

# Deep Learning - Project 2

**Submitted by**

Fatema Tabassum Liza

&

Arunima Mandal

In this assignment, recurrent neural networks have been applied in the given protein sequences as instructed.

## **Task I – Recurrent Neural Network Design**

In this task we have implemented LSTM as the Recurrent Neural Network.

The network we established is described by the Equations (10.40) to (10.44) (from the textbook).

During the data preprocessing steps, we divide the input sequences in 2 sets- comprising a training set and a validation set. We considered 2000 samples of protein sequences considering the lack of computational resources to process the entire dataset. In addition, to simplify the model, we set a maximum length of 100 and truncate the ones that are longer and pad the ones which are shorter.

In this specific task we chose LSTM because LSTM networks have been shown to learn long-term dependencies more easily than the simple recurrent architectures. LSTM overcame a drawback that RNN imposes. LSTM networks are well suited to classifying and making predictions based on time series data. It is developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs.

## Network Architecture

Here is a brief description of the network we have established in this task.

Layer (type)	Output Shape	Param #
embedding_13 (Embedding)	(None, 99, 24)	504
lstm_25 (LSTM)	(None, 99, 150)	105000
dropout_13 (Dropout)	(None, 99, 150)	0
lstm_26 (LSTM)	(None, 100)	100400
dense_13 (Dense)	(None, 21)	2121
Total params: 208,025		
Trainable params: 208,025		
Non-trainable params: 0		
None		

Figure: Network Overview

## Task II – Language Models for Protein Sequences and Evaluation

For this specific task we have used ‘Adadelata’ as optimizer and ‘categorical\_crossentropy’ as the loss function. We used a ‘softmax’ activation function and the metric used is ‘accuracy’. For weight initialization we have used random-initializer. Because overtime, with random initializers, the model tends to predict a good estimation which leads to a better metric and evaluation. We have trained the above mentioned LSTM using the given dataset and have plotted the loss, accuracy on the training set and validation set with respect to the epochs. The plot for the corresponding loss and accuracy is shown below:

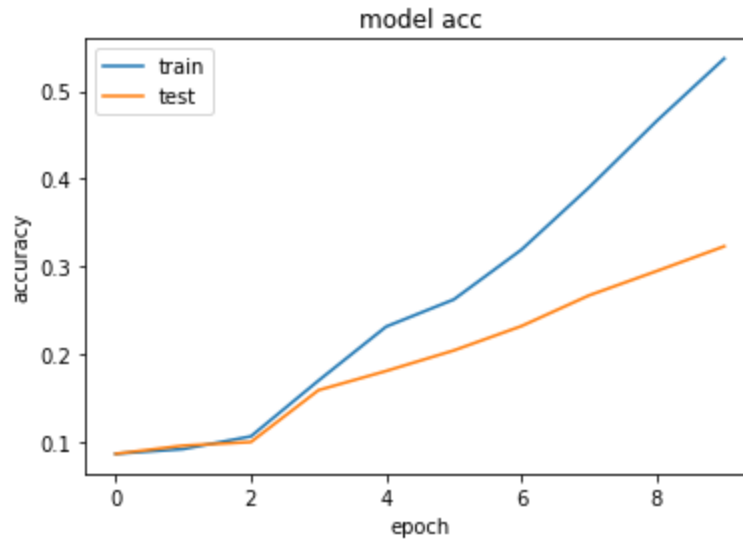


Figure: model accuracy

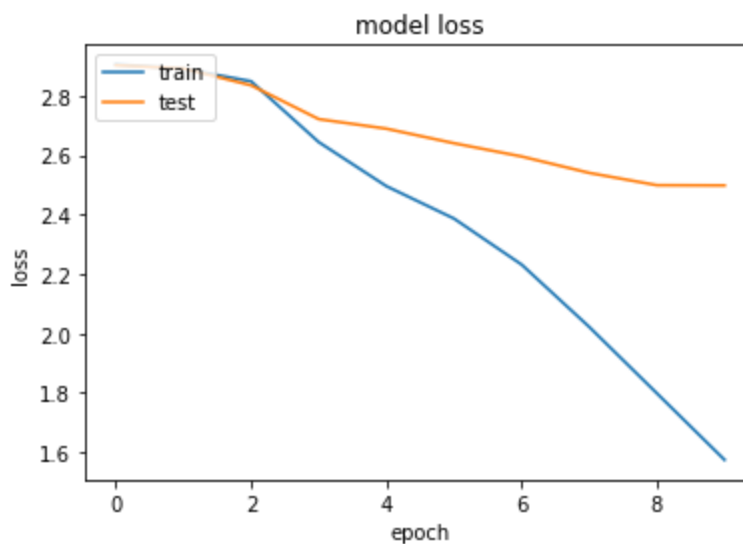


Figure: model loss

One of the advantages of recurrent neural networks using LSTM is that it can capture the long-term dependencies. For this specific problem, we tried different lengths of sequence and then changed one or more letters to identify the longest dependencies the network is capable of capturing experimentally. The result of this experiment can be described below.

#### **Test with no modifications in input:**

Sub sequence of first six letters: VMLSEG

Validation sample: VMLSEGEWQLVLHVWAKVEADMAVGDIL

Predicted sample: VMLSEGEwqlvlhvwakveadmavgdil

As we can see, the model was able to predict the next 22 letters correctly with the original input.

#### **Test with single character change:**

In this case, we changed the last letter in sub sequence: VMLSEW

Validation sample: VMLSEGEWQLVLHVWAKVEADMAVGDIL

Predicted sample: VMLSEWewqlvlhvwakveadmavgdil

As we can see, changing the last letter did not really affect the model since the model was still able to predict the next 22 letters correctly.

#### **Test with two characters change:**

In this case, we changed the last 2-letters.

So the subsequence becomes VMLS

Validation sample: VMLSEGEWQLVLHVWAKVEADMAVGDIL

Predicted sample: VMLSEWewqkknvmawkvedmaavgdlg

As we can see change in the last two letters has affected a lot and the model was able to predict only the next two letters correctly. This could be because of a small number of characters in the input.

#### **Test with larger input (11 characters) with change in two characters in input:**

Actual data: VMLSEGEWQLVLVHWAK

With larger input: VMLSEGEWQLVLHLWAK

Predicted output: VMLSEGEWQLVLVLWAKveadvageggdlilierllrhshetklklgti

Validation sample: VMLSEGEWQLVLHVWAKVEADVAGHGGQDILRILFDRTLE


The model was able to predict the first eleven sequences correctly. However, a few letters were wrongly predicted later. This shows the dependency of the two characters changed in input data over the predicted amino acids.

## **Task III – Sequence Generation Techniques**


### **Part 1:**

In this task, **at first** we were asked to use our trained RNN to complete protein sequences and generate new protein sequences. We have built a function called **generate\_text()** which predicts a particular sequence given the prefix sequence. Our generated output for 5 different predicted sequences example (k=2,3,4,5,6) are given below:


```
[26] print( generate_text("VWAKVEADVAGHGQDILIRLFKSHPET", 2, max_sequence_len))
```

 VWAKVEADVAGHGQDILIRLFKSHPET v l


```
[27] print( generate_text("VWAKVEADVAGHGQDILIRLFKSHPET", 3, max_sequence_len))
```

 VWAKVEADVAGHGQDILIRLFKSHPET v l e


```
[28] print( generate_text("VWAKVEADVAGHGQDILIRLFKSHPET", 4, max_sequence_len))
```

 VWAKVEADVAGHGQDILIRLFKSHPET v l e l

```
[29] print( generate_text("VWAKVEADVAGHGQDILIRLFKSHPET", 5, max_sequence_len))
```

 VWAKVEADVAGHGQDILIRLFKSHPET v l e l e

```
[30] print( generate_text("VWAKVEADVAGHGQDILIRLFKSHPET", 6, max_sequence_len))
```

 VWAKVEADVAGHGQDILIRLFKSHPET v l e l e r

**Secondly**, we were asked to populate the given table. As an example

1. For  $k=5$ , we fix the first five amino acids in each of the sequences in the validation set
2. We generate a sequence as long as the original sequence
3. Then we compare the two sequences, and find the longest matches right after the initial ones.
4. We store the found longest match's occurrence (frequency) in the given column(0-19+).
5. Thus after traversing all the training samples, each row will eventually contain the frequency of each length(0-19+) matched for all the training samples.

The table we got after running the code:

k	Number of sequences with the given maximal matches																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	24	2	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	23
2	17	2	2	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	28
3	8	6	3	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	32
4	13	3	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	32
5	9	2	0	0	0	2	0	0	0	0	1	0	0	2	0	0	0	0	0	35
6	7	1	0	0	2	0	0	0	0	1	0	0	2	0	0	0	0	0	1	37
7	5	1	0	2	0	0	0	0	1	0	0	2	0	0	0	0	0	1	0	39
8	4	1	3	0	0	0	0	1	0	0	2	0	0	0	0	0	1	0	1	38
9	5	3	0	0	0	0	1	0	0	2	0	0	0	0	0	1	0	1	0	38
10	7	1	0	0	0	1	0	0	2	0	0	0	0	0	1	0	1	0	0	38

From the above table, we can see that the maximal letters that are predicted correctly is - **the length of 19 or more (19+) where k=7.**

## Part 2:

**Firstly**, we were asked to use our network to generate a 4-gram language model, i.e., estimating the probability for all the possible protein sequences of length 4.

We used n-gram models to derive a probability of the sentence ,W, as the joint probability of each individual word in the sentence,  $w_i$ .

$$P(W) = P(w_1, w_2, \dots, w_n)$$

We then reduced to a sequence of n-grams using the **Chain Rule of conditional probability.**

$$P(x_1, x_2, \dots, x_n) = P(x_1)P(x_2|x_1)...P(x_n|x_1,...x_{n-1})$$

For example, let's predict the probability of the sequence 'ABCD'

$$\begin{aligned} P('ABCD') &= P('A', 'B', 'C', 'D') \\ &= P('A')P('B'|'A')P('C'|'BC')P('D'|'ABC') \end{aligned}$$

So here is a glimpse of the output after running the expected code which outputs all the unique 4-gram sequences trained on our RNN model and their probabilities:

```
gysa -> 2.5538989485391197e-07
rlvi -> 1.5933306418826758e-06
arag -> 7.5681035987024e-07
ragq -> 6.084938109490165e-08
agqd -> 5.331204481307665e-08
gqdd -> 3.423765025641288e-09
qdde -> 2.50268820583384e-08
ddev -> 3.131529901193998e-06
rilm -> 3.2143871316450215e-07
ilma -> 1.3809426042125128e-09
lman -> 2.0871859317932382e-08
anga -> 7.967418756279507e-08
gapf -> 2.716683308562885e-09
apft -> 9.937849499798271e-08
tt dw -> 3.673629407442364e-07
tdwl -> 9.303035031760234e-08
wlgt -> 9.712733634796473e-10
lgts -> 2.417545033240608e-06
tspl -> 1.8462657488683483e-05
plhl -> 8.573975533745168e-06
hlaa -> 1.6342520127881489e-06
```

**Secondly**, we also estimate a 4-gram model from the training set by counting the 4-amino acids. And we find out their probability by calculating the frequency of each unique 4-gram sequence and then divided that by the total sum of frequencies so that the resulting probabilities fall legally between 0 and 1. This is some modified version of **MLE (Maximum Likelihood Estimation)** - an intuitive probability calculation method.

So here is a glimpse of the output after running the expected code which outputs all the unique 4-gram sequences from our training set and their probabilities:

```
PEYL -> 2.725594009144368e-05
EYLR -> 2.725594009144368e-05
YLRD -> 3.40699251143046e-05
DSEA -> 2.725594009144368e-05
SEAN -> 2.725594009144368e-05
EANP -> 2.725594009144368e-05
NPVD -> 2.725594009144368e-05
VDQP -> 2.725594009144368e-05
DQPT -> 2.725594009144368e-05
QPTE -> 2.725594009144368e-05
PDVA -> 2.725594009144368e-05
VAAC -> 2.725594009144368e-05
ACRF -> 2.725594009144368e-05
TVSW -> 2.725594009144368e-05
```