Final Semester Project Report: Deep Learning in Computational Biology course

BIU, 2024

Hornstein Maor, Mino Hila

Link to detailed source code: git.
Please execute the evaluator.py file that is compatible with the assignment requirements. The required libraries, documented tested hardware, and running instructions are provided in the README file.

Main Goal

RNA binding proteins (RBPs) are special molecules that interact with certain parts of RNA called pre-mRNA and mature mRNA. They help with the processing of pre-mRNA and also regulate important functions like translation, localization, and stability of mature mRNA.
Studying how proteins and RNA interact helps us better understand how gene expression is controlled after transcription.
To determine if a protein binds to a specific RNA sequence, we look for a specific pattern within the sequence. RBPs can help us identify these patterns by analyzing sets of RNA sequences to which they were found to be bound.
In this work we will use RBNS data to build a deep neural network model that predicts protein-RNA binding. This model will be used to make predictions on the RNCMPT dataset.

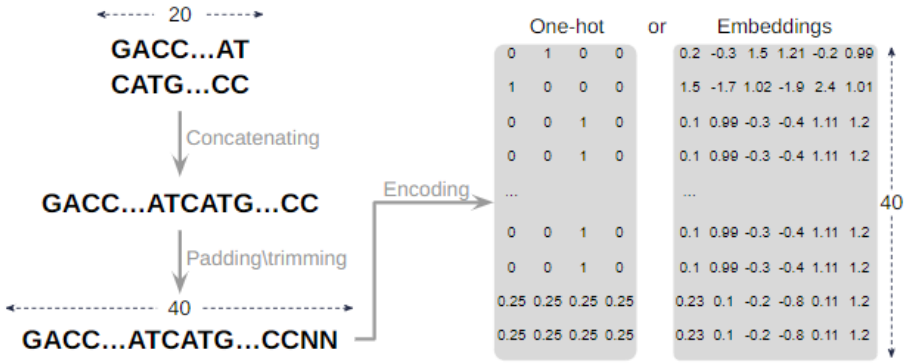
Input and Output of program

As input, we have a collection of 16 RBPs referred to as RBP1-RBP16.
For the RBNS dataset, each RBP is associated with 4-6 files. These files include one file without RBP concentration and additional files with increasing RBP concentrations. Each file contains approximately 10 to 20 million RNA sequences that were bounded by the RBP, with each sequence being around 20 base pairs long.
In the case of the RNCMPT dataset, we have a single file consisting of 250,000 RNA sequences ordered lexicographically. There are 16 score files, each corresponding to a specific RBP and indicating its binding intensity.
We train a neural net to predict whether a given RNA sequence will be bounded by a particular RBP or not using the RBNS dataset. Then, we apply the model to the RNCMPT dataset, where we refer to the binding probability as the intensity score. Our objective is to achieve a high correlation between the predicted probability and the actual binding intensities. The resulting output, therefore, includes three items: (1) a neural net model, (2) the model's learning graphs (accuracy and loss), and (3) the binding probabilities.

Pre and Postprocessing of the Data

The system is given 4-6 files containing RBNs data to train a classification model. The sequences in the file with zero concentration are read and labeled as negative, whereas the sequences in the remaining files are read and labeled as positive.

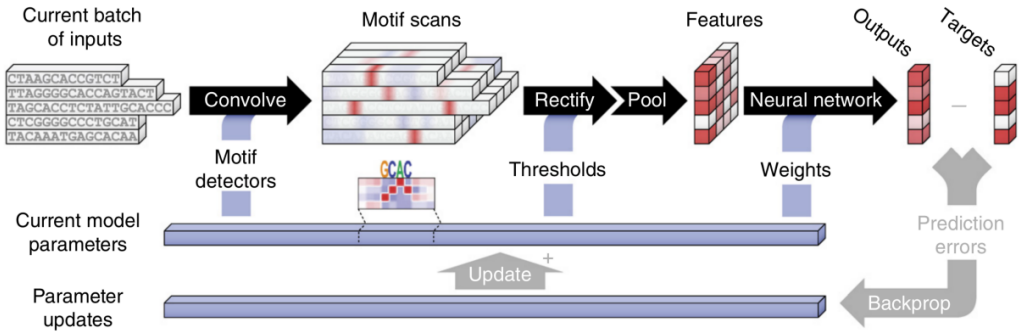
We set two hyperparameters to determine the dataset structure: set_size, which determines the number of positive and negative samples to load, resulting in a data size of 2 times set_size. Additionally, we explored four options for combining the RBPs files for the positive data: We named this hyperparameter "mode" and its options are HIGH (using the file with the highest concentration), LOW (using the file with the lowest concentration), WEIGHTED_HIGH (loading more information from files with higher concentrations), and WEIGHTED_LOW (loading more information from files with lower concentrations).
RBNS samples (mostly 20 nucleotides long) are concatenated to form a 40-nucleotide long sequence, trimmed or padded with N nucleotides if needed. RNCMPT samples (mostly 40 nucleotides long) are only trimmed or padded if needed.
The nucleotides are encoded using either one-hot vectors of size 4 or learned embeddings. In the one-hot vector representation, the unknown N nucleotide is encoded as {0.25, 0.25, 0.25, 0.25}.



As for the post processing, the neural net model generates binding probabilities ranging from 0 to 1. We use these probabilities directly without any additional processing.

Neural Net Architecture

Our architecture is similar to the one presented in the DeepBind paper as presented below:



Once the input is encoded, it goes through a series of steps: a convolution layer, rectification (we used ReLU layer), pooling, and finally, feeding into a deep neural network that has been trained to classify whether the outcome is positive or negative.

Compared Architectures and Our Final Selection

We examined additional architectures we read about in the scientific literature, including LSTM and Bilstm, as well as using multiple kernels of varying lengths. However, these approaches had a disadvantage in terms of training time: LSTM, in particular, required significantly more training time. As for the different kernel sizes, we observed that it also slowed down the training process, although not to the same extent as LSTM. Because we

found that using only constant kernel size gave us similar results in terms of classification accuracy (under time and hardware constraints), we decided to stick with this approach.
We also considered different options for determining the negative train-set. Initially, we randomly created sequences as negative samples. However, we decided to change our approach and use sequences from the input file as negative samples instead. The reasoning behind this decision was that although both options can serve as negative samples that meet the criterion of non-binding sequences, the sequences in the input file contain some information about the characteristics of the DNA sequence that random sequences lack.

Parameter Search and Training

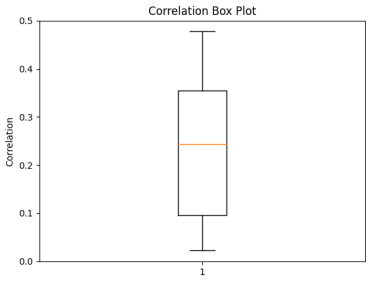
Throughout the parameter search process, we conducted 63 experiments for the exploration of diverse hyperparameter combinations. For each combination, we assessed the model's performance on the RNAcompete dataset by analyzing the resulting r2 score. Additionally, we evaluated the model's classification scores, namely accuracy, f1, and loss, for the bind-n-seq data.
The decision to assess the model's prediction capabilities, in addition to the r2 score, was motivated by the objective of ensuring that we were evaluating a functional network that had undergone proper training and successfully learned from the available data.
Furthermore, we recorded the training time, as well as the memory and CPU usage, for each model. If the training process exceeded one hour, it was terminated.
The table below displays the potential hyperparameter values and the final chosen hyperparameter values:

Parameter domain	**Calibration parameter**	**search space**	**Sampling**	**Chosen value**
Training parameters	Epochs	200 (or up to 1 hour training time)	fixed	2
	Batch Size	256, 128, 64	Uniform	64
	Initial learning rate	0.0001, 0.001, 0.01, 0.0005, 0.005, 0.05	Uniform	0.001
	Optimizer	Adam	fixed	Adam
	Kernel Batch normalization	True, False	Bernouli (p=0.5)	False
	Network batch normalization	True, False	Bernouli (p=0.5)	False
	Regularization L1_L2	[0.1, 0.001, 0.0001, 0] for combination of L1 and L2	Uniform	0
Network Parameters	Kernel size	5, 7, 9, 11, 15	Uniform	5
	Kernel out channels	256, 512, 1024	Uniform	800
	Pooling size	2, 3, Global	Uniform	Global
	Dropout	0, 0.25, 0.5	uniform	0
	Hidden layers	32, 64, 128, combination up to 2 and 3	Uniform	[800]
Data Parameters	Data Type	Low, High, relative-high, Relative Low	uniform	High
	Set site	2000000 (1000000 Positive and 1000000 negative)	fixed	1000000
	RBP	1-16	Uniform	

The experiments helped us determine the best configuration: we found that normalization and regularization are unnecessary, and that the learning process converges in just a few epochs, so 2-3 epochs are sufficient. We also learned that a moderate learning rate is required.
Yet the important discovery that led to the current configuration was understanding that using a kernel of size 5 gives good results. With that in mind, we initially used 1024 kernels to account for all possible combinations of sequences-encoding with a length of 5 (4^5=1024). However, we realized that not all of these options were useful, and using all of them would take too much time for training. So, we experimented with reducing the kernels until we found that using 800 kernels works well without harming the scores. The results of each kernel out of the 800 were reduced to a single number using a max pooling layer. Then, we added a hidden layer of the same size to examine the relationship between the found sequences, and finally, included a final layer for the prediction.
We experimented with multiple variations on top of this configuration, yet this setup produced the best results.

Results on Training Data

When evaluating the model on the RNCMPT dataset containing the 16 proteins, we reached a median correlation of 2.4:



The model performed best in detecting the binding of protein 13, showing a correlation of 0.477. For protein 3, the model exhibited the lowest performance with a correlation of 0.022.

Performance (time, memory, CPU)

Machine Configuration: 2 vCPUs, 7.5 GB of RAM, and 10 gigabytes of persistent storage.
The script used to evaluate a single RBP is called "evaluator.py." Its average performance, measured while evaluating RBPs 1, 3, and 13, is as follows:
Real Time: 47.37 minutes.
User CPU Time: 46.52 minutes.
System CPU Time (Sys): 1.006 minutes.
Memory % usage: 0.1%.

Conclusion

While our work didn't achieve state-of-the-art scores, we proposed and evaluated novel ideas, mainly in the data preprocessing aspect. As for the educational aspect, we explored and experimented with different models like LSTM and deep networks, and encoding methods such as one-hot and embedding. Ultimately, we learned to conduct and present consistent deep learning research, thanks to the research and papers presented and taught in class.

Links to results:

- Results for train set: predictions + training graphs, correlation calculation and illustration
- Results for test-set: RNCMPT Test-Set Prediction (also attached as part of the submitted files).
- Results for experiments: The 63 experiments configurations and their results