Gene Annotation LSTM Network

Siddarth Harinarayanan
CSE-523 Report
Stony Brook University

sharinarayan@cs.stonybrook.edu

Abstract

This project mainly focuses on predicting genomes, given DNA sequences based on the existence of start and stop codons. Primarily, this report includes deep learning network architecture which consists of LSTM model whose output is sent to a couple of fully connected layers. This model trains on the DNA sequences and predicts the annotation for genes. The architecture of the network and the suitable hyper parameters for the best model is explained in detail below with different experiments.

1 Related Work

The research on Genome Annotation problem is one of the most prominant in the field of genomics and it dates back to several years. It is the process of marking an assembled genome of different species with a rich collections of features to help understand how the structure of the sequences contributes to the classification of genomes.
I started by focusing on the start codons and stop codons in the DNA sequence which are essentially combination of three bases/nucleotides. The problem is further complicated because these start and stop codons can occur anywhere in the frame that is read. Each of the frames is really huge and searching for the codons mannually would be a tedious task as such. Also using basic algorithmic search approaches to identify the frame as genes would a great challenge. To devise a better and viable approach to solve this problem, I focused on how to come up with an efficient deep learning model to classify the given frames.

Below are the two approaches for Genome annotation in the deep learning world:

[1]**(Amin et al., 2018) Bi-directional LSTM Model:** In the paper titled, DeepAnnotator: Genome Annotation with Deep Learning, the authors have designed and proposed a network with pre-trained embeddings, Bi-directional LSTM, and multiple dense layers. In this paper, they have trained the embeddings using [3]gensim Word2Vec with CBOW on 2B bases from NCBIs bacterial genomes, breaking up the DNA sequences with "N"s and discarded the data points which have number of bases less than 20. The network is as follow, the first layer uses the pre-trained 100 dimension Kmer embeddings. Embedding vectors are passed through two bi-directional LSTM layers each stacked upon one another. Then, they had the output of Bi-LSTM passed through three Fully connected layers with sigmoid one after the other. Finally, the network performs binary classification on the vector representation of each data point for two classes, start codon and stop codon. On fine tuning the hyper-parameters the accuracy of the above proposed model they achieved was close to 95.6%.

[2]**(Khodabandelou et al., 2018) Convolutional Network Model:** In the paper titled, Genome Functional Annotation using Deep Convolutional Neural Networks, the authors have designed and proposed a network with convolutional network model with max pooling, and multiple dense layers. In this paper, they have fed the input sequence using the one-hot encoding of the DNA sequence. This input is passed into CNN Filter and CNN layer, then into max pooling layers. The output of CNN is then passed into multiple fully connected layers with sigmoid. They have achieved an accuracy of 83% of the above proposed model.

2 Dataset

The dataset consists of two files, positive and negative samples. Positive samples are the genes and negative samples are intragenic. Each of the records in the sample is a string of 101 nucleotides/bases where the mapping is as follows: *A:0, C:1, G:2, T:3*. There are close to 1.5 million data points for positive dataset and 2.2 million for the negative dataset. Initial data set for experiments are same amount of positive and negative data points. These two datasets are stacked/concatenated upon each other with corresponding labels and the data is shuffled This shuffled data is then split as training and testing data points in the measure of 75 percent and 25 percent of the selected amount of data respectively. The *train_test_split* function in sklearn shuffles the data and returns the training and testing data based on the split that we specify. For this project the train and test split is set as 75 percent and 25 percent respectively. These data points are then used for training the model, and eventually run the test data on it.

3 Network Architecure

I have used pytorch modules/frameworks to implement the neural network. The first layer of network is the embedding layer, where embeddings are trained at each epoch. These embeddings are sent into LSTM model with two layers stacked one after the other. The output of the LSTM layer is fed into a multilayer perceptron, which consists of two fully connected layer with dropouts after each layer. Eventually, sigmoid is applied to the last fully connected layer output and based of the probability, the classification is made. The part of fully connected layer is explained in detail in Aravind's Report. The LSTM that I have implemented in this project is the version of LSTM that pytorch in-built provides, where we set the number of layers required which are stack on top of one another before a fully connected layer. LSTM layer helps the training model in memorizing some knowledge regarding the occurrences of start/stop codons or ribosome bindings.

3.1 Input and Output Shapes at each Layer:

3.1.1 Variables

- Embedding Size
- Hidden Layer Size: 99
- Kmer Size: 3
- Number of LSTM layers

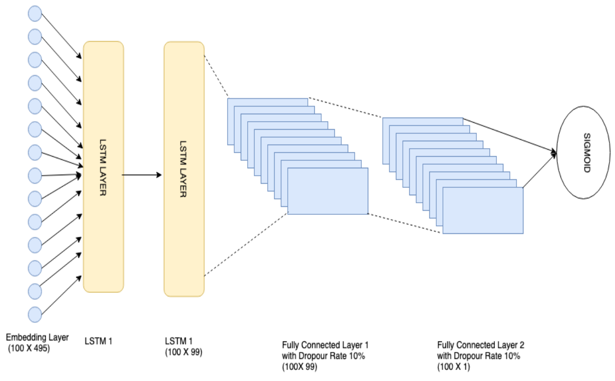
LSTM LAYER
LSTM LAYER
SIGMOID
Embedding Layer (100 X 495)
LSTM 1
LSTM 1 (100 X 99)
Fully Connected Layer 1 with Dropour Rate 10% (100X 99)
Fully Connected Layer 2 with Dropour Rate 10% (100 X 1)

Figure 1: LSTM with FC Network

3.1.2 Generalized Network Architecture

- LSTM((101 - KmerSize + 1)*embed_size, (101 - KmerSize + 1), num_layers)
- (fc1): Linear(in=(101 - KmerSize + 1), out=(101 - KmerSize + 1))
- (dropout): Dropout(p=0.1)
- (sigmoid): Sigmoid()
- (fc2): Linear(in=(101 - KmerSize + 1), out=1)

4 Experiments

After performing a series of experiments from Table 1, with hyper-parameter tuning we observed the following trend. The accuracy just drops down when we increase the dropout rate. In general, LSTM model works better with larger dataset. Hence we observe the trend that when we increase the data points the accuracy is better. From the below experiment table, for 500 thousand dataset, the model gives lesser accuracy than for 1.5 million data points. I conclude that more the data, better the accuracy.

5 Conclusion

The main motivation factor of this project is to design and implement an efficient deep learning model using LSTM network and improve the gene annotation accuracy. From the above experiments, we can conclude that increasing the datapoints improves the accuracy. The model achieved the best accuracy as 95.03% from experiment number 7. For future work we can enhance this network by integrating Pavan's attention model on top of LSTM model.

SNo	pos_d	neg_d	layers	lr	dropout	Acc.(train/test)
1	500K	500K	2	7%	10%	94.90 / 93.54(5)
2	1.5M	2.2M	2	7%	10%	95.09 / 94.94(5)
3	1.5M	1.5M	2	7%	15%	91.79 / 94.60(5)
4	1.5M	1.5M	2	7%	25%	86.72 / 94.58(5)
5	1.5M	1.5M	2	10%	10%	94.32 / 94.68(5)
6	1.5M	1.5M	2	20%	10%	94.18 / 94.78(5)
7	1.5M	2.2M	3	7%	10%	95.04 / 95.03(5)
8	1.5M	2.2M	4	7%	10%	95.14 / 94.97(5)
9	1.5M	2.2M	3	20%	10%	94.97 / 94.98(5)
10	1.5M	2.2M	2	7%	10%	95.86 / 94.54(15)

Table 1: Experiments
(embedding size in the last column)
pos_data: Positive Data
neg_data: Negative Data
layers: LSTM layers
lr: Learning Rate
dropout: Dropout Rate

The code for the model along with network visualization can be found here.

References

Mohammad Ruhul Amin, Alisa Yurovsky, Yingtao Tian, and Steven Skiena. 2018. Deepannotator: Genome annotation with deep learning. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, BCB '18, pages 254–259, New York, NY, USA. ACM.

Ghazaleh Khodabandelou, Etienne Routhier, and Julien Mozziconacci. 2018. Genome functional annotation using deep convolutional neural networks. *bioRxiv*.

Gensim: https://radimrehurek.com/gensim/models/word2vec.html