BIN Assignment 5: Protein Function Prediction

Protein Annotation with Recurrent Neural Networks

Matouš Soldát, 28. 5. 2023

1 Introduction

The assigned task is to try and predict function of proteins based on their primary structure (sequence of amino acids). Data from The Gene Ontology [1] are to be used to train and test the proposed solution. To solve the assignment, I chose the method of protein annotation with recurrent Neural Networks. My solution follows the recommended procedure outlined by Ronak Vijay in [2].

2 Data

2.1 GO term selection

Three GO Database terms were selected for the assignment. Two of the terms are more closely related – *regulation of cell population proliferation* (GO:0042127) and *regulation of apoptotic process* (GO:0042981), which share multiple common ancestors in the ontological tree: *regulation of cellular process*, *regulation off biological process* other more distant ancestors. The third term is *DNA damage response* (GO:0006974), which only shares one common ontological ancestor which is the root node – *biological process*.

The assumption is that the more closely related terms will be significantly more difficult to distinguish than unrelated terms because proteins with similar function are assumed to be more similar in primary structure.

2.2 GO term extraction

The data were extracted from the provided *CAFA3_training_data* dataset using an R script implemented in *select_GOterms.R* which builds upon the provided example in script *preprocess_CAFA3.R*. The implemented script requires the user only to specify the *GO_id* of the desired GO term and extracts all proteins from the database associated with either the selected GO term or any of its child terms. All extracted terms are then saved in one fasta file.

2.3 Creating training datasets from extracted GO terms

The extracted data files corresponding to the different selected GO terms were joined into training datasets and split into training and testing partitions using a Python script implemented in *split_data.py*. The script only requires the user to list all fasta files which are to be mixed into a dataset and select a name for the dataset to be saved as. The script can be used to create datasets with arbitrary ratio of training data to validation data and with an arbitrary number of inputed fasta files. Each fasta file is treated as a different class in the resulting training dataset. Each class is split with the selected ration into training and validation datasets (i.e. the ratio of different classes is the same in training and validation datasets).

3 Methods

Two separate CNNs were designed as proposed in [2], LSTM CNN and ProtCNN. The core ideas behind the CNNs are described in chapters 3.1 and 3.2.

3.1 LSTM design

The first CNN is a bidirectional LSTM (Long Short-Term Memory) which is a variation of RNN (Recurrent Neural Network). RNNs are neural networks with the ability to remember and take into account a short context. Bidirectional LSTM improves upon that concept by using the context from the entire past and future (the entire sequence in the case of protein prediction) [2].

The input of the LSTM CNN is a vector with values 0 to 20 and a set length of 100 values. The individual values from 1 to 20 correspond to the 20 most common amino acids found in proteins. The value 0 corresponds to one of the rarer amino acids. All protein sequences in the dataset are either cut off or padded by zeros to make them exactly 100 symbols long. The LSTM CNN has a typical softmax output, that is a vector of length equal to the number of defined classes (different protein families) with values between zero and one.

3.2 ProtCNN design

The second implemented CNN is ProtCNN, a deep neural network which solves the problem of vanishing gradient with a residual block inspired by ResNet architecture which introduces shortcut connections from the lower to the higher layers of the network [2]. The two CNNs are implemented in */lib/model/LSTM.py* and */lib/model/protCNN.py*, respectively. The input of the ProtCNN is a one-hot encoded version of the input of the LSTM CNN. That is an array of size 100 x 21, because each amino acid in the 100-long sequence is encoded with a one-hot vector of length 21. The ProtCNN has the same typical softmax output as the LSTM CNN.

3.3 Training and testing

The implemented CNN designs were trained to distinguish between each possible pair of the three selected GO terms using the attached Python script *main.py*. At the top of the script, the user can choose which CNN should be used, max length of a sequence, number of training epochs and batch size. Batch size of 256 and 15 epochs were used for training of the ProtCNN. Batch size 256 and 33 epochs were used or the training of the LSTM CNN. The batch size and a larger number of epoch for the LSTM model are recommended in [2]. Alternatively, the CNNs can be trained and tested in the attached Python Jupyter Notebook *main.ipynb*, which is better suited for exploratory purposes.

The *main.py* script loads the selected training dataset, converts the data into the correct format for the selected CNN, initializes the CNN model and trains it with the predefined batch size and number of epochs. The model's loss and accuracy are evaluated during each epoch on test data which are never used for the training. Categorical cross-entropy is used as the loss function for both of the CNNs during the training.

4 Results

The evolution of the loss value as well as prediction accuracy of the models during the training is plotted in figures below. The resulting accuracy of each trained model on test data is listed in table 1.

Accuracy on test data	LSTM CNN	ProtCNN
Apopptosis | Damage	0.6591	0.5545
Population | Damage	0.6485	0.5900
Apoptosis | Population	0.5095	0.5361

Table 1: Test accuracy of the CNNs trained on different data

Apoptosis = *regulation of apoptotic process*

Damage = *DNA damage response*

Population = *regulation of cell population proliferation*

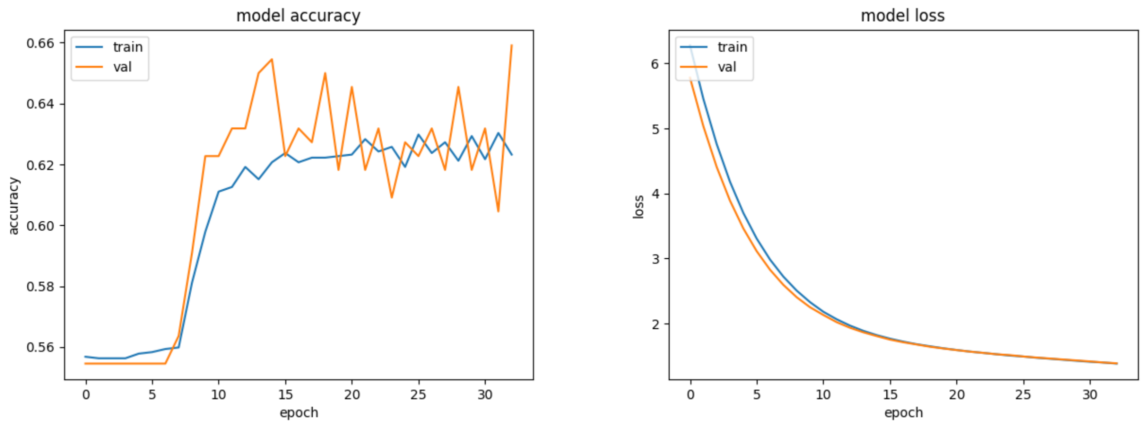

Figure 1: LSTM model loss and accuracy during training on apoptosis and damage

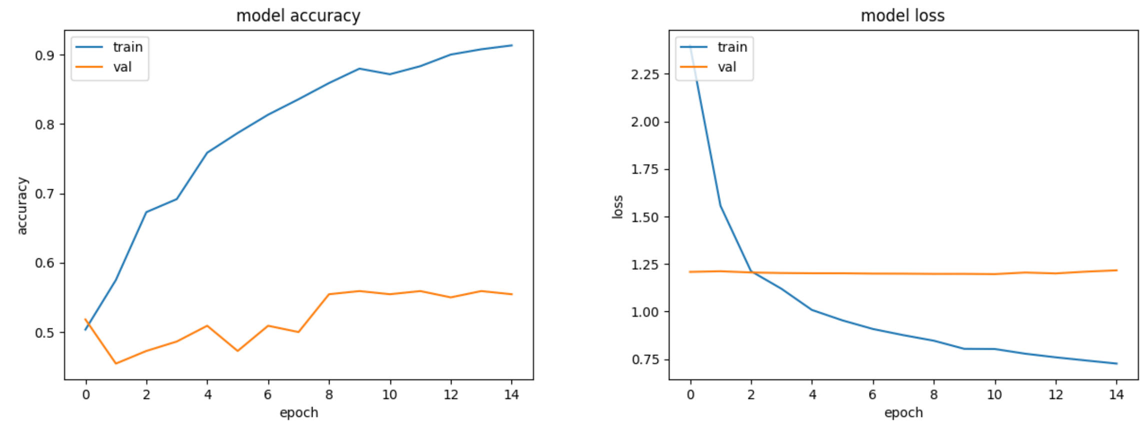

Figure 2: ProtCNN model loss and accuracy during training on apoptosis and damage

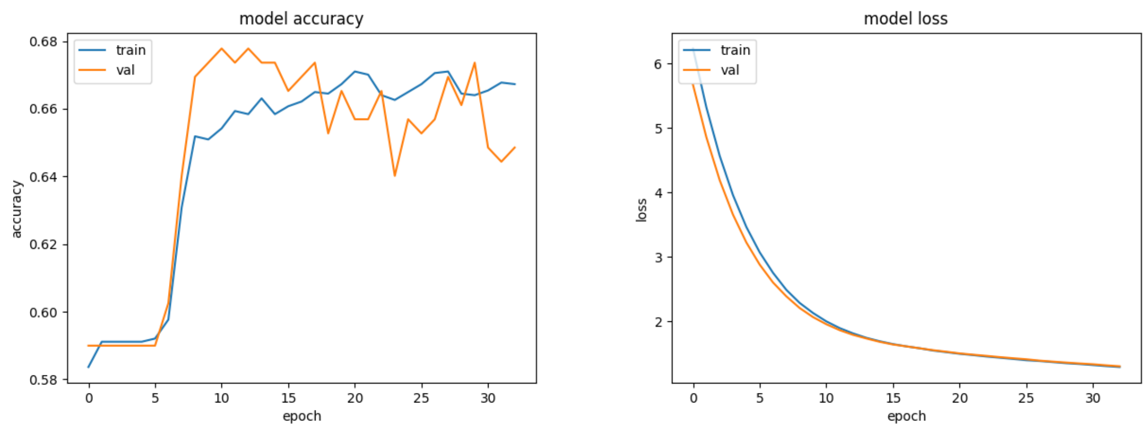

Figure 3: LSTM model loss and accuracy during training on population and damage

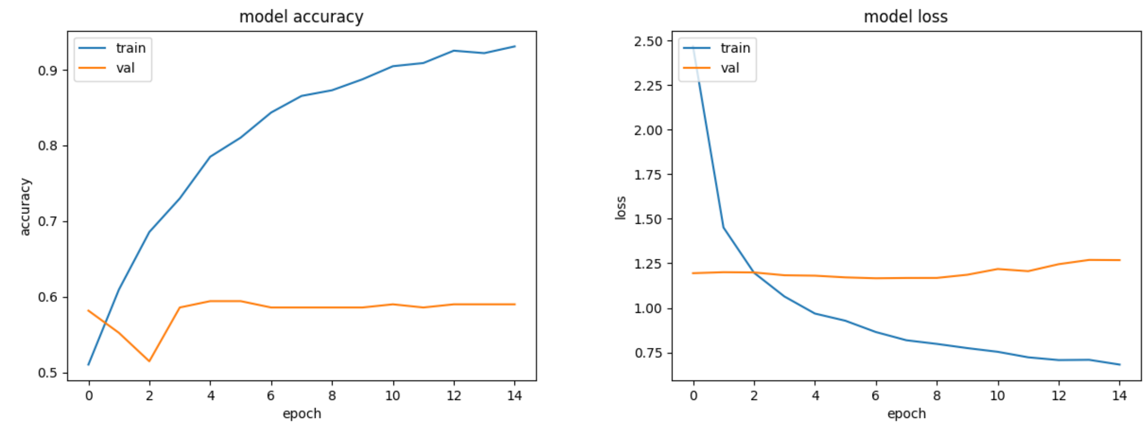

Figure 4: ProtCNN model loss and accuracy during training on population and damage

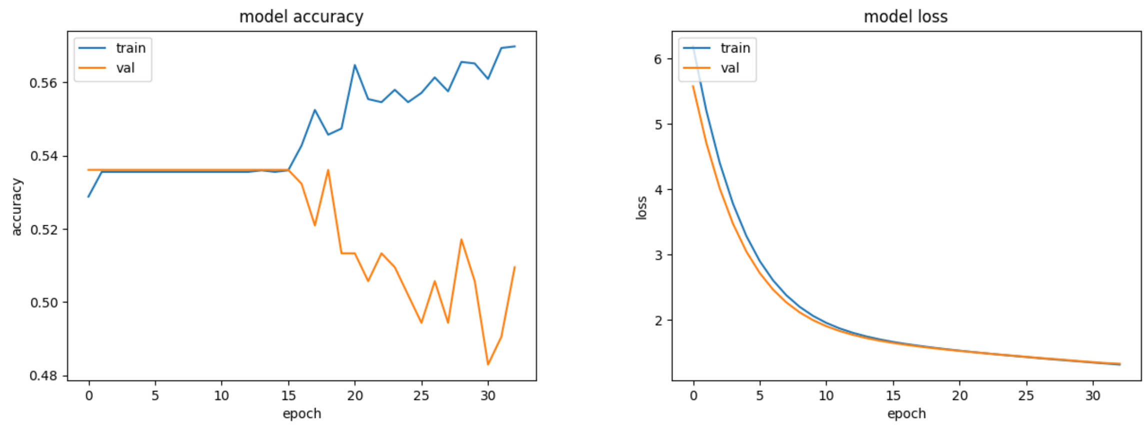

Figure 5: LSTM model loss and accuracy during training on apoptosis and population

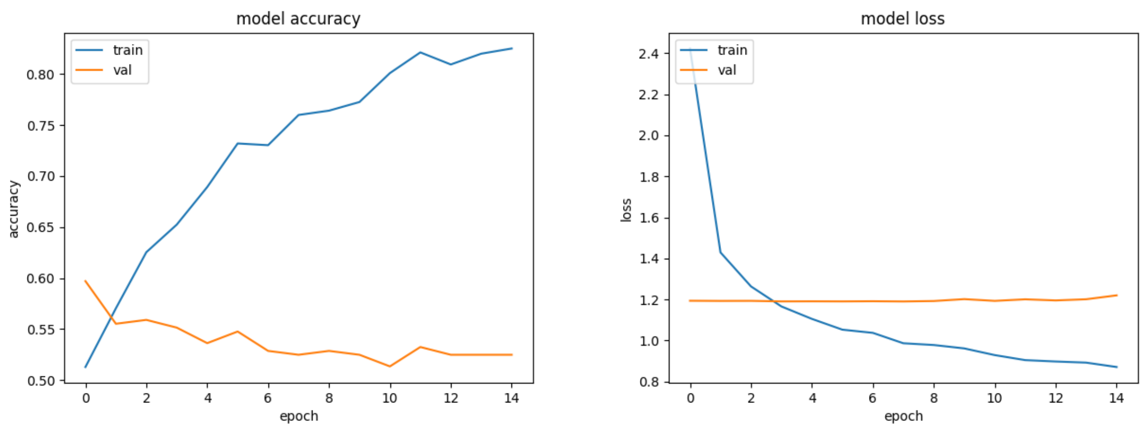

Figure 6: ProtCNN model loss and accuracy during training on apoptosis and population

5 Discussion

For the two more unrelated pairs of terms (apoptosis and damage, population and damage), the LSTM CNN had an accuracy well over 0.5, which shows that CNNs can help predict functions of proteins. The obtained accuracy of approximately 0.65 is, however, too low for the LSTM CNN to be usable in practice in its proposed and implemented form. In this analysis, the procedure proposed in [2] was accepted with no changes to the design and no exploration of effects of different parameters on its performance. With a proper fine-tuning and/or an exploratory analysis of similar CNN designs, the LSTM architecture might produce satisfactory results.

The accuracy of the ProtCNN on the unrelated pairs of terms was 0.55 and 0.59 which is fairly low, bordering on the 0.5 accuracy of a trivial constant model (assuming same representation of the two classes). It can be seen from the plotted evolution of the accuracy during the training that the training accuracy of ProtCNN was very high compared to the training accuracy of the LSTM design. It could be speculated that the ProtCNN is prone to overfit the training data and that the overfitting could be partially suppressed by fine-tuning of the internal parameters of the ProtCNN design which is outside of the scope of this analysis. Based on the results of the design proposed in [2] alone, the LSTM architecture seems to yield superior results to the ProtCNN architecture in the task of protein function prediction.

The accuracy of both of the CNNs on the pair of the more closely related terms was close to 0.5. This result is agreement with the initial assumption. Distinguishing between the related terms proved to be more difficult than distinguishing between the very different terms. It can be further observed from the plotted evolution of the accuracy during training that the models wildly overfitted the training data when training to distinguish the closely related terms. This can be seen from the opposite changes in training accuracy and validation accuracy. It hints at the fact that the differences between sequences with the same function and the differences between sequences with the two different, but closely related functions might be very similar and it might not be possible to distinguish between two similar functions of proteins from the primary structure of the proteins alone.

References

[1] The Gene Ontology. http://geneontology.org/.

[2] Ronak Vijay. Protein Sequence Classification. A case study on the Pfam dataset to classify protein families. 2019. Available at https://towardsdatascience.com/protein-sequence-classification-99c80d0ad2df.