epochs and batch size 64 for a total of 300 evaluations on MF data. The structure of the model found by TPE is shown in Table 2, which also shows the output dimensions for a model trained with MF data that uses the unigram one-hot encoding augmented with chemical properties of amino acids. Instead of multiple layers of convolution, TPE preferred to have one large convolution layer followed by aggressive max pooling to reduce the input dimension, and used a much larger number of fully connected units in comparison to the baseline model.

Table 3. Bottleneck CNN model structure. Output shape is for MF model with unigram one-hot encoding with chemical properties of amino acids

| Layer | Output Shape |
|---|---|
| Encoding | (1002, 27) |
| Conv + Relu (64 filters of size 9, dropout = 0.2) | (994, 64) |
| Max Pooling (size 2, stride 2) | (497, 64) |
| Conv + Relu (64 filters of size 7, dropout = 0.2) | (491, 64) |
| Max Pooling (size 2, stride 2) | (245, 64) |
| Conv + Relu (64 filters of size 7, dropout = 0.2) | (239, 64) |
| Max Pooling (size 2, stride 2) | (119, 64) |
| Conv + Relu (64 filters of size 5, dropout = 0.2) | (115, 64) |
| Max Pooling (size 3, stride 3) | (38, 64) |
| K Max Pooling (K = 10) | (10, 64) |
| Flatten | (640) |
| Fully Connected (2048 units) | (2048) |
| Sigmoid Output (One for each GO class) | (589) |

## 2.6 Bottleneck CNN method

Bottleneck in a neural network is a layer with lower dimensions than the layers before and after it, and has been shown to improve the performance of the network by forcing the network to learn a more efficient representation of the data that is less affected by changes in the input (Veselý *et al.*, 2011).

The best models found by TPE had shown an interesting trend in having a large number of fully connected units, which inspired me to use a large fully connected layer that can create a bottleneck by having the previous layer to have smaller dimensions. In order to create this bottleneck, I used multiple layers of convolution to reduce the input dimension. The reason for choosing multiple convolution layers was that by choosing different kernel sizes we can extract features at different scales, which can lead to a more effective representation of important information in the dataset. The model's structure is shown in Table 3, which also shows the output dimensions for a model trained with MF data that uses the unigram one-hot encoding augmented with chemical properties of amino acids. The k-max pooling layer acts as the bottleneck for the network.

## 2.7 Evaluation

Evaluation metrics are based on metrics defined by Critical Assessment of protein Function Annotation (CAFA) challenges (Jiang *et al.*, 2016) and is computed as follows:

$$p(t) = \frac{1}{m(t)} \sum_{i=1}^{m(t)} \frac{f_i(t)}{g_i(t)},$$

$$r(t) = \frac{1}{n} \sum_{i=1}^{n} \frac{f_i(t)}{h_i},$$

$$F_{max} = \max_t \left\{ \frac{2p(t)r(t)}{p(t)+r(t)} \right\},$$

where $t \in (0, 1)$ is the threshold that converts the sigmoid output to 0 or 1 as predicted label, $p(t)$ is the average precision for a given $t$, $m(t)$ is the number of sequences with at least one prediction above threshold $t$, $f_i(t)$ is number of correct predictions for a given $t$ for sequence $i$, $g_i(t)$ is the number of predictions for a given $t$ for sequence $i$, $r(t)$ is the average recall for a given $t$, $n$ is the total number of sequences, $h_i$ is the number of target labels for sequence $i$, and $F_{max}$ is the maximum $F$-score obtained by varying $t$ from 0 to 1.

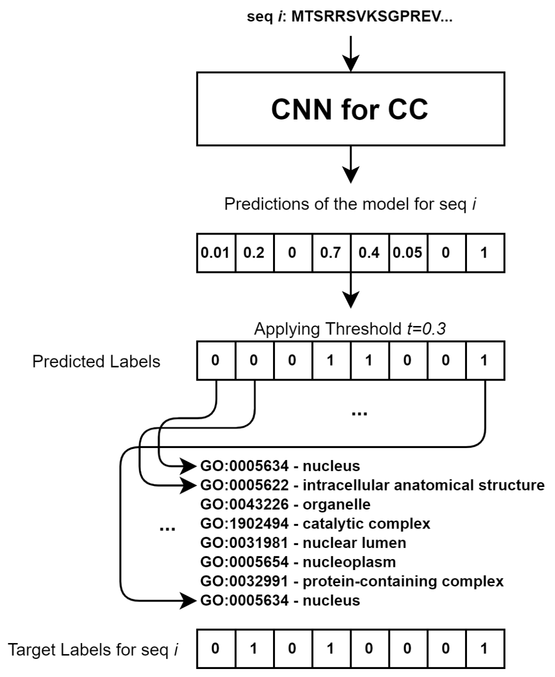
seq *i*: MTSRRSVKSGPREV...
CNN for CC
Predictions of the model for seq *i*
0.01 0.2 0 0.7 0.4 0.05 0 1
Applying Threshold *t=0.3*
Predicted Labels
0 0 0 1 1 0 0 1
...
GO:0005634 - nucleus
GO:0005622 - intracellular anatomical structure
GO:0043226 - organelle
GO:1902494 - catalytic complex
GO:0031981 - nuclear lumen
GO:0005654 - nucleoplasm
GO:0032991 - protein-containing complex
GO:0005634 - nucleus
Target Labels for seq *i*
0 1 0 1 0 0 0 1

**Fig. 1.** Evaluation process for a sample sequence for CC data

Figure 1 shows the process of evaluation for a sample sequence. After applying the threshold to the sigmoid output of the CNN, we get a vector of zeros and ones, where the presence of a 1 in a particular position indicates that the sequence belongs to that class. Comparing the predicted labels with the target labels (ground truth) allows us to calculate the $F_{max}$ when the process is repeated for all sequences and thresholds. Since we have infinite values between 0 and 1, in practice the threshold is incremented by 0.01 from 0 to 1, and the $F$-score is calculated for that t. The final $F_{max}$ is calculated as the maximum of all calculated F-scores.

## 3 Results

I ran the experiments on a Windows 10 Laptop with Intel Core i7-4720HQ CPU @ 2.6 GHz, 16 GB RAM, and Nvidia GeForce GTX 970M Graphic Card. Python 3.7 with Tensorflow 2.1 was used to code the models. For all the models, a separate model is trained for BP, MF, and CC. Each result is reported as the average of 10 runs including the corresponding 95% confidence intervals. The following sections will present the results of the proposed models.

## 3.1 Baseline model implementation

I used the same model and parameters as the baseline to run the experiments. Figure 2 shows the comparison of the results reported by (Zuallaert *et al.*, 2019) on the left side and the results of my implementation of the baseline model on the right side which also include the 95% confidence intervals shown by the error bars.

I was not able to get the same results as (Zuallaert *et al.*, 2019) which can be due to some implementation errors on my part or some inaccuracies in the reported parameters by (Zuallaert *et al.*, 2019), but without having access to their code, it is difficult to find the exact cause of this discrepancy.

Although I was unable to get the same results but there are some general trends that are still manifested in my results similar to (Zuallaert *et al.*, 2019). The first one is the fact that increasing $n$ for $n$-grams reduces the $F_{max}$, which is not very surprising since increasing $n$, results in more sparse input which makes the task of finding efficient representation more

Table 4. Comparison between results of baseline from (Zuallaert et al., 2019) with my baseline, my baseline with chemical properties, TPE with chemical properties, and Bottleneck with chemical properties for the unigram one-hot encoding

| Function | Baseline | My baseline | My baseline with chemical properties | TPE with chemical properties | Bottleneck with chemical properties |
|---|---|---|---|---|---|
| BP | 0.358 | 0.310 (+/- 0.00458) | 0.317 (+/- 0.00137) | 0.316 (+/- 0.00329) | 0.317 (+/- 0.00229) |
| MF | 0.411 | 0.383 (+/- 0.00515) | 0.393 (+/- 0.00376) | 0.402 (+/- 0.00329) | 0.406 (+/- 0.00281) |
| CC | 0.595 | 0.582 (+/- 0.00335) | 0.589 (+/- 0.00203) | 0.589 (+/- 0.00257) | 0.597 (+/- 0.00259) |

difficult. The second trend is that for unigram ad-hoc trainable embedding, increasing the vector size $v$, had no impact on $F_{max}$. This indicates that the unigram can be adequately represented by a vector of size 5 and increasing $v$ results in no improvements. The last trend is that for bigrams increasing $v$ results in some small improvement up to a point but stops shortly after. This is because the size of the bigrams vocabulary are much larger than unigrams which requires larger $v$, but because of the redundancy present in bigram representation, continuing to increase $v$ will not yield better results.

## 3.2 Amino acid chemical properties augmentation results

The results of augmenting the input to the baseline model with chemical properties of the amino acids is shown in Table 4, which also compares the results to the baseline model results for the unigram one-hot encoding from (Zuallaert *et al.*, 2019) and my implementation of the baseline model for the unigram one-hot encoding.

Although it still didn't manage to reproduce the baseline model's results, it improved the results of my implementation of the baseline model by 2.3% for BP, 2.6% for MF, and 1.2% for CC. It's noticeable that the largest increases in performance were for BP and MF, which might be due to slightly stronger correlation of biological processes (BP) and molecular functions (MF) with chemical properties of amino acids compared to cellular components (CC). The effect is small because the chemical properties that I used were easily deducible from the sequence such that a CNN model can implicitly learn them automatically, but this might be a hint that using other chemical properties that are more complex might have greater impact on the performance, since the model have immediate access to these useful but hard to extract features and does not need to learn them from scratch.

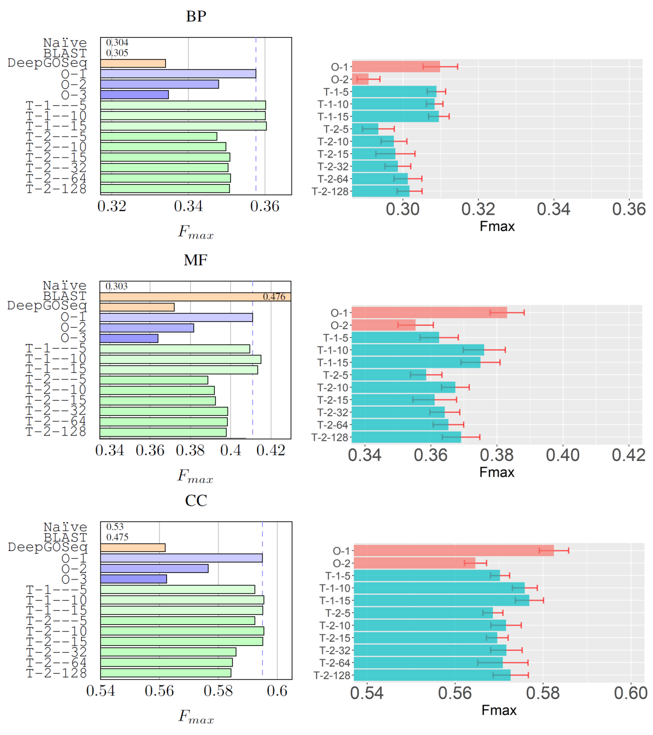

**Fig. 2.** Comparison between baseline model from (Zuallaert et al., 2019) on the left side and my implementation on the right side. The error bars on the right side show the 95% confidence intervals.
Note. Left hand side graphs are from Fig. 5 from (Zuallaert et al., 2019)

## 3.3 TPE model results

The results of the model found by TPE is shown in Table 4, which also compares the results to the baseline model results for the unigram one-hot encoding from (Zuallaert *et al.*, 2019) and my implementation of the baseline model for the unigram one-hot encoding with augmented chemical properties of the amino acids.

The results of the TPE model didn't show a significant improvement over the previous model of baseline with augmented chemical properties, except for 2.3% improvement for MF. This is to be expected since TPE was run on MF data and found the best parameters for MF.

## 3.4 Bottleneck model results

The results of the bottleneck model is shown in Table 4, which also compares the results to the baseline model results for the unigram one-hot encoding from (Zuallaert *et al.*, 2019) and all previously mentioned implemented models.

Although it still didn't manage to replicate the baseline model results from (Zuallaert *et al.*, 2019) for BP, but got almost the same results for CC and the difference for MF was only 1.2%. In addition it was able to improve the results of my previous implementations for MF and CC. More specifically, it improved the results of the TPE model for MF by 1% and for CC by 1.4%. The improvement over my baseline with chemical properties for MF was 3.3% and for CC was 1.4%. For BP it got the same results as my baseline with chemical properties. The reason for the improvement over TPE being small is that TPE is similar to the bottleneck model in having a large fully connected layer which was my inspiration for choosing the bottleneck model in the first place. The difference is their performance is due to the effectiveness of having multiple convolutional layers in extracting useful information and the bottleneck effect in making the network more resilient to input variations.

## 4 Conclusion

Given the importance of protein function prediction in many areas of bioinformatics, it is very important to have automated systems that can accurately and efficiently predict the protein function's from their sequence data.

Deep learning methods have shown great success in solving difficult prediction tasks. A recent example of employing deep learning for solving the protein function prediction is the work done by (Zuallaert *et al.*, 2019).

I implemented my baseline model based on their work and experimented with different models in order to improve their results. I first attempted to reproduce their implementation exactly but did not succeed in achieving the same results. Although I found the same trend as the baseline results in that using larger $n$ for $n$-grams lowered the $F_{max}$ score and ad-hoc trainable embedding didn't offer better results than one-hot encoding. Larger $n$-grams are more sparse and result in more network parameters that need to be learned and thus lower the model's performance. The added complexity of adding an additional encoding layer for ad-hoc trainable embedding was shown to not result in better performance which could be due to having more parameters that need to be trained compared to a simple unigram one-hot encoding.

In my second model, adding chemical properties of amino acids to the input, showed improvement over my baseline model for BP and MF, but not for CC. This difference between protein functions performance could be due to a more correlation between chemical properties of amino acids and BP and MF compared to CC. This model was still not successful in reproducing the original baseline model's results.

For my third model, I used TPE to find the best model over the defined parameters search space using MF data. The TPE model not surprisingly performed best on MF compared to the previous model, but didn't reproduce the original baseline model's results. Even though TPE might not give us the best results but it can point us to an unexplored area of the parameter space for further manual tuning, which lead me to developing my final model.

## References

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24. Neural Information Processing Systems Foundation.

Hennig, S., Groth, D., and Lehrach, H. (2003). Automated gene ontology annotation for anonymous sequence data. *Nucleic Acids Research*, **31**(13), 3712–3715.

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., *et al.* (2016). An expanded evaluation of protein function prediction methods shows an improvement in accuracy. *Genome biology*, **17**(1), 1–19.

Kulmanov, M., Khan, M. A., and Hoehndorf, R. (2018a). Dataset for deepgo. http://deepgo.bio2vec.net/data/deepgo/data.tar.gz. Accessed: 2021-02-11.

Kulmanov, M., Khan, M. A., and Hoehndorf, R. (2018b). Deepgo: predicting protein functions from sequence and interactions using a deep ontology-aware classifier. *Bioinformatics*, **34**(4), 660–668.

Szalkai, B. and Grolmusz, V. (2018). Near perfect protein multi-label classification with deep neural networks. *Methods*, **132**, 50–56.

Veselỳ, K., Karafiát, M., and Grézl, F. (2011). Convolutive bottleneck network features for lvcsr. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 42–47. IEEE.

Zuallaert, J., Pan, X., Saeys, Y., Wang, X., and De Neve, W. (2019). Investigating the biological relevance in trained embedding representations of protein sequences. In *Workshop on Computational Biology at the 36th International Conference on Machine Learning (ICML 2019)*.