Final Project Progress Report for -

COM S 574
Intro to Machine Learning

Spring 2021

Submitted By

Md Sakib Ferdous
Chemical and Biological Engineering Department

Ibne Farabi Shihab
Computer Science Department

Iowa State University

Submitted To

Dr Forrest Bao,

Course Teacher

Abstract

This project is divided into two parts, one is the classification part and other one is the generation part. For the first part of the project, we proposed a deep learning-based protein sequence classifier while on the other part we proposed a GPT-2 and GAN based novel protein generation. Here the classifier will be trained on a dataset consisting 37000 annotated protein sequence with 4 types of deep neural nets – first the only embedding layer, then CNN, LSTM and combination of LSTM and embedding layer. The performance of each type of algorithm is presented with validation score and confusion matrix. In the later part we showed a way to generate novel protein sequence with use of pretrained GPT-2 and GAN. In this work, for the first portion, we found greatest success with CNN based classifier. The LSTM based classifier seems to over fit and lots of fluctuation in validation score is observed. For the generation part, we were able to generate some novel sequence which are subject to lab testing for the verification.

Contents

1. Introduction

1.1 Background

Proteins are chains of amino acids arranging themselves along the backbone alpha carbons and folds to give 3D structures like alpha helix of beta sheets. The function and efficacy of proteins largely depends on the arrangement of amino acids which results in three dimensional structures. There are different zones in the structure with the help of which proteins are able to carry out many functions, like breaking bonds, forming bonds, carrying out reactions etc. Proteins are classified based on the type of task they perform. Classifying proteins based on their attributes is called the protein classification problem.

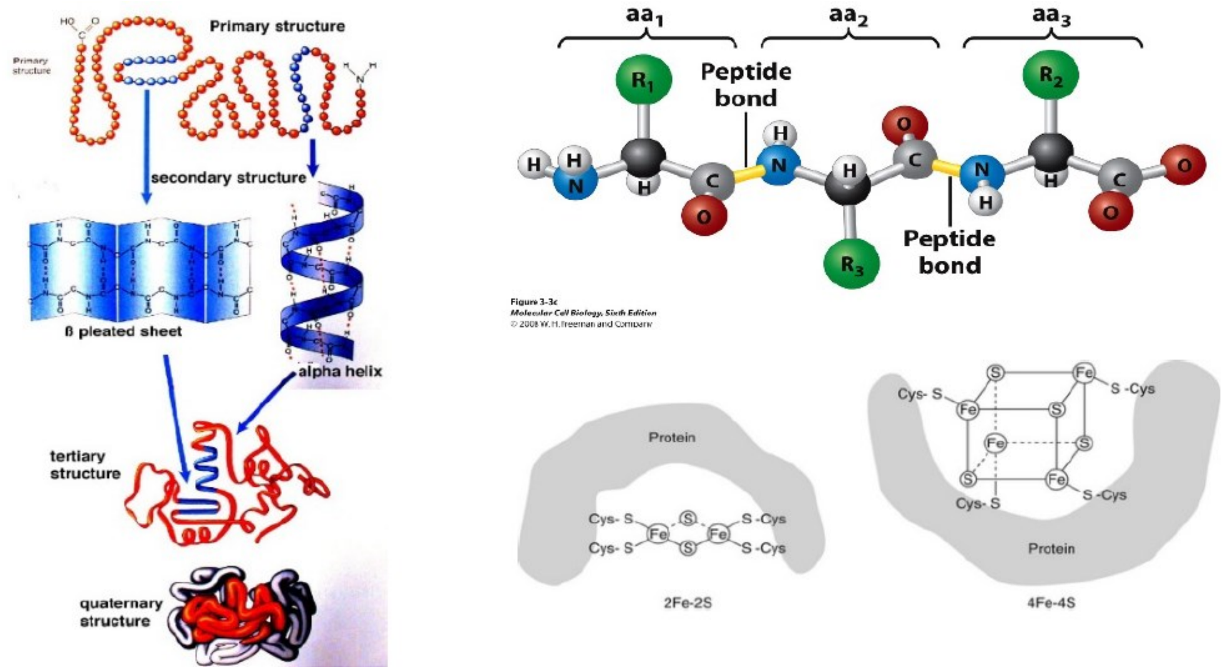
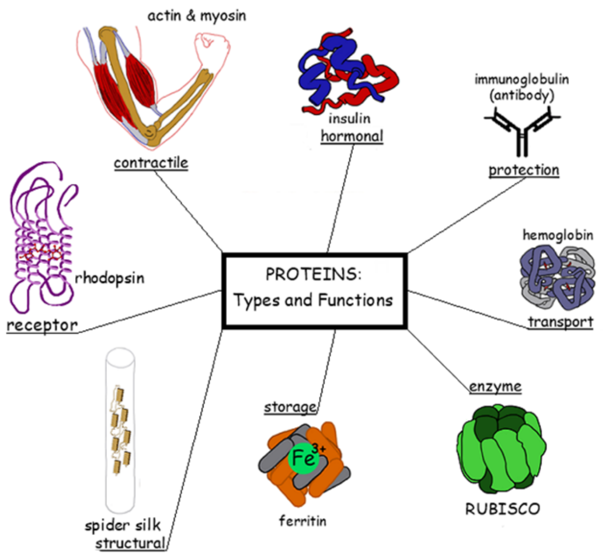


Figure: a) Structure of protein, grey carbon atoms are alpha carbon. R1 R2 R3 are side chains with charge, yellow lines are peptide bond. b) Protein structure representation c) Binding of iron molecule on a protein depends on structure and positions for side chains.

Protein Generation Problem formulation

Proteins are the building block of all living organism. They perform diverse functions like structural backbone, fighting disease, catalyzing reaction etc. Modern characterization techniques along with recently emerged sophisticated data analyzing techniques brought a revolution in the field. For this huge amount of structurally cured data is available in databases



line NCBI, RCSB, Brenda etc.

Figure: Functions of Proteins

Protein is produced in cells by a series of complex mechanism. Generally speaking, ribosome produces the single stranded template called RNA from DNA inside chromosome. From this RNA Protein is produced. The different nature of produced proteins defines the diversification between and beyond organisms. So it can be said that, the type of proteins that are produced depends solely on DNA.

Once proteins are formed, they take 3D structure. This 3D structure depends on the charges and different interplaying forces on the amino acid residues. There are different regions in the structure that are active by which proteins perform certain task.

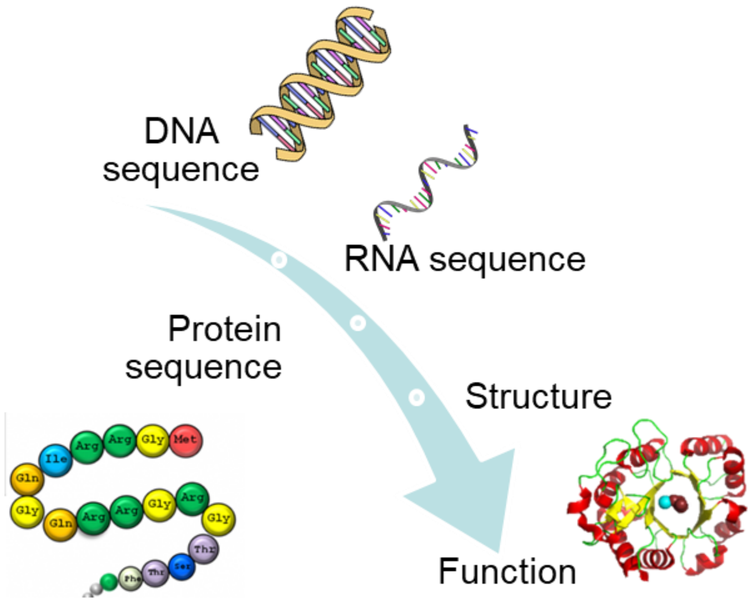


Figure: Protein generation pipeline and structure formation tailored to certain function

By harnessing the power of machine learning it might be possible to produce proteins that did not exist in nature. In this way we are no more limited to natures design and can generate proteins suited to our needs in future. For this it is essential to check whether Generative machine learning algorithms are capable of capturing the 'grammar' in the existing sequence and can generate viable novel sequence.

1.2 Machine Learning Tasks

Recently there has been a very rapid development in the field of Natural Language Processing in Artificial Intelligence. One the common NLP algorithms work with classifying texts; the most common example includes classifying movie reviews in IMDB dataset. These NLP algorithms revolutionized the protein classification problem, where the sequence of proteins is input as text. After training from a large pool of labelled data the model learns to classify proteins solely based on sequence data.

For sequential text classification problems most widely used algorithms are RNN or recurrent neural networks. Convolution neural nets are also used alongside RNN to boost up the efficiency. The sequence or string type data are converted to 'one-hot encoding' or array of binary numbers before entering the network. To utilize the neighboring effect of the amino acids 'k-mers' conversion is also a very fruitful approach where words are converted into k set of words.

As they field of protein is moving fast forward, it is getting extremely important to get to know how new proteins will behave and also if it is always possible to generate novel sequence which will behave like a real protein.

1.3 Dataset Used

In this work we are using the Kaggle 'Structural Protein Sequence' dataset. The dataset consists of one hundred forty thousand labelled protein sequence data. Then the data should be divided into two categories, - train set and test set. Train set is also divided into two sets – model building set and validation set.

1.4 Target Machine Learning Models

'Scikit Learn' is an excellent tool for separating and processing the data. It is also used for non-neural network models like decision trees, clustering or regressions. For neural network based models two machine learning libraries were formerly used – Tensorflow and Keras. In this report we are using the latest TensorFlow library.

First wel attempted only word embedding with sequential model. Next we brought several reformations on the model in the following order -

1. Convolution layer addition CNN
2. Long short term memory (embedding + LSTM) layer
3. LSTM + Dense layer
4. LSTM + CNN, CNN + LSTM

After using those mentioned structure or approach it turned out that number 4 gave us the expected result where the data did not get overfitted. After we found the best two models from classification, we used it in our GAN networks as discriminator (embedded one) and generator (number 4). Lastly, because of the recent reputation of GPT-2 and its huge success in natural language processing, we used an off the shelve pretrained GPT-2 in the form of transfer learning. By using this GAN and GPT-2, we believe that we generated novel sequences. However, it is a matter of lab testing to prove our claim which is a part of our future work.

1.5 Expected Outcomes

In the protein classification problem, our goal is to train a neural network so that it can classify proteins based on the sequence data only. In the Kaggle dataset competition highest accuracy obtained was 60% using convolutional neural net. From the different combination of the models (described in previous section), and tuning hyperparameters we expect to get validation set accuracy greater than 70%.

In case of the generated sequence, though we believe we generated novel sequence, however, there is not any way to justify our results without any lab testing and like it is said before it is a part of our future work.

2. Related Work

2.1 Literature Review

There are dedicated search tools called Blast already available for the purpose of protein classification based on sequence similarity. [1] But these techniques become inadequate as the search space is increasing day by day for huge amount of data produced by Next generation sequencing tools. The reason for attempting machine learning techniques on this dataset is due to recent interest of applying AI in biological sequence data. Specifically, text based natural language processing (NLP) showed great promise in analyzing biological sequence. [2]Google research team recently solved the classification problem with 17929 protein classes (here we used only 10) which outperformed conventional search tools. [3]

Jing and coworkers recently came up with biological sequence classification toolkit called AutobioSeqPy which works by CNN and bi directional LSTM layer. [4] The UDSMProt framework took this one step further by adding Enzyme class prediction, homology detection and fold detection on Swissprot Database. [5] DeepFam is another protein family modelling method.

Very recently Anand and Huang, came up with a idea of generating 3D protein sequence where they used a general adversarial network to perform the work.[6]. This work is the most closely related work to ours though we are focusing on generating sequence(string) instead of a 3D structure.

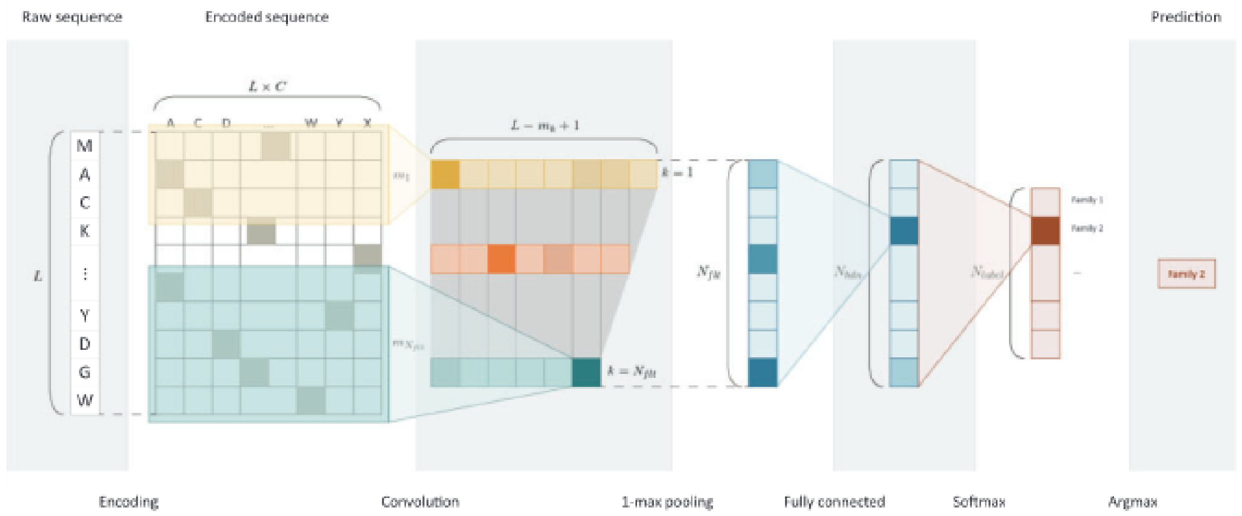
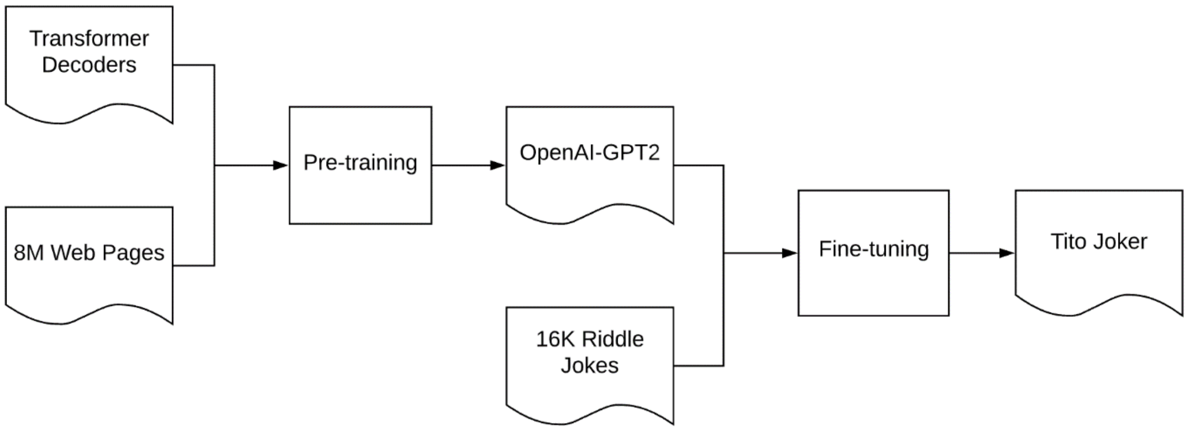


Figure 2: DeepFam model description, the sequence is input as one hot encoding passes through several layers and undergo convolution, max pooling and activation function.

It is of great interest to observe how development in NLP techniques contributes to the bioinformatics field. There are several notebooks in Kaggle where the classification problem was solved with the help of TensorFlow and Keras with CNN layers. In most cases the highest accuracy they get on the validation set is about 60%. Here in this project, we used TensorFlow models and made the models increasingly complex to observe the effect on accuracy. While doing that we found our mentioned (number 4) model works the best. Then when we moved to the generation part, we observed the effect of models on the generated sequences. This will offer us valuable insights on which regions in a protein sequence is most important in determining its class. Afterwards we applied GPT and a generative neural network for producing novel protein sequences. Below is a pictorial example of how a GPT-2 algorithm works for riddle jokes generation.



Generative Adversarial Network

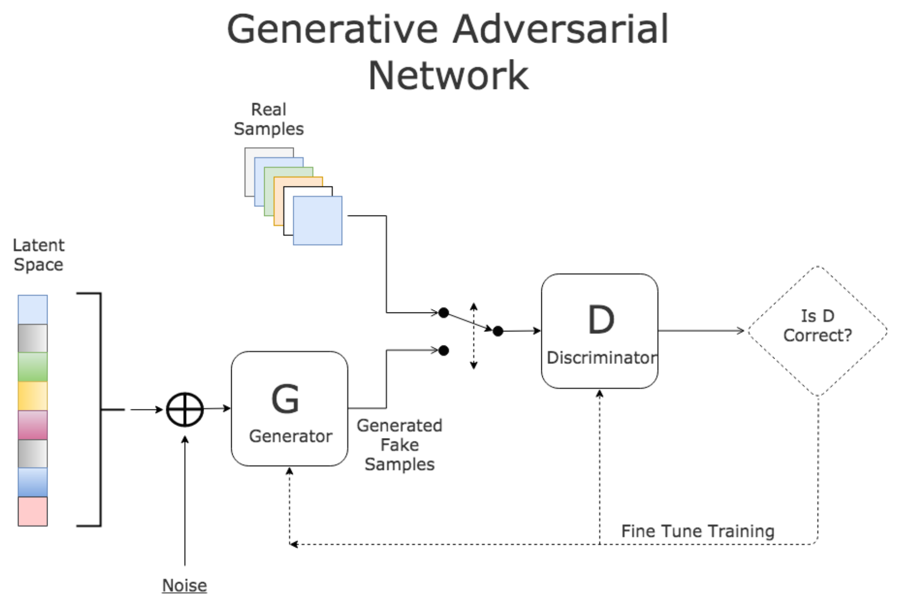


Figure: Generative Adversarial Network Structure
Source - https://cuicaihao.com/2017/05/

Generative adversarial networks are neural networks where two neural networks compete with each other. Here random inputs are generated from a Latent space and passed to the generator network. The generator output is sent to the discriminator network where actual sample are input. The discriminator tries to differentiate between the two. Based on this the loss term is generated which is sent back to the generator to tune the weights. Over the iterative learning steps the generator tries to generate such instance that is statistically identical to the actual samples and thereby 'fool' the discriminator. In a well-trained generator the fake instances produced are statistically very similar to the real instances.

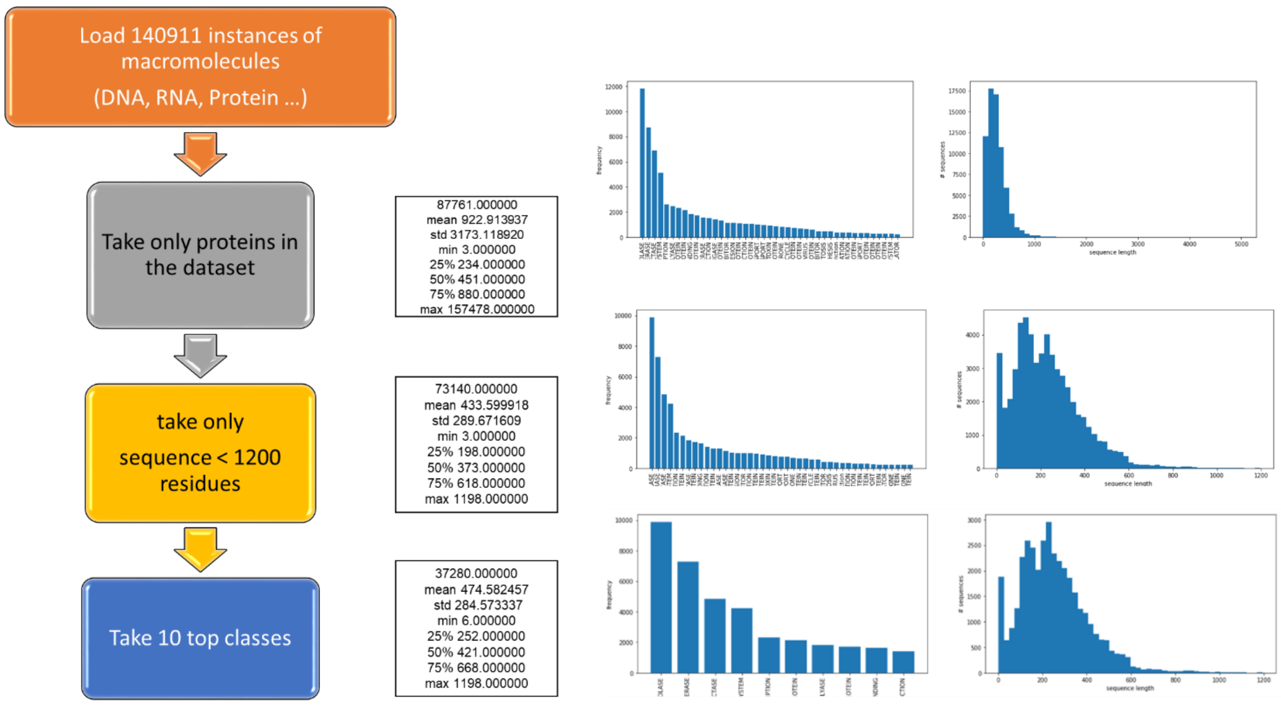
In this paper random latent factor z are sampled and sent to the generator to produce distance maps G(z). The actual protein structure distance maps are input to the discriminator to produce identical fake distance maps. From identical distance maps missing part of the proteins are inpainted.

3. Method Name

The dataset is directly downloaded from Kaggle in CSV format, they are joined with Pandas to produce the final annotated dataset.

3.1 Data Preprocessing

Firstly, we trim the dataset to extract the only protein sequences. The first step in building a model is data preprocessing. The raw data are input and their properties are observed with the help of 'Pandas' statistical visualization tools. The most important criteria in classification tasks is even distribution of the data classes. If the data properties are skewed, some data are dropped or resampled to get a well distributed dataset.

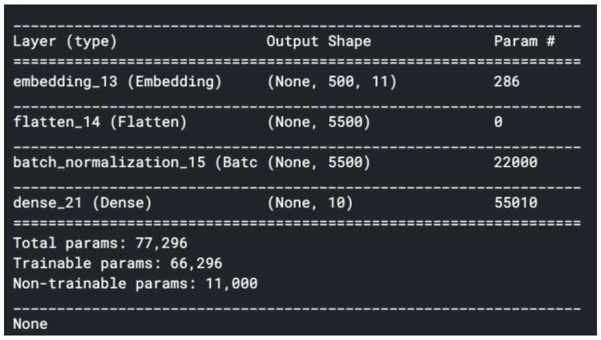


```
87761.000000
mean 922.913937
std 3173.118920
min 3.000000
25% 234.000000
50% 451.000000
75% 880.000000
max 157478.000000
```

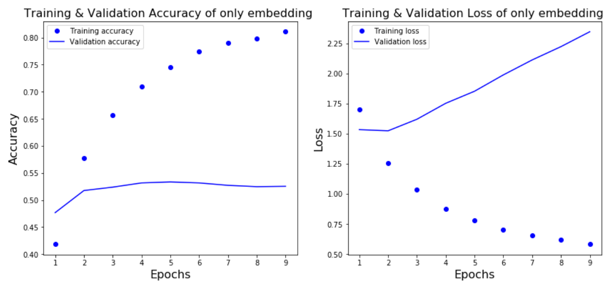
```
73140.000000
mean 433.599918
std 289.671609
min 3.000000
25% 198.000000
50% 373.000000
75% 618.000000
max 1198.000000
```

```
37280.000000
mean 474.582457
std 284.573337
min 6.000000
25% 252.000000
50% 421.000000
75% 668.000000
max 1198.000000
```

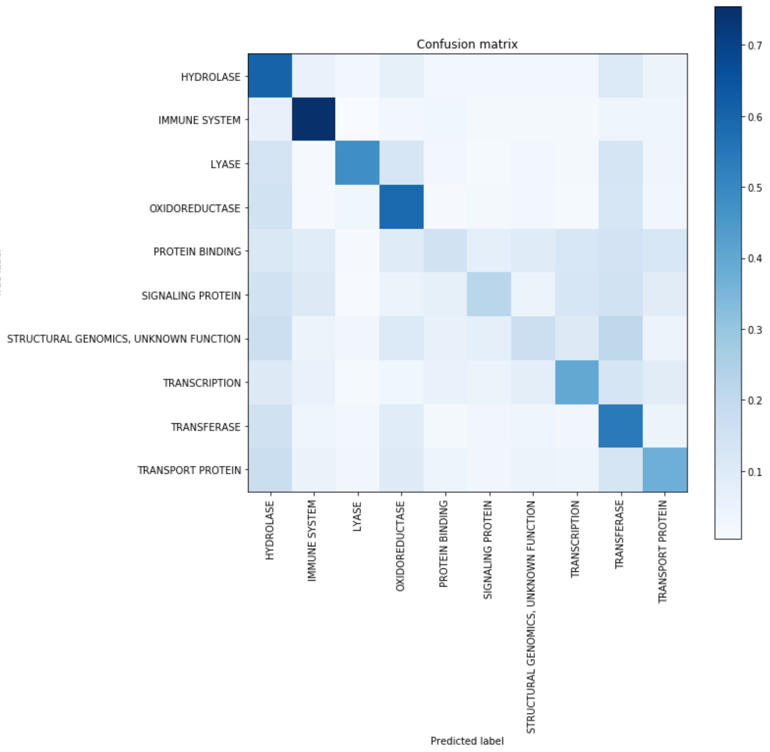
4. Results



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_13 (Embedding)     (None, 500, 11)           286
_________________________________________________________________
flatten_14 (Flatten)         (None, 5500)              0
_________________________________________________________________
batch_normalization_15 (Batc (None, 5500)              22000
_________________________________________________________________
dense_21 (Dense)             (None, 10)                55010
=================================================================
Total params: 77,296
Trainable params: 66,296
Non-trainable params: 11,000
_________________________________________________________________
None
```



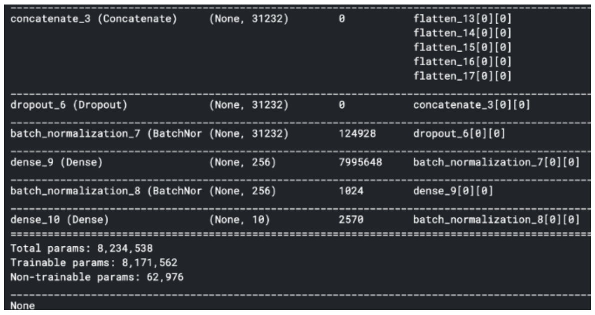
4.1 Embedding



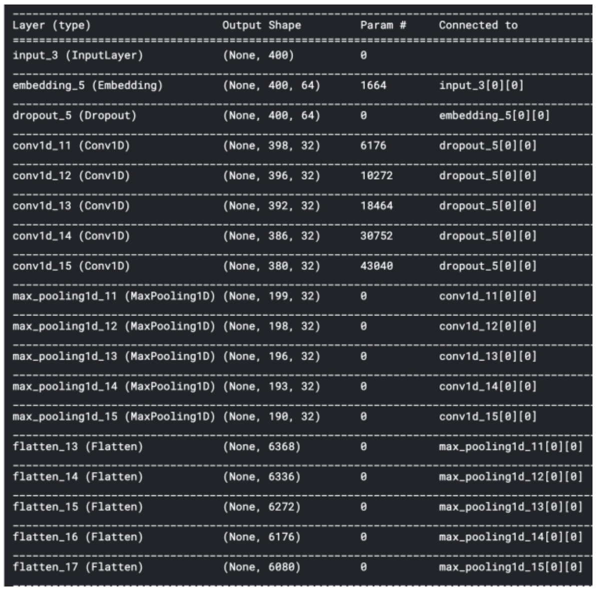
Model structure	Training accuracy	Testing accuracy
word embedding with sequential model	81.5%	51.8%
CNN	~91%	~60%
CNN+LSTM	10.2%	10.3%
LSTM+Embedding	73.1%	70.3%
LSTM+Embedding+dense	37.5%	36.9%

Table 1: Result of classification

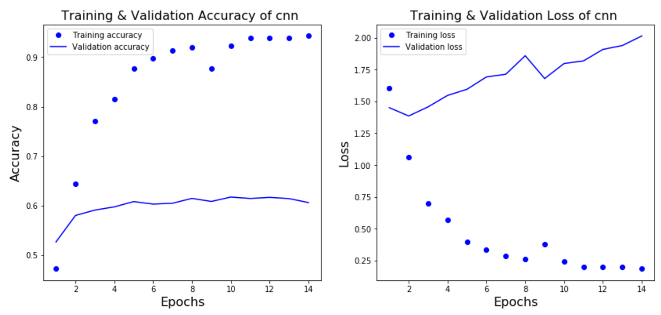
4.2 CNN

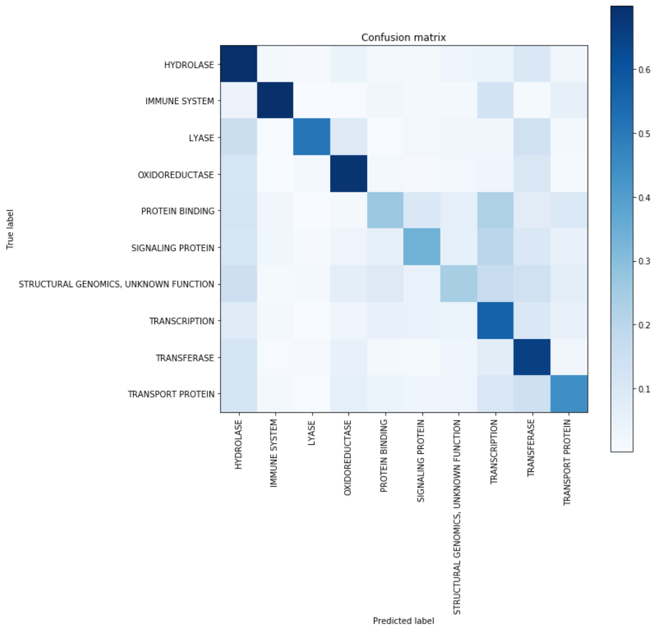


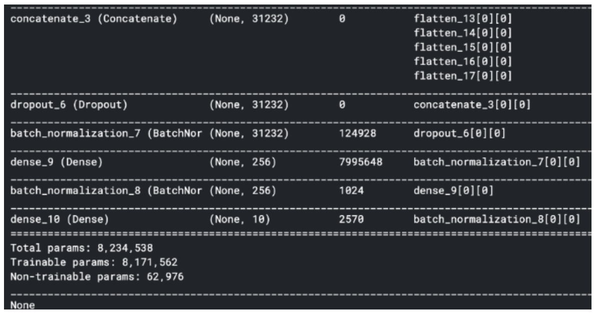
```
--------------------------------------------------------------------------------------------------
concatenate_3 (Concatenate)     (None, 31232)        0           flatten_13[0][0]
                                                                 flatten_14[0][0]
                                                                 flatten_15[0][0]
                                                                 flatten_16[0][0]
                                                                 flatten_17[0][0]
--------------------------------------------------------------------------------------------------
dropout_6 (Dropout)             (None, 31232)        0           concatenate_3[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_7 (BatchNor (None, 31232)        124928      dropout_6[0][0]
--------------------------------------------------------------------------------------------------
dense_9 (Dense)                 (None, 256)          7995648     batch_normalization_7[0][0]
--------------------------------------------------------------------------------------------------
batch_normalization_8 (BatchNor (None, 256)          1024        dense_9[0][0]
--------------------------------------------------------------------------------------------------
dense_10 (Dense)                (None, 10)           2570        batch_normalization_8[0][0]
==================================================================================================
Total params: 8,234,538
Trainable params: 8,171,562
Non-trainable params: 62,976
--------------------------------------------------------------------------------------------------
None
```



```
----------------------------------------------------------------------------------------------------
Layer (type)                     Output Shape         Param #     Connected to
====================================================================================================
input_3 (InputLayer)             (None, 400)          0
----------------------------------------------------------------------------------------------------
embedding_5 (Embedding)          (None, 400, 64)      1664        input_3[0][0]
----------------------------------------------------------------------------------------------------
dropout_5 (Dropout)              (None, 400, 64)      0           embedding_5[0][0]
----------------------------------------------------------------------------------------------------
conv1d_11 (Conv1D)               (None, 398, 32)      6176        dropout_5[0][0]
----------------------------------------------------------------------------------------------------
conv1d_12 (Conv1D)               (None, 396, 32)      10272       dropout_5[0][0]
----------------------------------------------------------------------------------------------------
conv1d_13 (Conv1D)               (None, 392, 32)      18464       dropout_5[0][0]
----------------------------------------------------------------------------------------------------
conv1d_14 (Conv1D)               (None, 386, 32)      30752       dropout_5[0][0]
----------------------------------------------------------------------------------------------------
conv1d_15 (Conv1D)               (None, 380, 32)      43040       dropout_5[0][0]
----------------------------------------------------------------------------------------------------
max_pooling1d_11 (MaxPooling1D) (None, 199, 32)      0           conv1d_11[0][0]
----------------------------------------------------------------------------------------------------
max_pooling1d_12 (MaxPooling1D) (None, 198, 32)      0           conv1d_12[0][0]
----------------------------------------------------------------------------------------------------
max_pooling1d_13 (MaxPooling1D) (None, 196, 32)      0           conv1d_13[0][0]
----------------------------------------------------------------------------------------------------
max_pooling1d_14 (MaxPooling1D) (None, 193, 32)      0           conv1d_14[0][0]
----------------------------------------------------------------------------------------------------
max_pooling1d_15 (MaxPooling1D) (None, 190, 32)      0           conv1d_15[0][0]
----------------------------------------------------------------------------------------------------
flatten_13 (Flatten)             (None, 6368)         0           max_pooling1d_11[0][0]
----------------------------------------------------------------------------------------------------
flatten_14 (Flatten)             (None, 6336)         0           max_pooling1d_12[0][0]
----------------------------------------------------------------------------------------------------
flatten_15 (Flatten)             (None, 6272)         0           max_pooling1d_13[0][0]
----------------------------------------------------------------------------------------------------
flatten_16 (Flatten)             (None, 6176)         0           max_pooling1d_14[0][0]
----------------------------------------------------------------------------------------------------
flatten_17 (Flatten)             (None, 6080)         0           max_pooling1d_15[0][0]
----------------------------------------------------------------------------------------------------
```

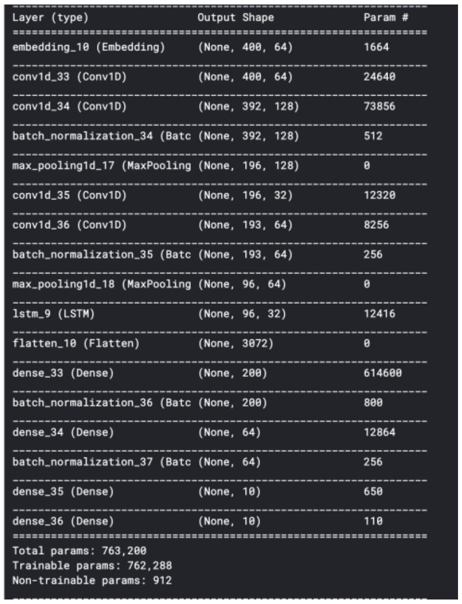




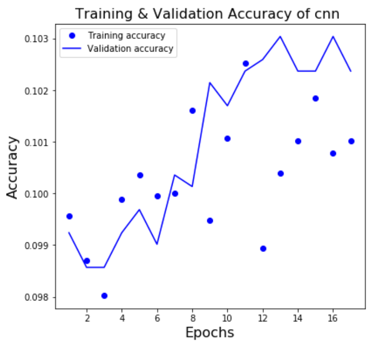


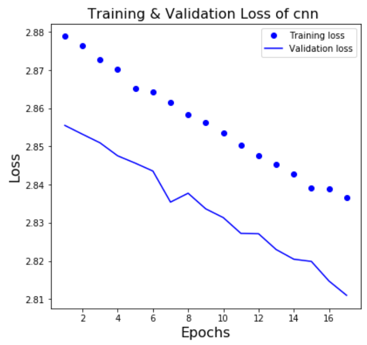
```
----------------------------------------------------------------------------------------------------
concatenate_3 (Concatenate)      (None, 31232)        0           flatten_13[0][0]
                                                                  flatten_14[0][0]
                                                                  flatten_15[0][0]
                                                                  flatten_16[0][0]
                                                                  flatten_17[0][0]
----------------------------------------------------------------------------------------------------
dropout_6 (Dropout)              (None, 31232)        0           concatenate_3[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_7 (BatchNor (None, 31232)        124928      dropout_6[0][0]
----------------------------------------------------------------------------------------------------
dense_9 (Dense)                  (None, 256)          7995648     batch_normalization_7[0][0]
----------------------------------------------------------------------------------------------------
batch_normalization_8 (BatchNor (None, 256)          1024        dense_9[0][0]
----------------------------------------------------------------------------------------------------
dense_10 (Dense)                 (None, 10)           2570        batch_normalization_8[0][0]
====================================================================================================
Total params: 8,234,538
Trainable params: 8,171,562
Non-trainable params: 62,976
----------------------------------------------------------------------------------------------------
None
```

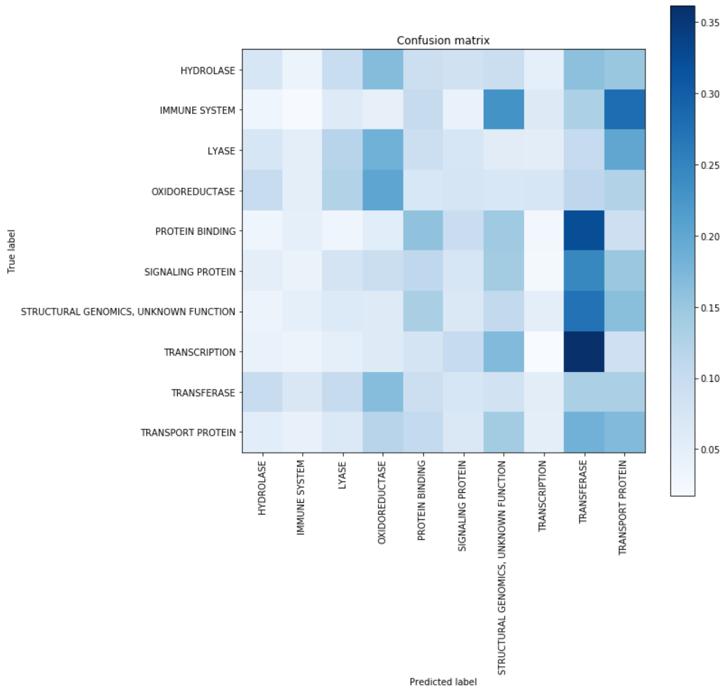
4.3 CNN + LSTM



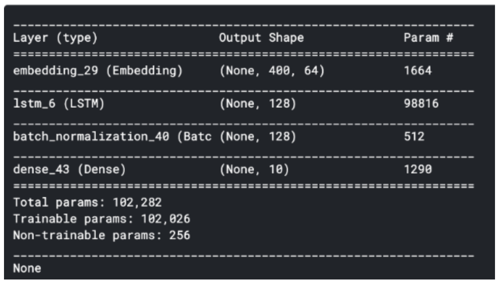
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_10 (Embedding)     (None, 400, 64)           1664
_________________________________________________________________
conv1d_33 (Conv1D)           (None, 400, 64)           24640
_________________________________________________________________
conv1d_34 (Conv1D)           (None, 392, 128)          73856
_________________________________________________________________
batch_normalization_34 (Batc (None, 392, 128)          512
_________________________________________________________________
max_pooling1d_17 (MaxPooling (None, 196, 128)          0
_________________________________________________________________
conv1d_35 (Conv1D)           (None, 196, 32)           12320
_________________________________________________________________
conv1d_36 (Conv1D)           (None, 193, 64)           8256
_________________________________________________________________
batch_normalization_35 (Batc (None, 193, 64)           256
_________________________________________________________________
max_pooling1d_18 (MaxPooling (None, 96, 64)            0
_________________________________________________________________
lstm_9 (LSTM)                (None, 96, 32)            12416
_________________________________________________________________
flatten_10 (Flatten)         (None, 3072)              0
_________________________________________________________________
dense_33 (Dense)             (None, 200)               614600
_________________________________________________________________
batch_normalization_36 (Batc (None, 200)               800
_________________________________________________________________
dense_34 (Dense)             (None, 64)                12864
_________________________________________________________________
batch_normalization_37 (Batc (None, 64)                256
_________________________________________________________________
dense_35 (Dense)             (None, 10)                650
_________________________________________________________________
dense_36 (Dense)             (None, 10)                110
=================================================================
Total params: 763,200
Trainable params: 762,288
Non-trainable params: 912
_________________________________________________________________
```



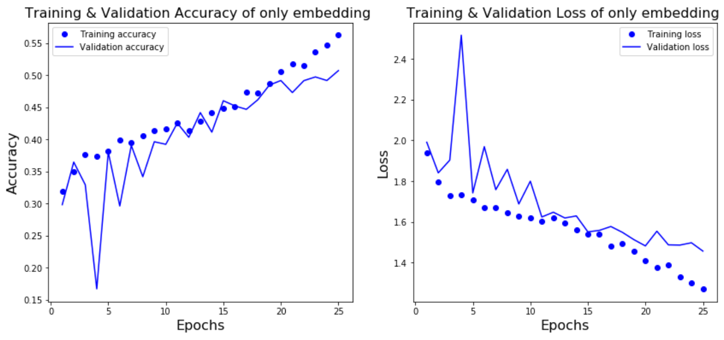


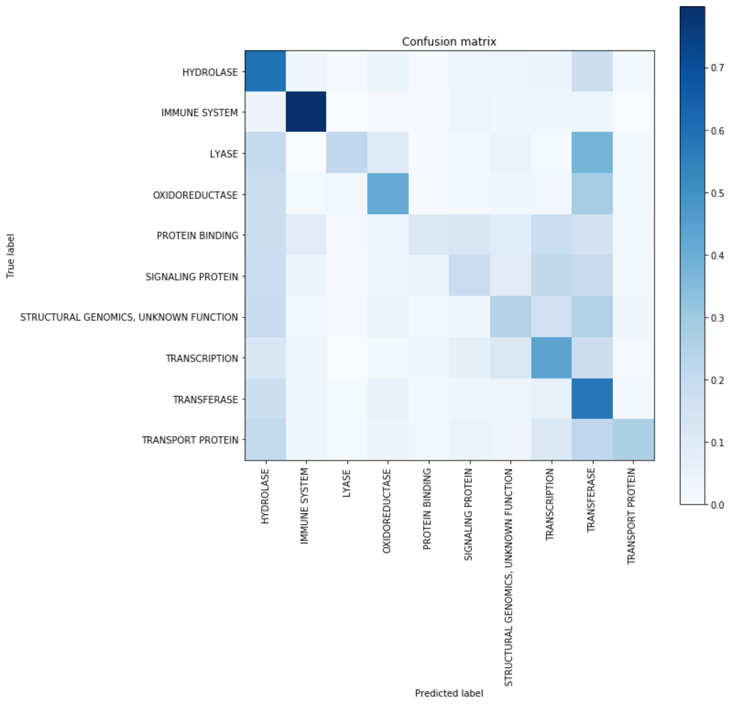


4.4 LSTM + embedding

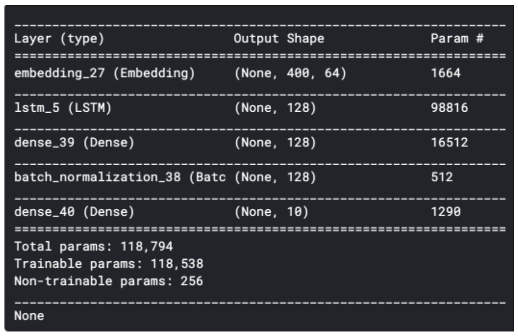


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_29 (Embedding)     (None, 400, 64)           1664
_________________________________________________________________
lstm_6 (LSTM)                (None, 128)               98816
_________________________________________________________________
batch_normalization_40 (Batc (None, 128)               512
_________________________________________________________________
dense_43 (Dense)             (None, 10)                1290
=================================================================
Total params: 102,282
Trainable params: 102,026
Non-trainable params: 256
_________________________________________________________________
None
```

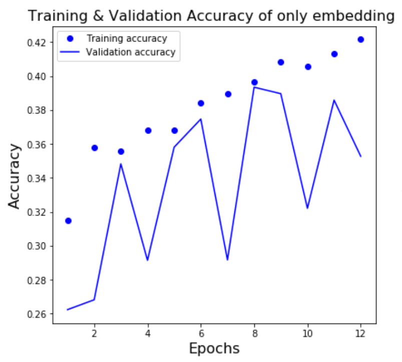


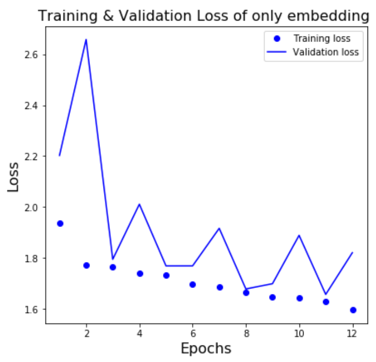


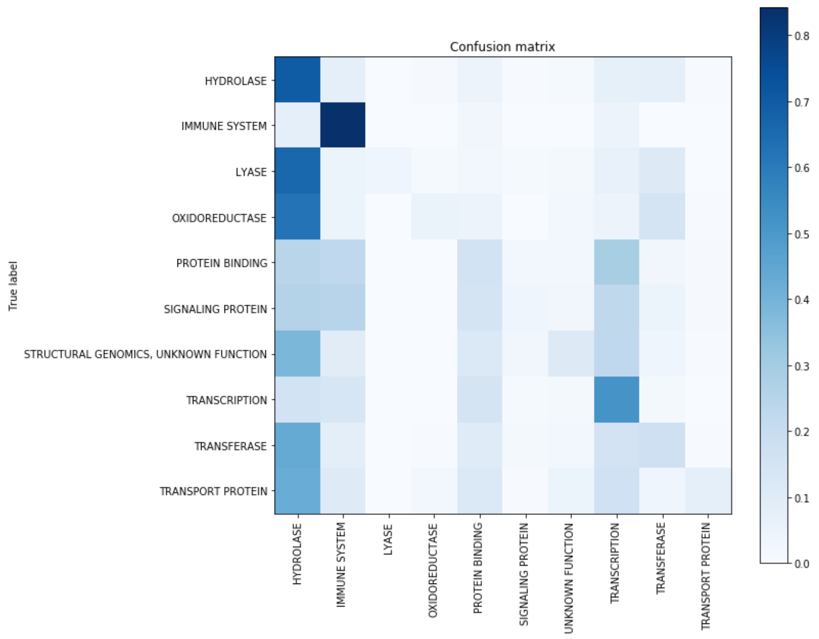
4.5 LSTM + embedding + dense



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_27 (Embedding)     (None, 400, 64)           1664      
_________________________________________________________________
lstm_5 (LSTM)                (None, 128)               98816     
_________________________________________________________________
dense_39 (Dense)             (None, 128)               16512     
_________________________________________________________________
batch_normalization_38 (Batc (None, 128)               512       
_________________________________________________________________
dense_40 (Dense)             (None, 10)                1290      
=================================================================
Total params: 118,794
Trainable params: 118,538
Non-trainable params: 256
_________________________________________________________________
None
```







We put confusion matrix and pictorial view of our experimental models above to have an idea what we did. and In section, we the classification result will be described. We first followed the traditional way of sequence classification which is embedding. We got the good training accuracy result though the testing accuracy was on the lower side. This indicated that our model got overfitted and too long sequence can be one of the reasons for this as the model might have hard time to generalize. Then we tried a simple 2 layered CNN where we get a very good training accuracy though the testing is poor as before. We assumed the same reason for this model to as we did not go deeper for this and also the CNN had hard time to have an idea about the relation among the sequence.Then, we tried different variation and mixture of CNN and LSTM and from there we found that LSTM+embedding did a really good job. The idea behind this was that as we know sequence has a relation and they do follow some certain patterns, LSTM learned that when we passed that using embedding. In addition to, the batch normalization helped us to reduce the overfitting. Thus, from this experiment we found that LSTM+embedding is the best among all. [Table 1]

In this portion the experimental setup and some observation will be explained. The experimental setup has been given in Table 2 while the training process has been given in Table 3. These two tables depict the whole experimental setups and how we moved forward with the process. Batch size and the embedding layer size came from the empirical analysis as at the beginning it was uncertain which size will work best. To find out the hyperparameters we did a exhaustive grid search and found those mentioned sizes for batch and embedding layer respectively. The step size for GPT-2 was more like a trial-and-error process where it turned out to be that 10000 is the steps where we can get stable tuning process. Here, by stable, we mean that we can get a certain portion of a protein from the database. We used this intuition from the fact that a generated sequence cannot be totally new. In table 3, the epochs size and learning rate has been given. These ones were chosen intuitively rather than doing any exhaustive search or trial and error.

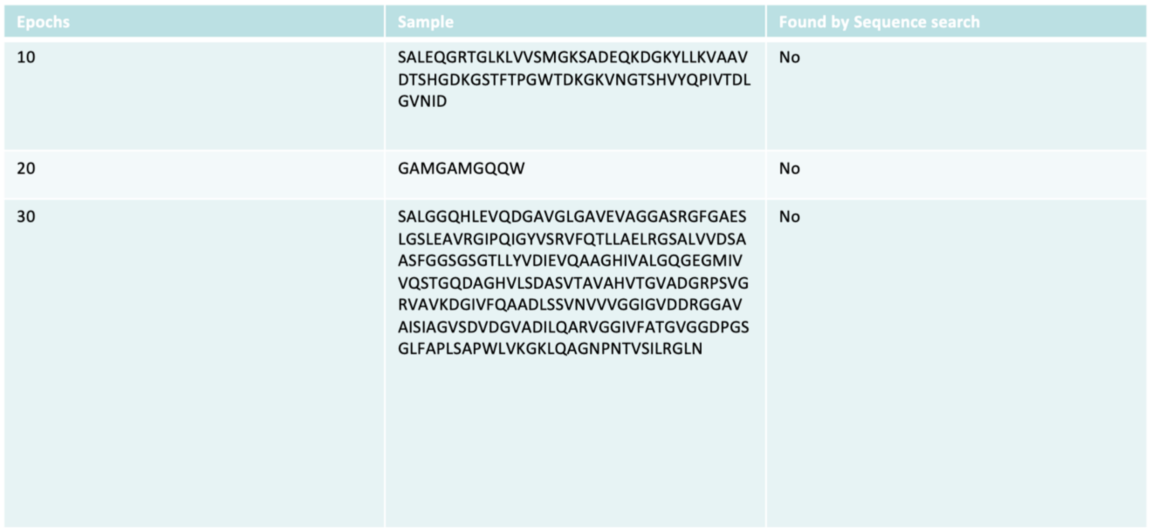
Batch size	200
Embedding layer size	300
Steps for GPT-2	10000

Table 2: experimental setup

Number of epochs	30
Learning rate	0.0001
Hardware	GPU
Training time	5 hours(GAN), 6 hours(GPT-2)

Table 3: experimental setup

Our GAN networks were nothing but built with our two best model we found from our classification portion which has already been given in the classification part. In case of lose function, we used the standard lose function from Sequence GAN [7] as this particular GAN is closer to our work than others to best of our knowledge. Now, moving to GPT-2, all the standard parameters has been used apart from the mentioned as it is a pretrained model and we just used it as a feature extractor. Some generated sequences has been given in the Table 4 and Table 5 after every 10 epochs with the indicator about if that particular sequence has been found in the protein database or not. In terms of validating if those generated sequences are valid or not is a matter of lab testing which is a part of our future work.



Epochs	Sample	Found by Sequence search
10	SALEQGRTGLKLVVSMGKSADEQKDGKYLLKVAAV DTSHGDKGSTFTPGWTDKGKVNGTSHVYQPIVTDL GVNID	No
20	GAMGAMGQQW	No
30	SALGGQHLEVQDGAVGLGAVEVAGGASRGFGAES LGSLEAVRGIPQIGYVSRVFQTLLAELRGSALVVDSA ASFGGSGSGTLLYVDIEVQAAGHIVALGQGEGMIV VQSTGQDAGHVLSDASVTAVAHVTGVADGRPSVG RVAVKDGIVFQAADLSSVNVVVGGIGVDDRGGAV AISIAGVSDVDGVADILQARVGGIVFATGVGGDPGS GLFAPLSAPWLVKGKLQAGNPNTVSILRGLN	No

Table 4: experimental result of GAN

Epochs	Sample	Found in sequence search
10	XATLSSQSLRPLHPLQDTSDLLTPLPPLCIVSSDP APNLTQRPNTPWVTPWNGSSTLSSDTPFLTDP GQEQRTGKGSSRSTNAPASLATGTNYPSTGAPV FSAVASNIPEIGGNRQRVSDDTSGNTGRLLLHG ALRQKLGGSLYFVSPSTNAPATGTNYPSTVMSR SPTSRATTGGQVITVAAGASPQDTSNAAASANP SFVSDQYVPGMA	No
20	ATQTAYLLLARLLGGAAARGYGEGLGVFADSAA RSLVVADTAAGAADAGSGGADAGAVGAAAGG APAAARAGTAD	No
30	ATSTKKLHKEPATLIKAIDGDTVKLMYKGQPMT FRLLLVDTPEFNEKYGPEASAFTKKMVENAKKIE VEFDKGQRTDKYGRGLRYIYADGKMVNEALVR QGLAKVAYVYKGNNTHEQLLRKAEAQAKKEKL NIWSEDNADSGQ	No

Table 5: experimental result of GPT-2

In this paragraph, few interesting observations will be explained. Interestingly, both GAN and GPT adopted to the size of the hydrolase (a specific protein) differently. GAN started with a large sequence and drastically moved to a very short sequence while GPT-2 behave the same way, but the short sequence was not as short as GAN in terms of size. Another interesting observation is that GAN generated larger sequence than GPT-2 after 30 epochs. It is most likely because of the adversarial effect GAN has and also the 30% case where our classification model failed to classify (as we are using them as a generator and discriminator of our GAN). As intuitive solution, we tried to match partial portion of the sequence with the sequences from the database which turned out to be the right one. That is why we believe our generated sequences to be a novel one though we need to test in the lab to come to the final conclusion.

5. Future Plan

After the experiments and going through literatures we found few relative options to explore. The use of k-mers or bag of words technique to modify the sequence is one of them. Another thing to look at is to compare the novel proteins with the test set to see their match with the specified type with local alignment score. They will be also searched through protein data bases to see their match with specific types. The sequence generated were not found anywhere in the database, so it could be concluded that novel sequence was generated as said before. But if we search shorter sequences, then they appear in the databases but the name is not the targeted protein. That means the model could capture certain pattern of the production. Next onwards there are few more modifications that we can attempt. Such as –

1. Using conditional GAN
2. Attempting simple auto-encoder decoder
3. Using n grams of word.

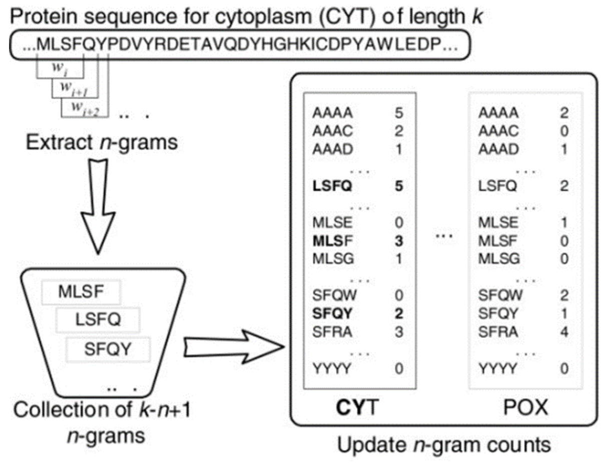


Figure: Conditional GAN

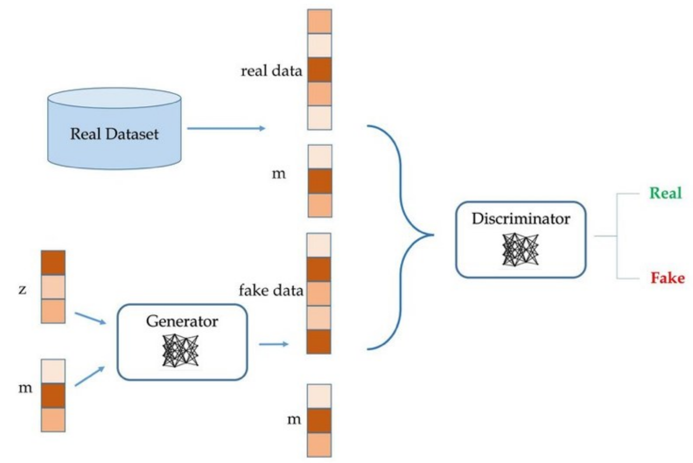


Figure: N gram of word embedding

References

[1] Z. Wu, “Data-Driven Protein Engineering Thesis by,” 2021.

[2] M. Tsubaki, K. Tomii, and J. Sese, “Compound-protein interaction prediction with end-to-end learning of neural networks for graphs and sequences,” *Bioinformatics*, vol. 35, no. 2, pp. 309–318, 2019, doi: 10.1093/bioinformatics/bty535.

[3] M. L. Bileschi *et al.*, “Using Deep Learning to Annotate the Protein Universe,” *bioRxiv*, pp. 1–29, 2019, doi: 10.1101/626507.

[4] R. Jing, Y. Li, L. Xue, F. Liu, M. Li, and J. Luo, “AutoBioSeqpy: A Deep Learning Tool for the Classification of Biological Sequences,” *J. Chem. Inf. Model.*, vol. 60, no. 8, pp. 3755–3764, 2020, doi: 10.1021/acs.jcim.0c00409.

[5] N. Strodthoff, P. Wagner, M. Wenzel, and W. Samek, “UDSMProt: Universal deep sequence models for protein classification,” *Bioinformatics*, vol. 36, no. 8, pp. 2401–2409, 2020, doi: 10.1093/bioinformatics/btaa003.

[6] N. Anand, P. Huang, "Generative modeling for protein structures", NIPS,2018.

[7]. (Yu et al., 2017)Yu, L., Zhang, W., Wang, J., & Yu, Y. (2017). SeqGAN: Sequence generative adversarial nets with policy gradient. 31st AAAI Conference on Artificial Intelligence, AAAI 2017, 2852–2858.