Berkay ALTINTAŞ

22002709

EEE 443 NEURAL NETWORKS MINI PROJECT REPORT

QUESTION 1

In this question, it is asked to design and implement an autoencoder with a single hidden layer for unsupervised feature extraction from natural images. To do so, the following cost function which includes average squared error, Tykhonov regularization and KL divergence term respectively. The first term which is average squared error represents the difference between the desired output and the output created by autoencoder. As it can be seen in the Figure 1, the desired output which is reconstructed image is same with the input image. The second term, Tykhonov regularization, tries to use regularization to the weights of the encoder by using the parameter λ. The third term, Kullback-Leibler divergence term, adds the sparsity to the weights by using the parameter ρ. More specifically, the sparsity is arranged by the parameter ρ between Bernoulli parameters with mean ρ and ρ_b which represents the average activation of the hidden unit. The general architecture of the task can be seen in the Figure 1. Moreover, the cost function for this task can be seen Figure 2.

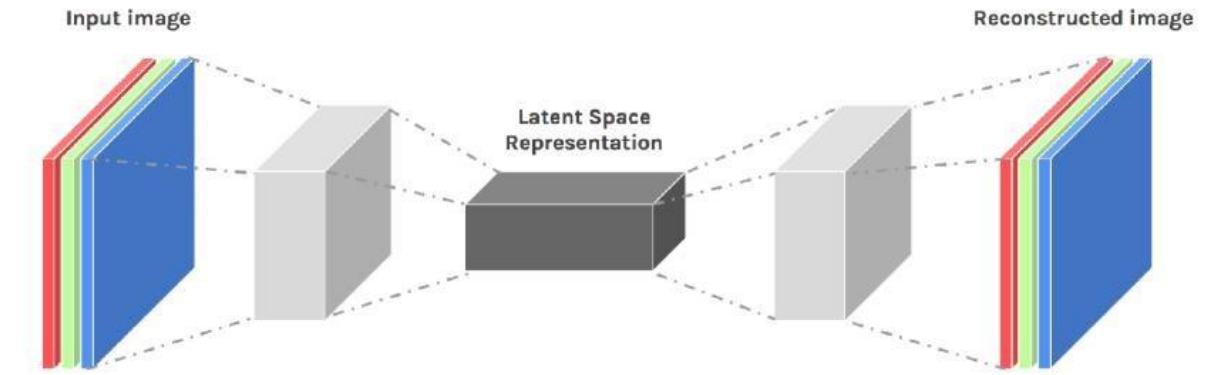

Figure 1: General Architecture of Autoencoders [1]

$$J_{ae} = \frac{1}{2N}\sum_{i=1}^{N}\|d(m) - o(m)\|^2 + \frac{\lambda}{2}\left[\sum_{b=1}^{L_{hid}}\sum_{a=1}^{L_{in}}\left(W_{a,b}^{(1)}\right)^2 + \sum_{c=1}^{L_{out}}\sum_{b=1}^{L_{hid}}\left(W_{b,c}^{(2)}\right)^2\right] + \beta\sum_{b=1}^{L_{hid}} KL(\rho|\hat{\rho}_b) \quad (1)$$

Figure 2: Cost Function

Part A

In this task, the given dataset 'data1.h5' contains 16x16 RBG patches. Before starting any model implementation, we have to do data preprocessing. To do that, the original data we have, is converted to grayscale images with the help of the Luminosity Model. Luminosity model gives different weights for red, green and blue regarding the human perception. More specifically, the applied Luminosity Model is as follows,

Y = 02126 * R + 07152 * G + 00722 * B

After applying the Luminosity Model, dataset is needed to be normalized. For this purpose, the mean pixel intensity of the each image is calculated and subtracted from the related image's itself. Then, the standard deviation of the all pixels in the dataset is calculated. The calculated standard deviation is used to clipped the dataset in between positive and negative 3 standard deviations. Moreover, to eliminate the possible saturations, these 3 standard deviations are mapped to [0.1, 0.9]. The formula to do mapping is as follows.

$$x_{\text{mapped}} = a + (b - a) \cdot \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

where a is lower bound and b is upper round. In our case, a is 0.1 and b is 0.9.

After completing the data preprocessing part, it is asked to plot random 200 RGB images and their normalized forms. These random 200 RGB images and their normalized versions can be seen in below.

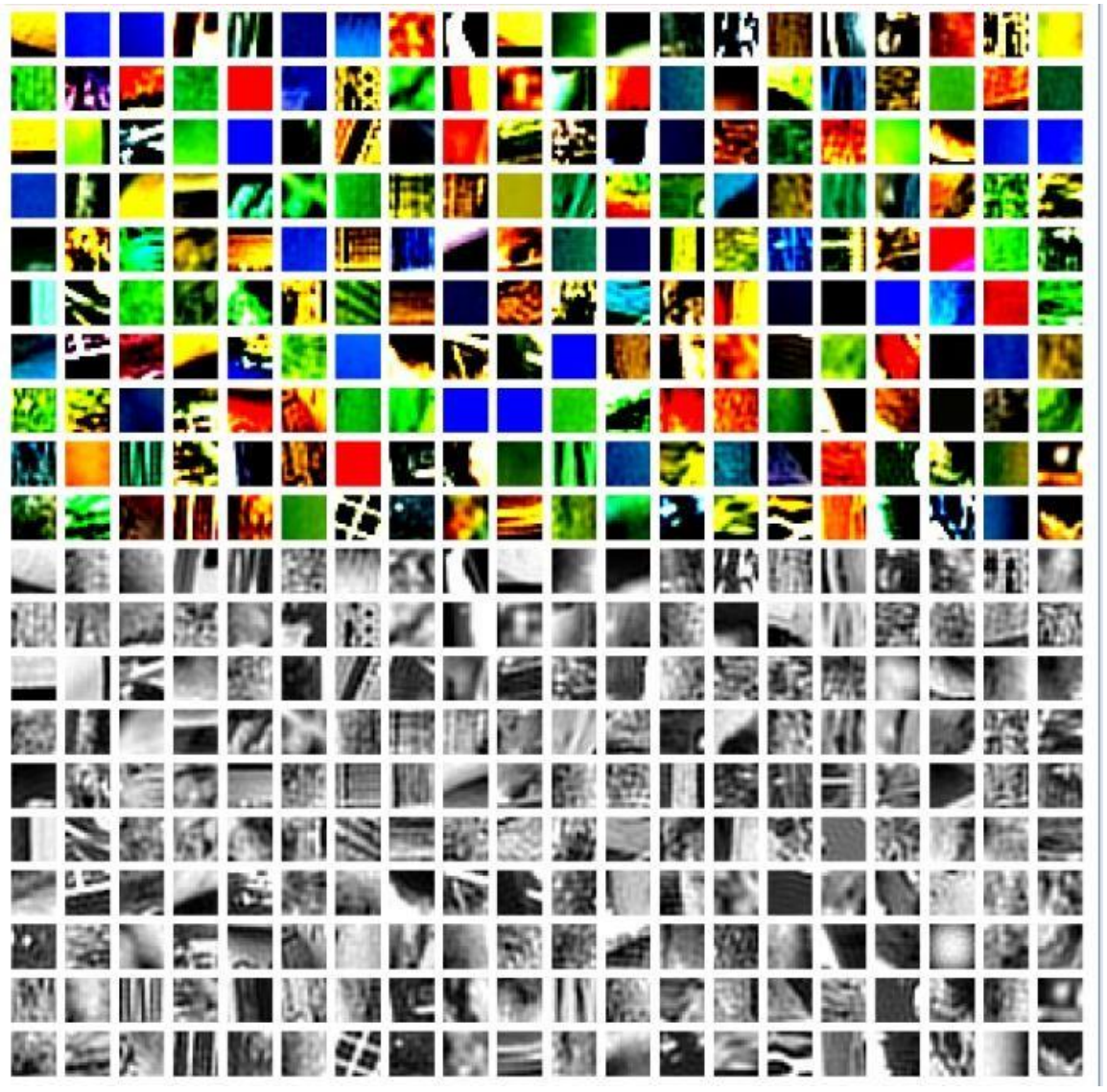

Figure 3: Random 200 RGB Patches and Their Normalized Versions

By looking at the Figure 3, we can see that our RGB data is correctly converted into the normalized and grayscale form. In the grayscale version, we can see clearly the shapes of the lines or curves in RGB data. For example, if we look at the RGB images which is completely red, blue or green, we can see the shapes in it in its normalized and gray scaled version.

Part B

After completing the data preprocessing, now it is time to initialize our learnable parameters which are weights and biases. For this purpose, it is used to specific interval [$-w_o$, w_o] where w_0 is like as follows,

$$w_o = \sqrt{\frac{6}{L_{\text{pre}}+L_{\text{post}}}}$$

where L_{pre} and L_{post} represents the number of neurson before and after the connecting weights respectively. After initializing the weights and biases regarding the given interval, we are ready to apply forward pass, compute the loss function and gradients and then update the weights using the backpropagation.

For the forward pass part, we have an encoder, a single hidden layer and a decoder. Thus, our forward is like as follows sequentially,

$$h = \sigma(X \cdot W_1 + b_1)$$

$$\hat{X} = \sigma(h \cdot W_2 + b_2)$$

The X is our input, W_1 and b_1 is learnable parameters between encoder and hidden. Moreover, $\hat{X}$ is output or reconstruction created by the decoder part. W_2 and b_2 is the learnable parameters between hidden layer and the decoder part. Furthermore, the activation in here is sigmoid.

If we think the given cost function in Figure 2, we have reconstruction loss, regularization terms and sparsity terms as follows respectively.

$$J_{\text{reconstruction}} = \frac{1}{2N}\sum_{i=1}^{N} |\hat{x}^{(i)} - x^{(i)}|^2$$

$$J_{\text{regularization}} = \frac{\lambda}{2}(|W_1|^2 + |W_2|^2)$$

$$J_{\text{sparsity}} = \beta \sum_{j=1}^{L_{\text{hid}}} (\rho \log\frac{\rho}{\hat{\rho}_y} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_y})$$

Thus, in total we have**,**

$$J = J_{\text{reconstruction}} + J_{\text{regularization}} + J_{\text{sparsity}}$$

After completing the forward pass and loss calculations, we can dive into the gradient calculations. The gradient calculations starting from at the end of the network to starting of the network is as follows.

The decoder part error is as follows,

$$\delta_{\text{out}} = (\hat{X} - X) \odot \sigma'(\hat{X})$$

where σ' represents the derivative of the sigmoid activation like as follows

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$$

Moreover, the hidden layer error is like as follows,

$$\delta_{\text{hidden}} = (\delta_{\text{out}} \cdot W_2^{\mathsf{T}} + \beta \cdot (-\frac{\rho}{\hat{\rho}} + \frac{1-\rho}{1-\hat{\rho}})) \odot \sigma'(h)$$

Then the gradients for the decoder part is as follows,

$$\frac{\partial J}{\partial W_2} = \frac{1}{N} h^{\mathsf{T}} \cdot \delta_{\text{out}} + \lambda W_2$$

$$\frac{\partial J}{\partial b_2} = \frac{1}{N} \sum_{i=1}^{N} \delta_{\text{out}}^{(i)}$$

Then the gradients for the hidden layer part is as follows,

$$\frac{\partial J}{\partial W_1} = \frac{1}{N} X^{\mathsf{T}} \cdot \delta_{\text{hidden}} + \lambda W_1$$

$$\frac{\partial J}{\partial b_1} = \frac{1}{N} \sum_{i=1}^{N} \delta_{\text{hidden}}^{(i)}$$

The all workflow for this network is shown above. Then we can move onto the training/testing parts.

In this part, it is asked to use 64 neurons in the hidden layer and λ = 0.0005. Moreover, it is asked to use find the proper β and ρ values. It is important because these parameters affect the performance of network. If the performance is affected, quality of the features exctracted from the network also changes.

Part C

In this part, it is asked to display the first layer of the connection weights as a separate image for each neuron in the hidden layer. In Part B, we have defined the parameters as L_{hid} = 64 and λ = 0.0005. Moreover, the values for β and ρ are chosen as 0.5 and 0.05 respectively. Since we have 64 neurons in the hidden layer, we expect that 64 different learned features in total. The learned features by each neuron and the training error can be seen below

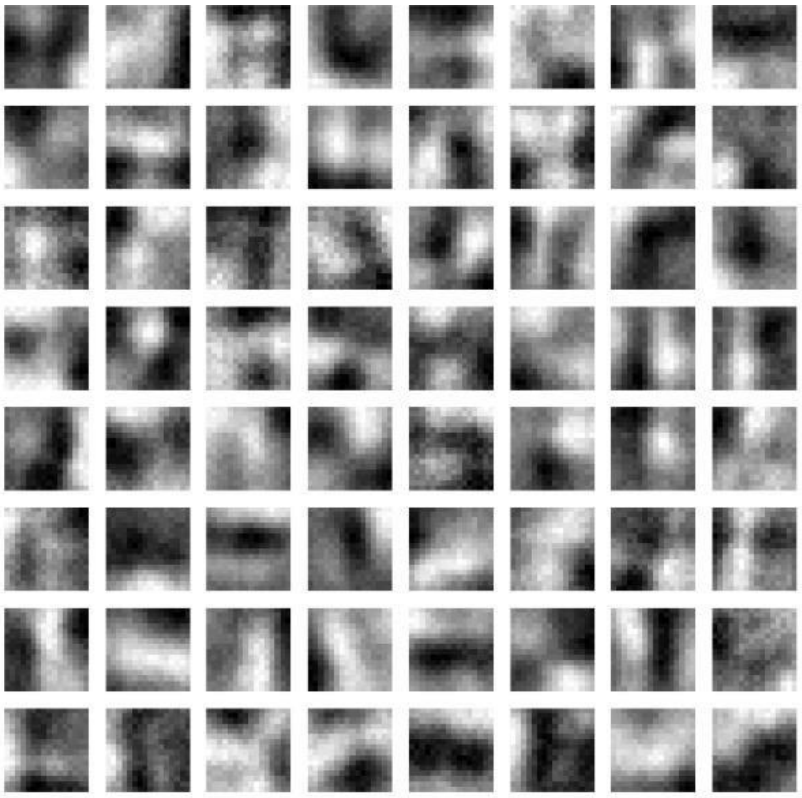

Figure 4: Learned Features when $L_{hid} = 64$ and $\lambda = 0.0005$

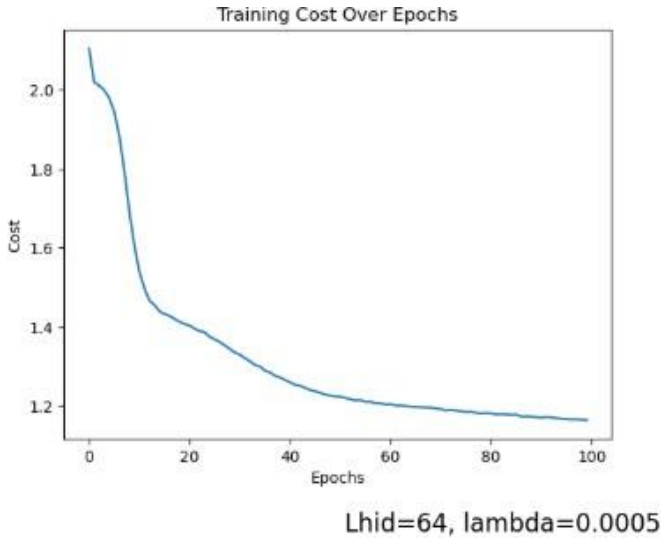

Figure 5: Training Loss Curve when $L_{hid} = 64$ and $\lambda = 0.0005$

As it can be seen in Figure 4, these are not directly the representative of natural images. They look like lines, curves or some kind of details. In other words, these learned features represent the patterns, shapes, lines or curves in the images. These learned features are important because these are used in the decoder part to reconstruct the image again.

Part D

In this part, it is asked to repeat the Part C by changing the number of neurons in the hidden layer and λ values while keeping the β and ρ same. In other words, in this part, it is asked to choose 3 different values which are low, medium and high in the given intervals for the number of neurons in the hidden layer and λ. Specifically, we neuron sizes are chosen as 10, 50 and 100. Moreover, the λ values are chosen as 0.00001, 0.0005, 0.001. All results are shown below.

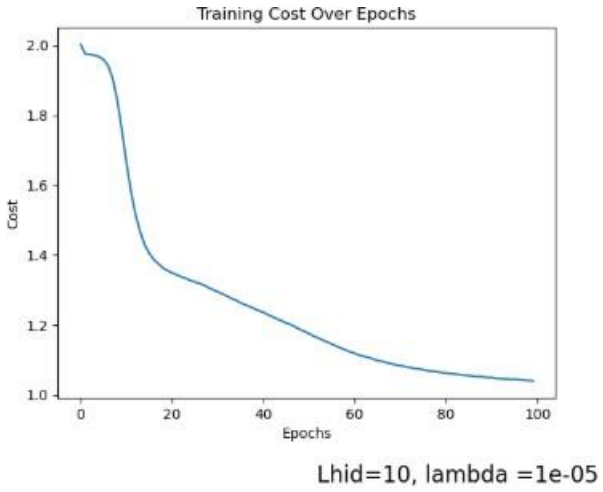

Figure 6: Training Loss Curve when L_{hid} = 10 and λ = 0.0001

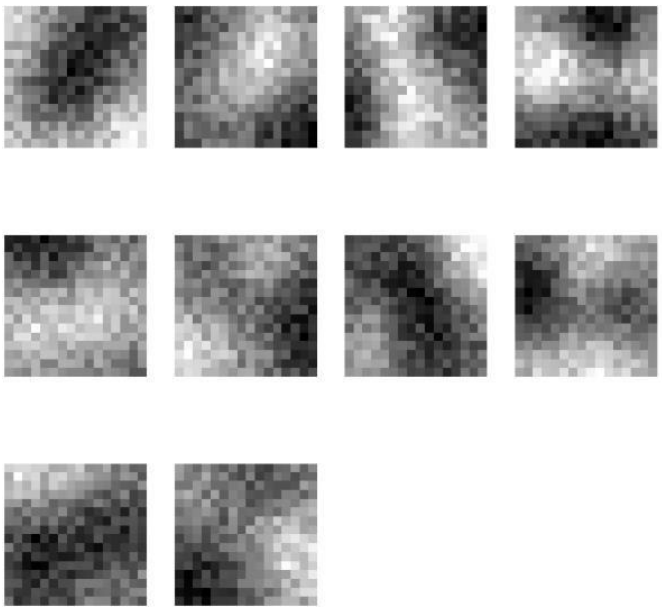

Figure 7: Learned Features when L_{hid} = 10 and λ = 0.00001

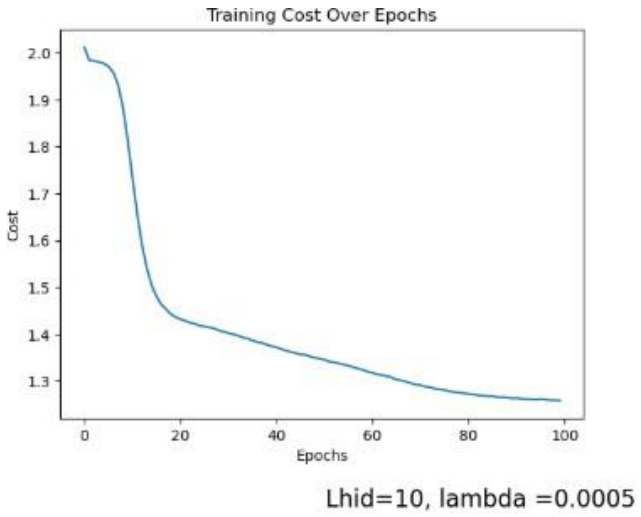

Figure 8: Training Loss Curve when L_{hid} = 10 and λ = 0.0005

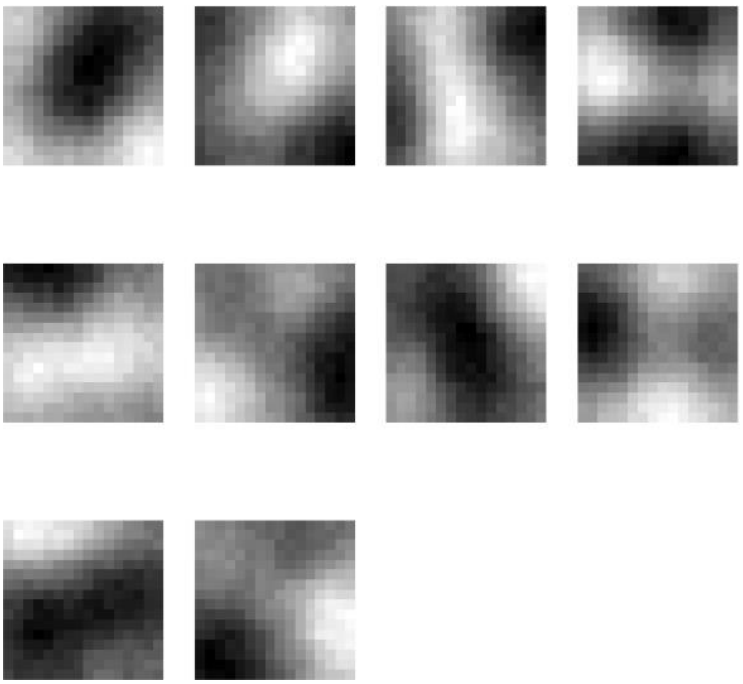

Figure 9: Learned Features when L_{hid} = 10 and λ = 0.0005

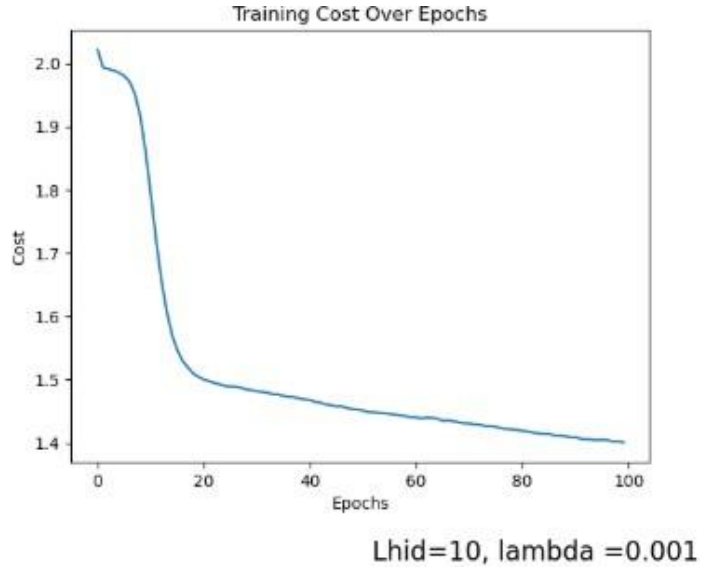

Figure 10: Training Loss Curve when L_{hid} = 10 and λ = 0.001

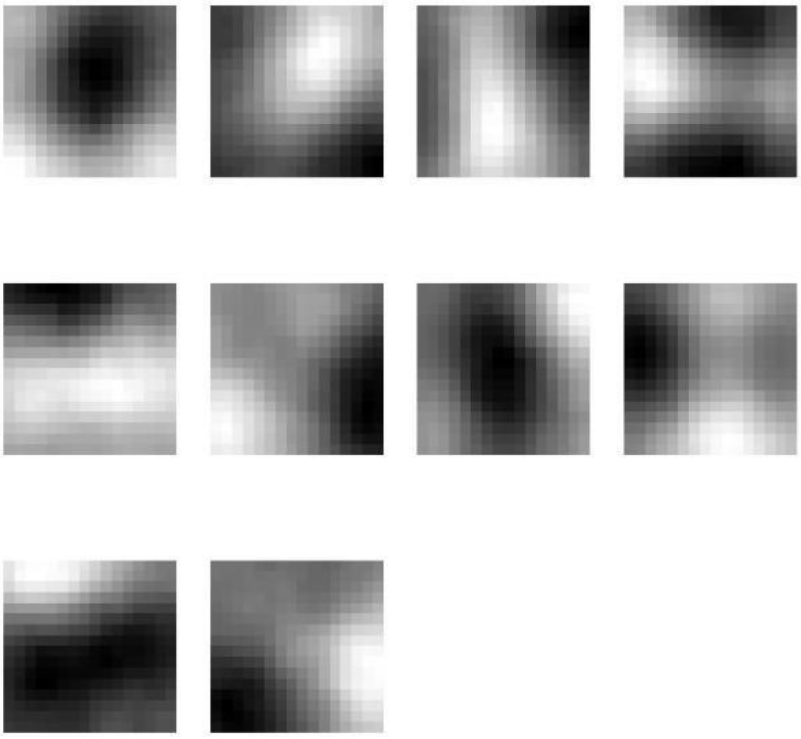

Figure 11: Learned Features when L_{hid} = 10 and λ = 0.001

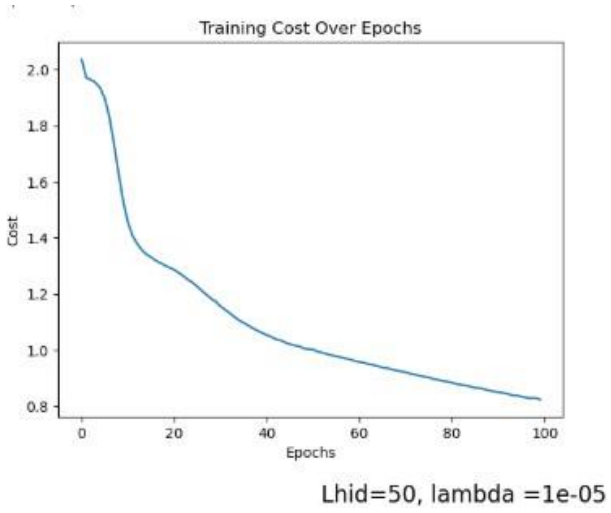

Figure 12: Training Loss Curve when $L_{hid} = 50$ and $\lambda = 0.00001$

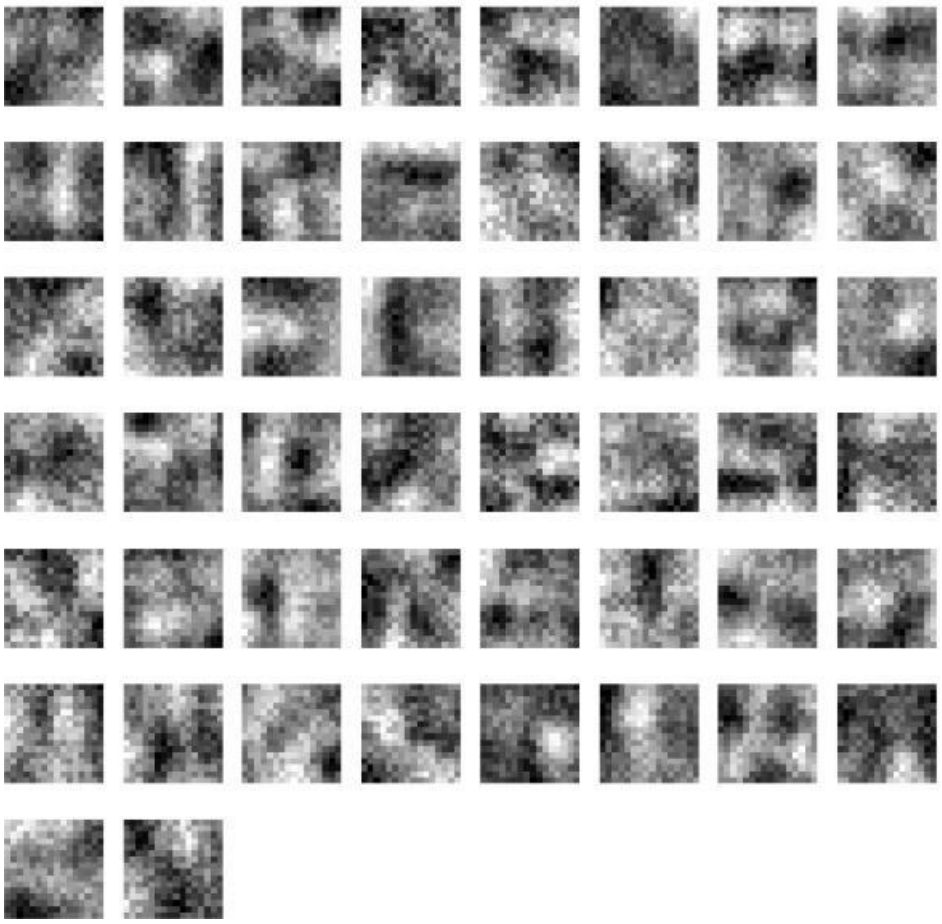

Figure 13: Learned Features when $L_{hid} = 50$ and $\lambda = 0.00001$

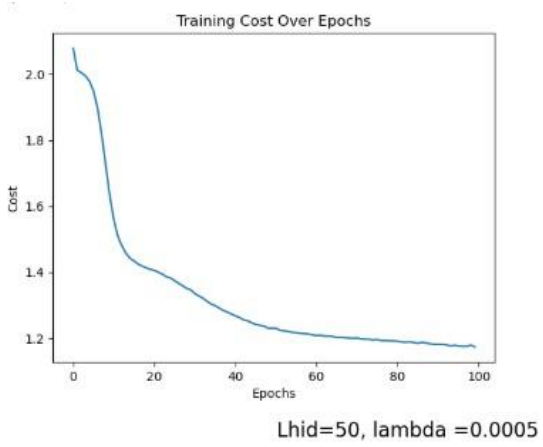

Figure 14: Training Loss Curve when $L_{hid} = 50$ and $\lambda = 0.0005$

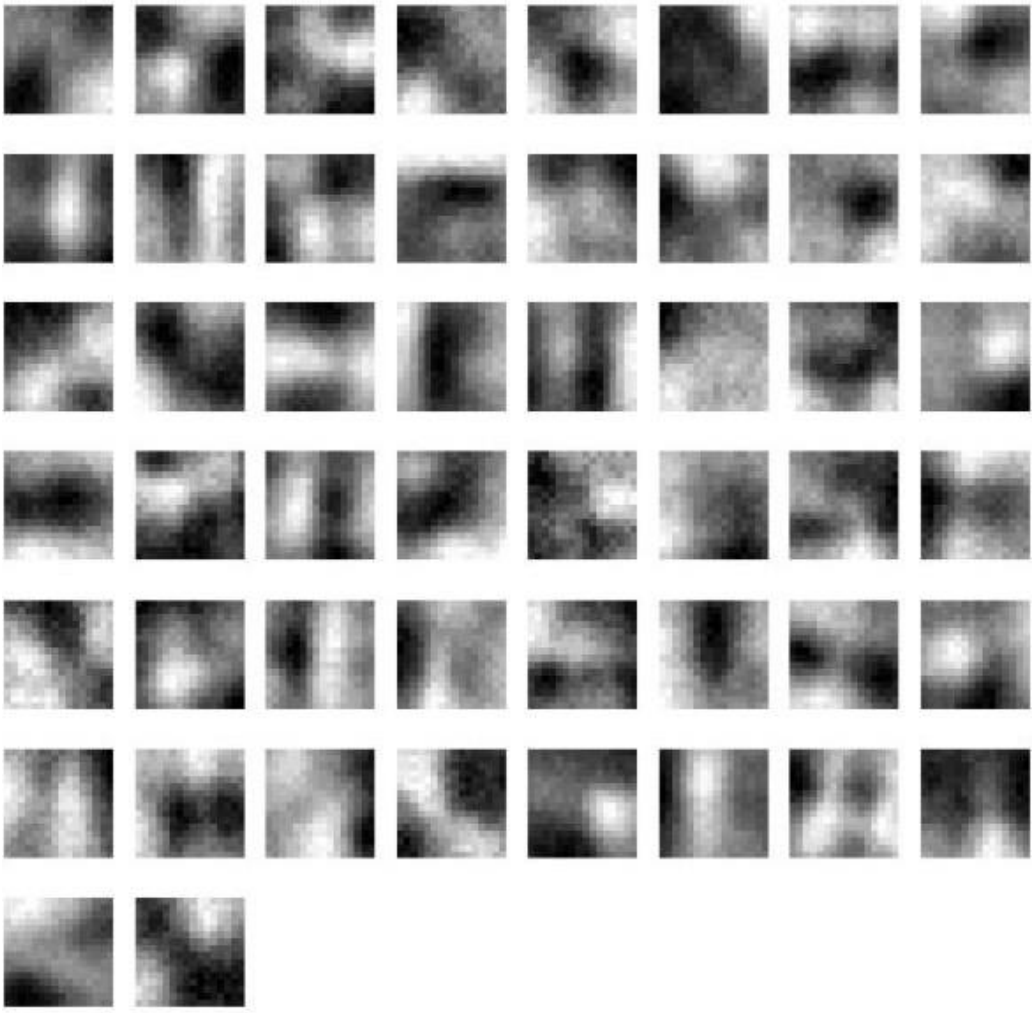

Figure 15: Learned Features when $L_{hid} = 50$ and $\lambda = 0.0005$

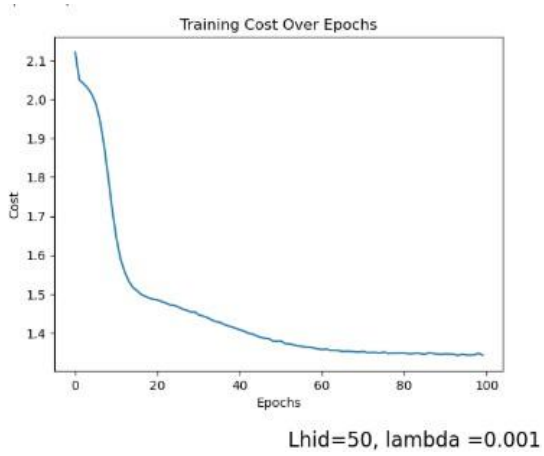

Figure 16: Training Loss Curve when L_{hid} = 50 and λ = 0.001

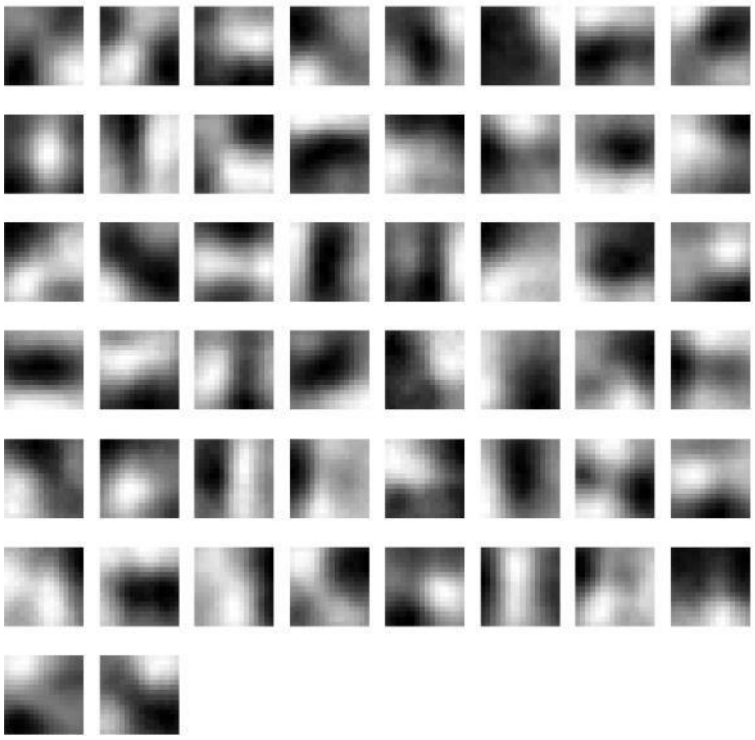

Figure 17: Learned Features when L_{hid} = 50 and λ = 0.001

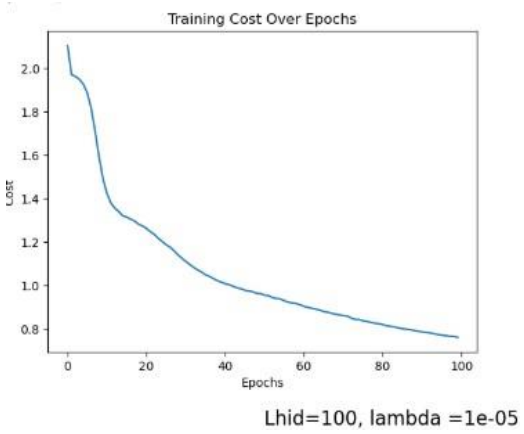

Figure 18: Training Loss Curve when $L_{hid} = 100$ and $\lambda = 0.00001$

Figure 19: Learned Features when $L_{hid} = 100$ and $\lambda = 0.00001$

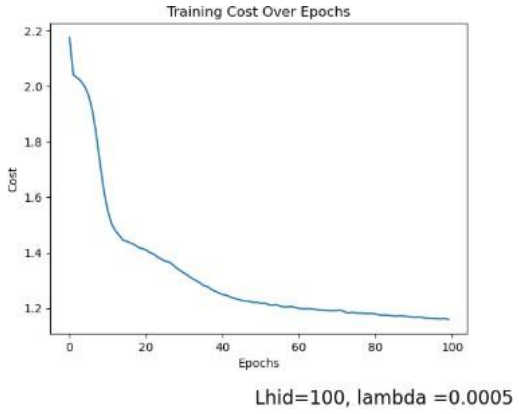

Figure 20: Training Loss Curve when L_{hid} = 100 and λ = 0.0005

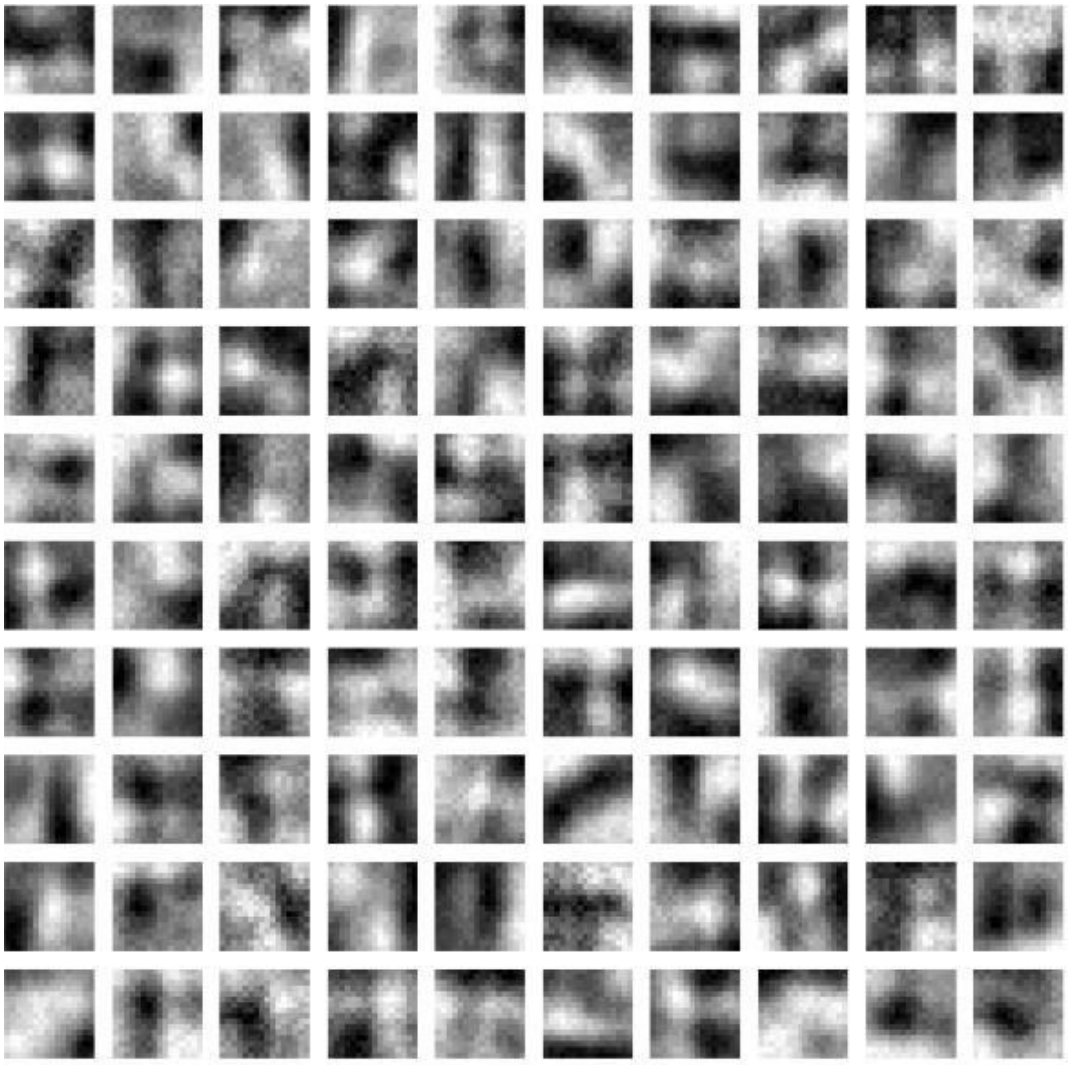

Figure 21: Learned Features when L_{hid} = 100 and λ = 0.0005

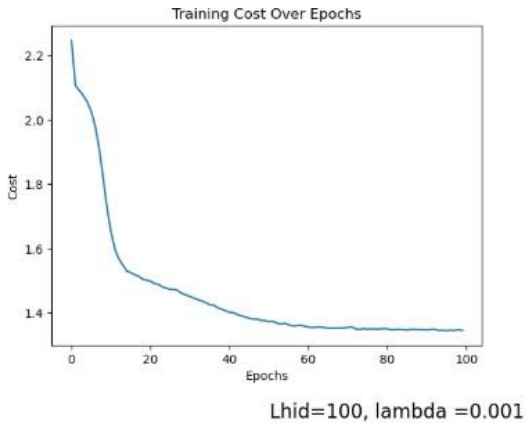

Figure 22: Training Loss Curve when $L_{hid} = 100$ and $\lambda = 0.001$

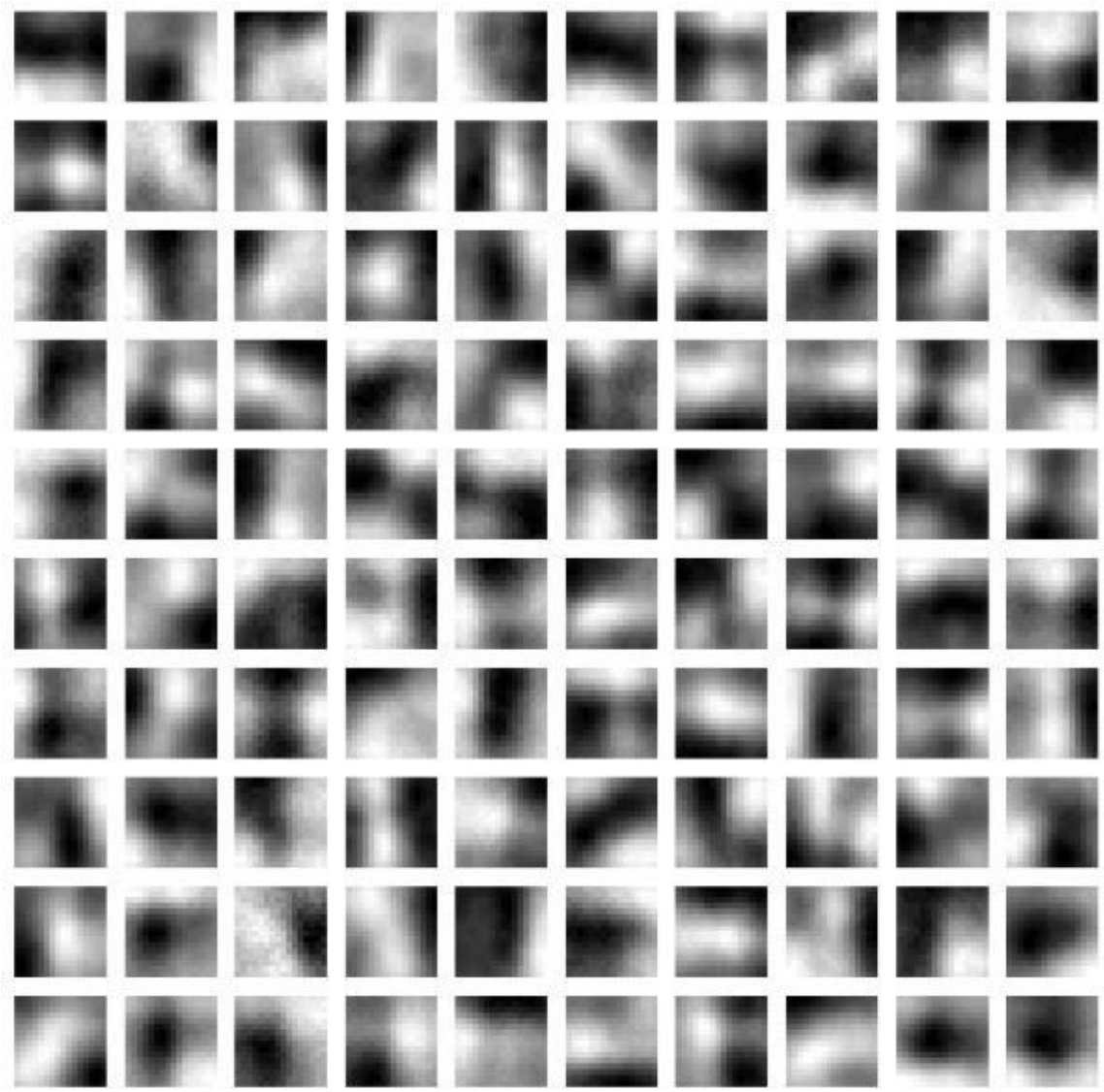

Figure 23: Learned Features when $L_{hid} = 100$ and $\lambda = 0.001$

When we look at the 9 figures which are obtained by differen low, medium and high values, we see some differences. For example, if we keep the lambda same and increase the number of neurons in

the hidden layer, the learned features become more noisy. In other words, they represents more general learned features rather than specific patterns. If we keep the number of neurons in hidden layer same and increase the lambda, we see that the model learns more clearly. It means that as the lambda increases, the learned features become more specific.

QUESTION 2

In this question, it is asked to predict the fourth word in sequence given the first three words in a trigram format using the neural networks which works fine for natural language processing tasks. The architecture which is implemented for this task is as follows:

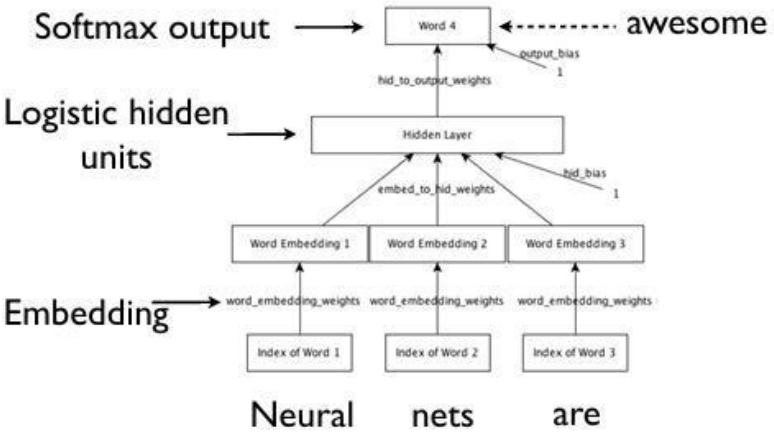

Figure 24: Architecture for the Question 2

As it can be seen clearly, the input layer has 3 different neurons which corresponds to 3 words in the trigram. However, since the neural networks cannot understand the vocabularies directly, we have to represent them in numerical format. Therefore, the embedding matrix is used to convert these words into vector representations which can be understood by the neural networks. Moreover, these representations are concatenated to throw in a neural network. In the design of the neural network, the sigmoid activation is preferred for the hidden layer. Furthermore, the softmax activation is used in the output layer of the network to produce the probabilities of each word and give answer taking the probabilities into consideration.

Part A

In this part, it is asked to train the network using the mini batch stochastic gradient descent with specified parameters like learning rate, batch size and momentum rate. However, before implementing these, we need to define and initialize our learnable parameters.

In this part, the parameters like weights and biases are initialized using the Gaussian variables having 0 mean and 0.01 standard deviations. Regarding these, the parameters which takes role in the network are as follows;

X : input vector which is a concatenation of the embeddings of the three words in the trigram

W_1: Connecting weights between input and hidden layer

b_1: bias for the hidden layer

σ: Sigmoid activation

Z_1: Induced field in the hidden layer

A_1: Activation of the hidden layer

W_2: Connecting weights between hidden and output layer

B_2: bias for the output layer

Z_2: Induced field in the output layer

A_2: Activation of the output layer which is softmax.

Then our forward pass like as follows,

$$Z_1 = W_1 \cdot X + b_1$$

$$A_1 = \sigma(Z_1)$$

$$Z_2 = W_2 \cdot A_1 + b_2$$

$$A_2 = \text{Softmax}(Z_2) = \frac{\exp(Z_2 - \max(Z_2))}{\sum_{j=1}^{250} \exp(Z_{2j} - \max(Z_2))}$$

In the forward pass, as you can see, I introduce different version of the softmax since the regular softmax suffers from exploding gradient problem. What I do here is the subtracting the mean of each row from itself to get better convergence. At the end, it gives probabilities like regular softmax.

For the loss calculation, the cross entropy loss is introduced like as follows,

$$J = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{250} y_j^{(i)} \log\left(A_{2j}^{(i)}\right)$$

The important part here is that the $y_j^{(i)}$ has a one hot encoding form for the target word since the target word is needed to defined as categorical value where the one of the words among the 250 words is correct. Thus, it gets whether 0 or 1. Besides these, since we use the mini batches, N represents the number of samples in the batch. Moreover, as mentioned before A_2 represents the probability for the related entry.

After doing the forward pass and loss calculations, we can now calculate the gradients of the network.

Error for the output layer is as follows,

$$\delta_{\text{out}} = A_2 - Y$$

where Y is the encoded actual label.

The gradients for the weight and bias term of the output layer are as follows,

$$\frac{\partial J}{\partial W_2} = \frac{1}{N} \delta_{\text{out}} \cdot A_1^{\intercal}$$

$$\frac{\partial J}{\partial b_2} = \frac{1}{N} \sum_{i=1}^{N} \delta_{\text{out}}$$

Moreover, the error for the hidden layer is as follows,

$$\delta_{\text{hidden}} = (W_2^{\mathsf{T}} \cdot \delta_{\text{out}}) \odot \sigma'(Z_1)$$

where $\sigma'(Z_1)$ is derivative of the sigmoid. So,

$$\sigma'(Z_1) = \sigma(Z_1) \cdot (1 - \sigma(Z_1))$$

Furthermore, the weights and biases for the hidden layer are as follows,

$$\frac{\partial J}{\partial W_1} = \frac{1}{N} \delta_{\text{hidden}} \cdot X^{\mathsf{T}}$$

$$\frac{\partial J}{\partial b_1} = \frac{1}{N} \sum_{i=1}^{N} \delta_{\text{hidden}}$$

Then, we need to average the gradients over the mini batch

$$\nabla J = \frac{1}{m} \sum_{i=1}^{m} \nabla J^{(i)}$$

After the calculations above are done, the parameters are updated with the mini batch stochastic gradient with momentum. So, the velocity updates for the momentum is as follows,

$$v_t = \alpha v_{t-1} + (1 - \alpha)\nabla J$$

Where v_t is the velocity, α is momentum rate and ∇J is the gradient of the cost function.

By using the velocity terms, the parameters can be updated as follows,

$$W \leftarrow W - \eta v_t, \quad b \leftarrow b - \eta v_t$$

where η is the learning rate.

In this task, the learning rate is chosen as 0.15, momentum rate is chosen as 0.85, number of epochs chosen as 50 and batch size chosen as 200.

Besides these, it is also asked to use early stopping which stops the algorithm based on the cross entropy error on the dataset. To do so, the early stopping algorithm is defined which works regarding the best validation loss and validation loss of the last epoch. For this algorithm, the patience number chosen as 2 which means that if the 2 consecutive epochs gives higher error than the minimum validation loss, the early stopping is triggered and the all process is stopped.

After giving the details for the calculations and the entire structure of the task, we can now move onto the training/testing process. In this task, it is asked to train the model with 3 different (D, P) pairs which are (32,256), (16,128), (8,64). Remember, while D represents the length of the vector representation, P represents the number of neurons in the hidden layer. The results for these pairs can be seen below.

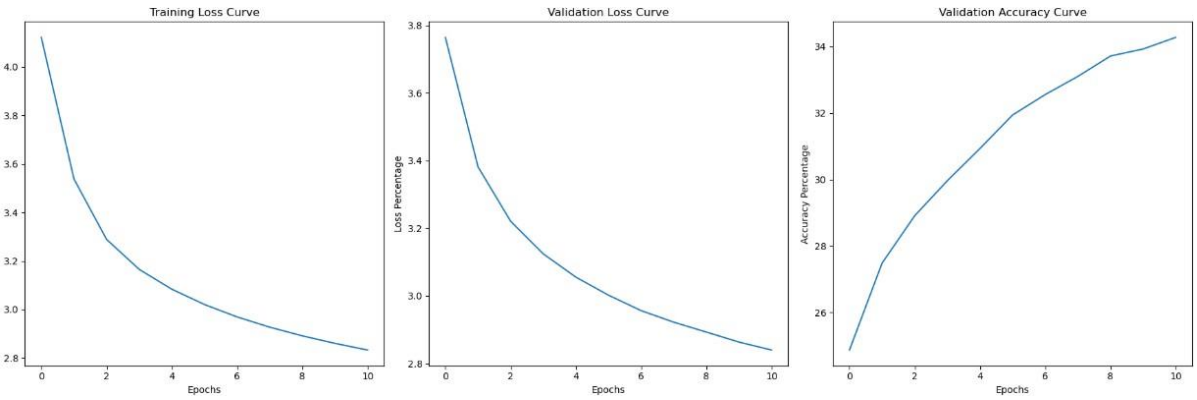

Figure 25: Training Loss, Validation Loss and Validation Accuracy Curves for D = 32, P = 256

```
Validation Cost: 2.8398, Validation Accuracy: 34.28%
Early stopping at epoch 12. Best validation cost: 2.8634
```

Figure 26: Triggered Early Stopping for D = 32, P = 256

As it can be seen in Figure 26, the validation accuracy for this pair is 34.28%.

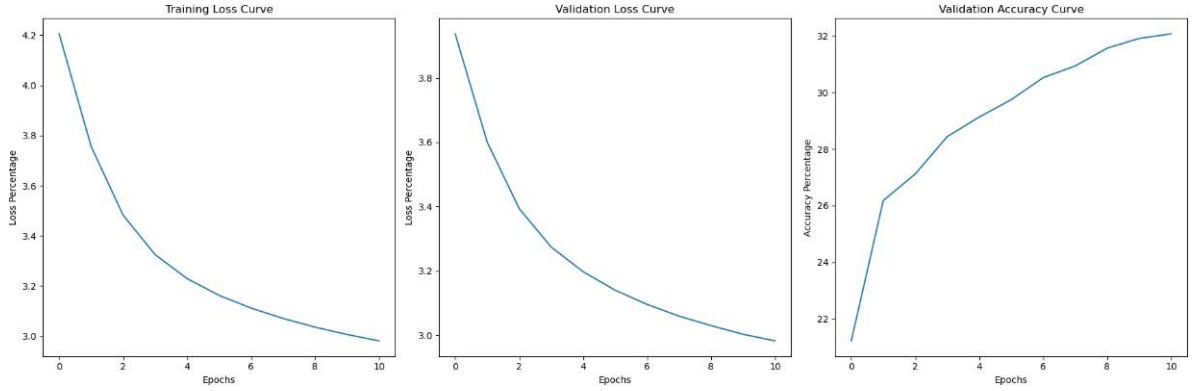

Figure 27: Training Loss, Validation Loss and Validation Accuracy Curves for D = 16, P = 128

```
Validation Cost: 2.9822, Validation Accuracy: 32.07%
Early stopping at epoch 12. Best validation cost: 3.0027
```

Figure 28: Triggered Early Stopping for D = 16, P = 128

As it can be seen in Figure 28, the validation accuracy for this pair is 32.07%.

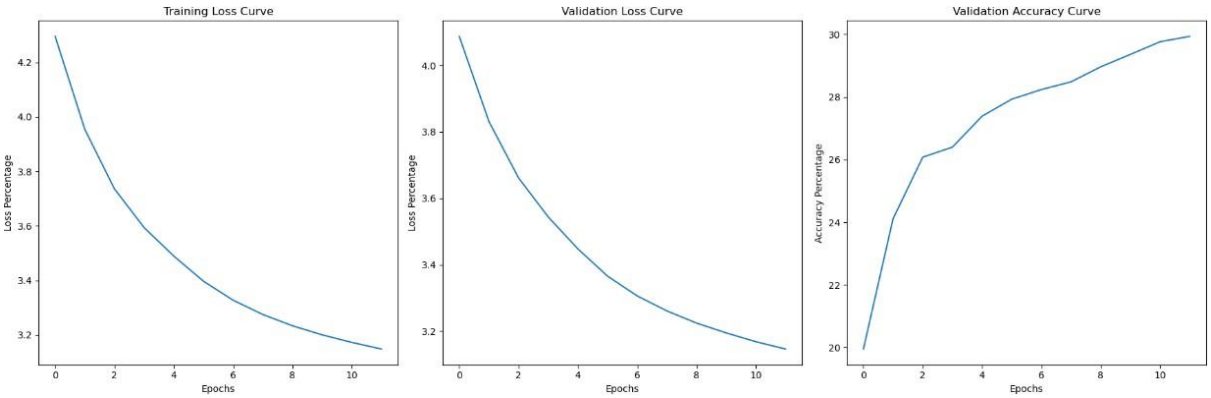

Figure 29: Training Loss, Validation Loss and Validation Accuracy Curves for D = 8, P = 64

```
Validation Cost: 3.1462, Validation Accuracy: 29.94%
Early stopping at epoch 13. Best validation cost: 3.1682
```

Figure 30: Triggered Early Stopping for D = 8, P = 64

As it can be seen in Figure 28, the validation accuracy for this pair is 29.94%.

If we compare the results obtained in Figure 26, Figure 28 and Figure 30, the highest validation accuracy is obtained in the Figure 26 which refers to the pair of D = 32, P = 256. If we look at the results in a more detailed way, we can observe that while the smoothness of the training and validation loss curves are similar for each pair, the validation accuracy curve for the pair of (D=32, P=256) are more smoother than the other pairs. Moreover, although the loss curves are similar in shape, the pair of (D = 32, P = 256) gets smaller validation cost and higher accuracy than others. If we compare the performance of these three pairs, we can make order like following,

(D = 32, P = 256) > (D = 16, P = 128) > (D = 8, P = 64)

Based on the comparisons above, we can conclude that increasing the D and P increases the performance. In other words, when D = 32, it means that we represent our words in a longer vector than others which may give a chance for better representations for the words. Moreover, when P = 256, it means that our network may be capable of learning more complex patterns or relationships in the data than others.

Part B

In that part, it is asked to pick some trigrams from the test data and generate the fourth word following the trigram. Moreover, to evaluate the performance of the model, it is asked to store top 10 candidates for the fourth word which have top ten probabilities among the 250 words. Then, it is asked to apply this process for 5 different trigrams. To answer this questions, I choose the pair of (D=32, P=256) since it gives better performance than other pairs as we see in Part A.

The randomly generated 5 trigrams, top ten candidate fourth words and the probabilities of the top ten candidate fourth words for these trigrams can be seen below.

```
Trigram 1: up to the
  Predicted Word: best
  Top 10 Predictions:
    Rank 1: best (Probability: 0.0657)
    Rank 2: same (Probability: 0.0632)
    Rank 3: way (Probability: 0.0632)
    Rank 4: time (Probability: 0.0414)
    Rank 5: game (Probability: 0.0406)
    Rank 6: world (Probability: 0.0319)
    Rank 7: case (Probability: 0.0311)
    Rank 8: first (Probability: 0.0303)
    Rank 9: end (Probability: 0.0296)
    Rank 10: money (Probability: 0.0294)
```

Figure 31: Top Ten Candidates for Trigram 1

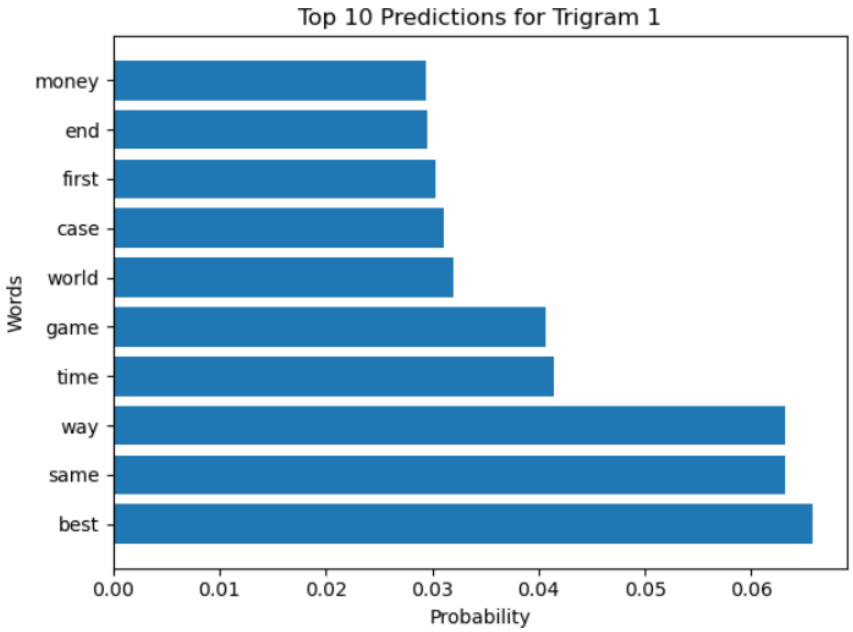

Figure 32: Comparison of Top Ten Candidates for Trigram 1

```
Trigram 2: does that put
  Predicted Word: it
  Top 10 Predictions:
    Rank 1: it (Probability: 0.1657)
    Rank 2: ? (Probability: 0.1340)
    Rank 3: me (Probability: 0.0705)
    Rank 4: the (Probability: 0.0698)
    Rank 5: them (Probability: 0.0605)
    Rank 6: that (Probability: 0.0471)
    Rank 7: him (Probability: 0.0348)
    Rank 8: us (Probability: 0.0270)
    Rank 9: to (Probability: 0.0268)
    Rank 10: this (Probability: 0.0264)
```

Figure 33: Top Ten Candidates for Trigram 2

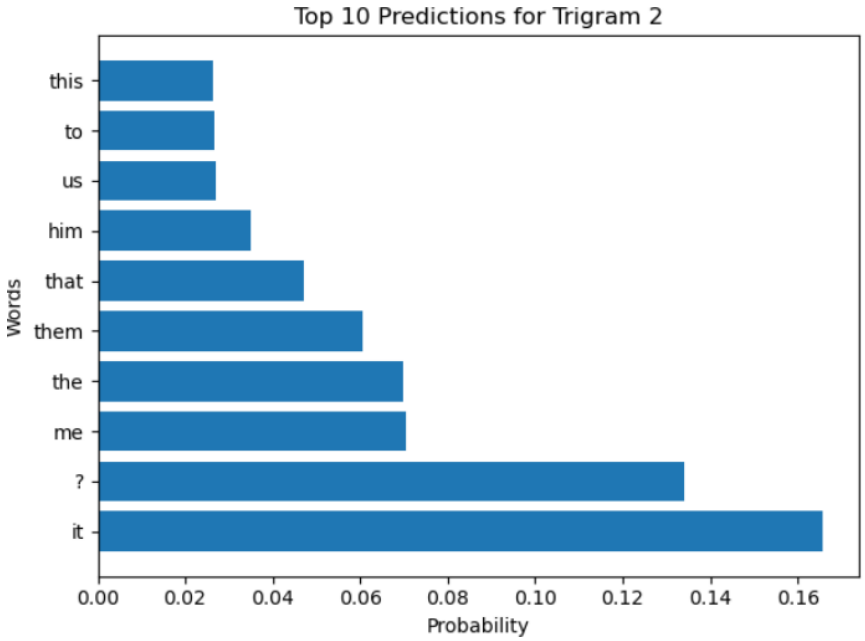

Figure 34: Comparison of Top Ten Candidates for Trigram 2

```
Trigram 3: after all ,
  Predicted Word: he
  Top 10 Predictions:
    Rank 1: he (Probability: 0.1588)
    Rank 2: i (Probability: 0.1035)
    Rank 3: it (Probability: 0.0819)
    Rank 4: she (Probability: 0.0815)
    Rank 5: you (Probability: 0.0382)
    Rank 6: they (Probability: 0.0358)
    Rank 7: we (Probability: 0.0340)
    Rank 8: that (Probability: 0.0294)
    Rank 9: but (Probability: 0.0283)
    Rank 10: what (Probability: 0.0269)
```

Figure 35: Top Ten Candidates for Trigram 3

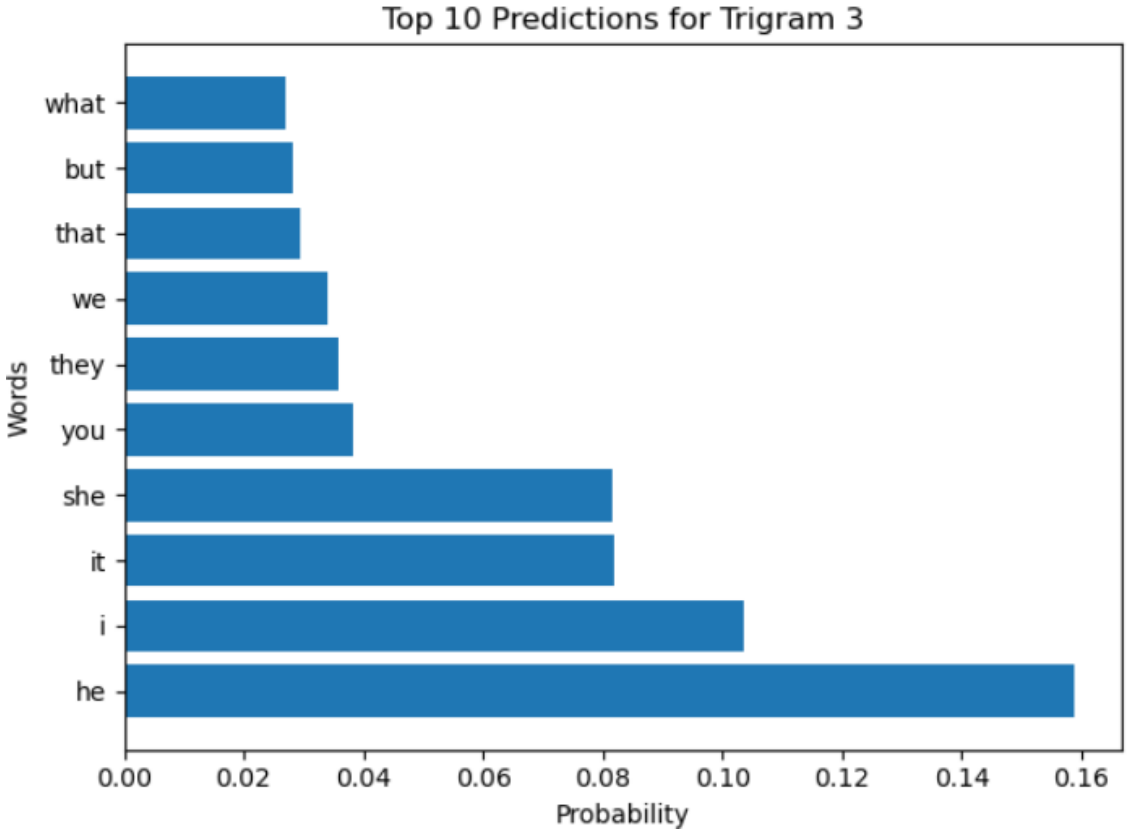

Figure 36: Comparison of Top Ten Candidates for Trigram 3

```
Trigram 4: of it ,
  Predicted Word: he
  Top 10 Predictions:
    Rank 1: he (Probability: 0.3706)
    Rank 2: she (Probability: 0.1397)
    Rank 3: i (Probability: 0.0684)
    Rank 4: it (Probability: 0.0426)
    Rank 5: you (Probability: 0.0309)
    Rank 6: they (Probability: 0.0300)
    Rank 7: we (Probability: 0.0283)
    Rank 8: but (Probability: 0.0276)
    Rank 9: and (Probability: 0.0271)
    Rank 10: too (Probability: 0.0246)
```

Figure 37: Top Ten Candidates for Trigram 4

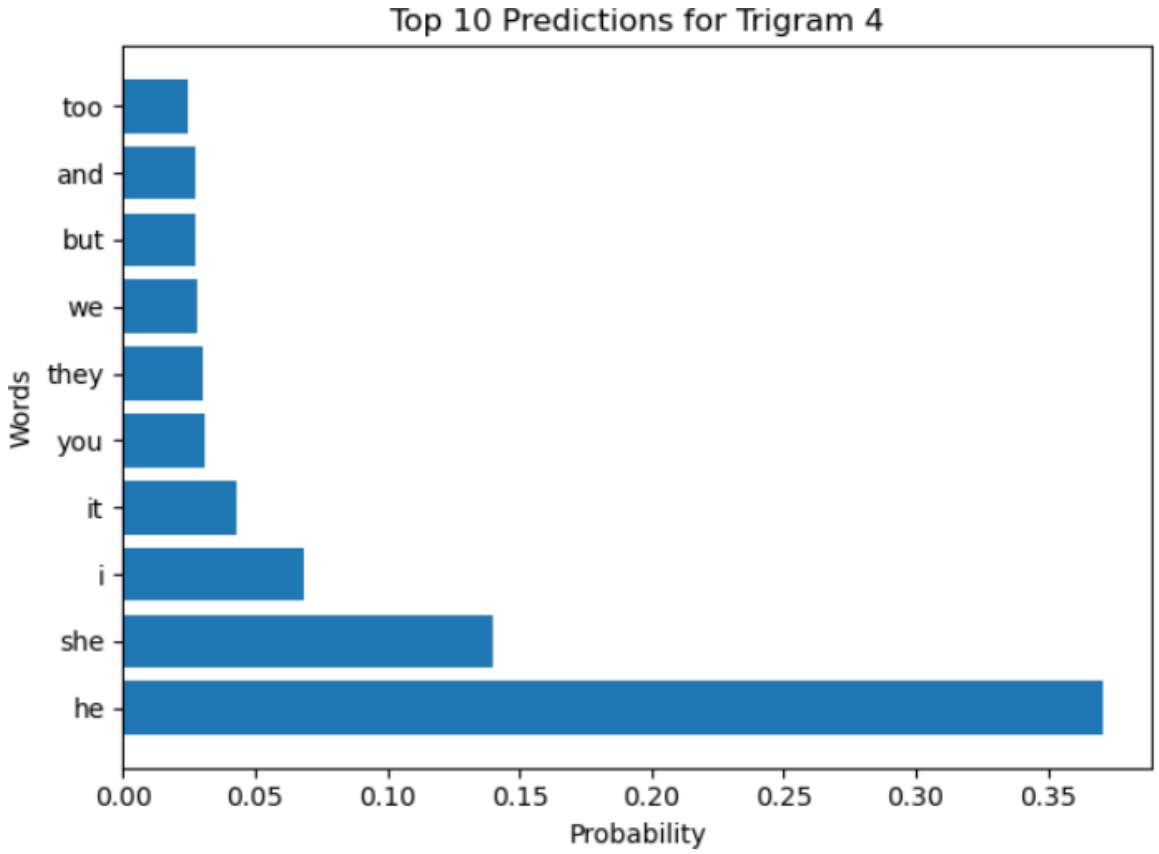

Figure 38: Comparison of Top Ten Candidates for Trigram 4

```
Trigram 5: but what did
  Predicted Word: you
  Top 10 Predictions:
    Rank 1: you (Probability: 0.2130)
    Rank 2: i (Probability: 0.1525)
    Rank 3: we (Probability: 0.1436)
    Rank 4: he (Probability: 0.1098)
    Rank 5: they (Probability: 0.1005)
    Rank 6: it (Probability: 0.0943)
    Rank 7: she (Probability: 0.0217)
    Rank 8: not (Probability: 0.0162)
    Rank 9: that (Probability: 0.0144)
    Rank 10: nt (Probability: 0.0121)
```

Figure 39: Top Ten Candidates for Trigram 5

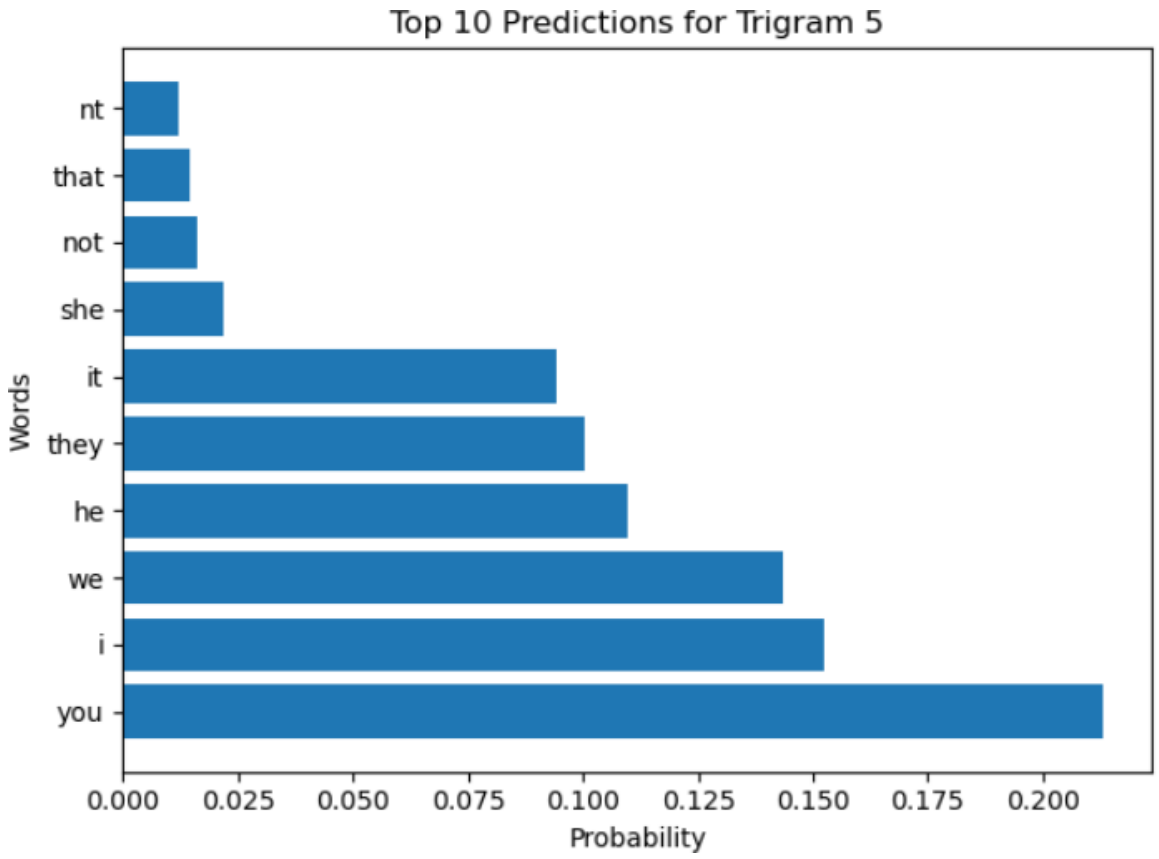

Figure 40: Comparison of Top Ten Candidates for Trigram 5

If the analyze the results we get, the predictions seems reasonable and sensible. Although the context length is very short, the predictions for the fourth word are sensible and mostly grammatically correct. Specifically, the candidate word with the highest probability is a sensible and reasonable word for the fourth word predictions. Moreover, the candidates other than the best candidate are also reasonable, sensible and grammatically correct. Thus, we can say that our neural

network confirm our assumption at the beginning of the question which is related to the powerful capabilities of the neural networks in the natural language processing tasks.

QUESTION 3

In this question, it is asked to do multi class time series classification using the Recurrent Neural Networks (RNN), Long Short Term Memort (LSTM) and Gated Recurrent Unit (GRU). More specifically, we try to classify human activities like downstairs, jogging, sitting, standing, upstairs and walking from movement signals.

Part A

In this part, it is asked to implement a single layer recurrent network with 128 neurons and followed by the multi layer perceptron having softmax activation in its output layer to make classification. Before starting our task, we can visit the general architecture of the RNN and backpropagation through time (BPTT) algorithm as follows,

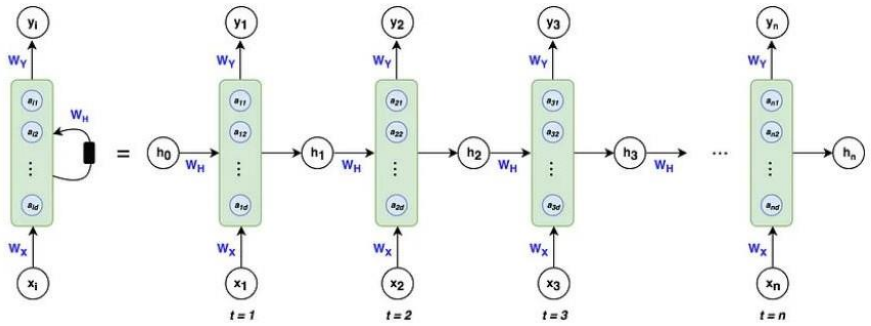

Figure 41: RNN Architecture and its Unfolded Version [2]

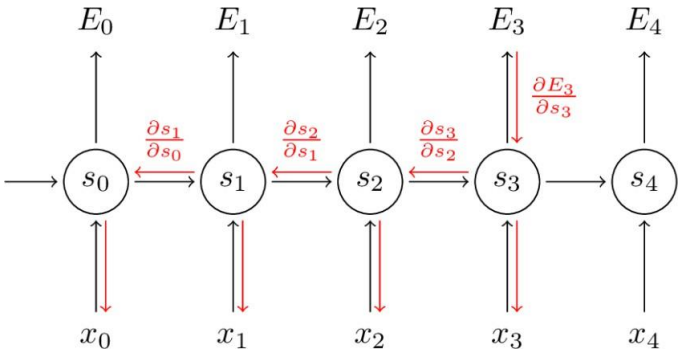

Figure 42: BPTT Algorithm in Unfolded Version of RNN [3]

After visiting these, we can start the question by initializing our weights regarding the Xavier initializations like the formula given below.

$$W, b \sim \mathcal{U}\left(-\sqrt{\frac{6}{\text{Lin} + \text{Lout}}}, \sqrt{\frac{6}{\text{Lin} + \text{Lout}}}\right)$$

The weights and parameters are defined uniformly regarding the Xavier initialization given above. After defining and initializing our learnable parameters, we can start with the forward pass. Let's start with the forward pass of the RNN.

The forward pass of the RNN is as follows,

$$h_t = \tanh(X_t \cdot W_{\text{in}} + h_{t-1} \cdot W_h + b_h)$$

where

X_t: our input at time step t

W_{in}: connecting weights between input layer and hidden layer of RNN

W_h: connecting weights between hidden layer and hidden layer of RNN

b_h: bias in the hidden layer

h_{t-1}: hidden state from previous time step t-1

h_t: hidden state from current time step t

Also, after completing the all time steps, the final step is calculated to use it in MLP later. The final step is,

$$h_{\text{final}} = h_T$$

Then, we have the forward pass of the MLP with single hidden layer. The process like as follows,

$$z_{\text{hidden}} = h_{\text{final}} \cdot W_{\text{hidden}} + b_{\text{hidden}}$$

$$h_{\text{hidden}} = \text{ReLU}(z_{\text{hidden}})$$

$$z_{\text{out}} = h_{\text{hidden}} \cdot W_{\text{out}} + b_{\text{out}}$$

$$Y_{\text{pred}} = \text{Softmax}(z_{\text{out}})$$

where

W_{hidden}: weights of hidden layer of the MLP

b_{hidden}: biases of the hidden layer of the MLP

W_{out}: weights of output layer of the MLP

b_{out}: biases of the output layer of the MLP

In the hidden activation, the ReLU is preferred due to the its better performance compared to other activation functions.

After completing the forward pass part, we can compute the loss function like as follows,

$$J = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{\text{output size}} Y_j^{(i)} \log\left(Y_{\text{pred},j}^{(i)}\right)$$

where $Y_j^{(i)}$ one hot encoded actual label and $Y_{\text{pred},j}^{(i)}$ is the predicted probability for given entry. Moreover, the m is the batch size since we will use mini batch stochastic gradient descent.

After completing these parts, we can move onto the gradient calculations.

Error at the output layer of the MLP is as follows,

$$\delta_{\text{out}} = \frac{1}{m}(Y_{\text{pred}} - Y)$$

Gradients for weights and biases of the output layer are as follows,

$$\frac{\partial J}{\partial W_{\text{out}}} = h_{\text{hidden}}^{\mathsf{T}} \cdot \delta_{\text{out}}$$

$$\frac{\partial J}{\partial b_{\text{out}}} = \sum_{i=1}^{m} \delta_{\text{out},i}$$

Error at the hidden layer of the MLP

$$\delta_{\text{hidden}} = (\delta_{\text{out}} \cdot W_{\text{out}}^{\mathsf{T}}) \odot \text{ReLU}'(z_{\text{hidden}})$$

Then gradients for the weights and biases of the hidden layer of the MLP are as follows

$$\frac{\partial J}{\partial W_{\text{hidden}}} = h_{\text{final}}^{\mathsf{T}} \cdot \delta_{\text{hidden}}$$

$$\frac{\partial J}{\partial b_{\text{hidden}}} = \sum_{i=1}^{m} \delta_{\text{hidden},i}$$

Now, it is time for BPTT of RNN. (I have put a figure because when I implement this in word, word documents stops working)

$$\delta_t = \left(\delta_{t+1} \cdot W_h^{\top} + \delta_{\text{final}}\right) \odot \tanh'(h_t)$$

$$\tanh'(z) = 1 - z^2$$

Then the gradient for the RNN weights and biases are as follows,

$$\frac{\partial J}{\partial W_{\text{in}}} = \sum_{t=1}^{T} X_t^{\mathsf{T}} \cdot \delta_t$$

$$\frac{\partial J}{\partial W_h} = \sum_{t=1}^{T} h_{t-1}^{\mathsf{T}} \cdot \delta_t$$

$$\frac{\partial J}{\partial b_h} = \sum_{t=1}^{T} \delta_t$$

The all workflow of the network which uses RNN and MLP can be seen above.

Besides these, the update algorithm as like we did in Question 2,

$$\nabla J = \frac{1}{m} \sum_{i=1}^{m} \nabla J^{(i)}$$

$$v_t = \alpha v_{t-1} + (1 - \alpha)\nabla J$$

$$W \leftarrow W - \eta v_t, \quad b \leftarrow b - \eta v_t$$

Moreover, like we did before, the early stopping algorithm tracks the validation error. If the consecutive 7 epochs gives higher validation error than the minimum validation error, the algorithm stops. Also note that since the our dataset does not have validation data directly, we split our training set as an 90% training and 10% validation.

In this task, the network is designed with the following parameters; number of neurons in the hidden layer of RNN is 128, number of neurons in the hidden layer of the MLP is 64, the learning rate is 0.001, batch size is 32, epochs is 50, patience for early stopping is 7 and momentum is 0.85. Regarding these, the results obtained for this part are as follows,

```
Epoch 47/50 | Train Loss: 1.3958, Train Acc: 42.70%, Val Loss: 1.4040, Val Acc: 43.67%
Early stopping triggered.

Test Accuracy: 37.50%
```

Figure 43: Early Stopping for RNN

As you can see from the Figure 43, the test accuracy for the RNN is 37.50%

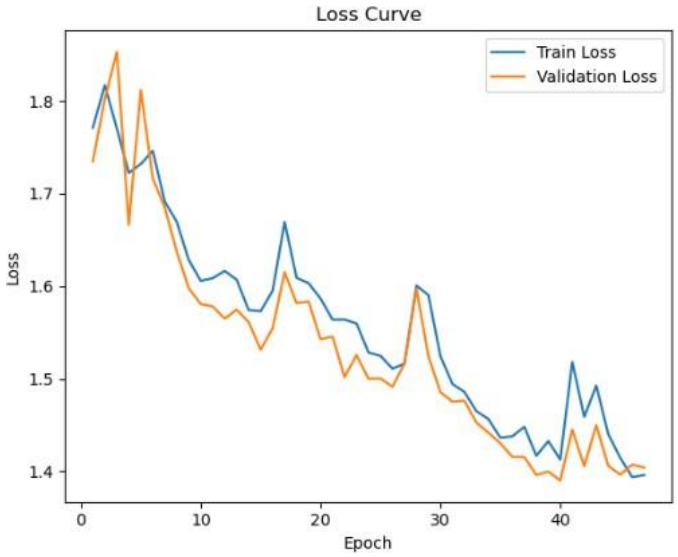

Figure 44: Train and Validation Loss Curves for RNN

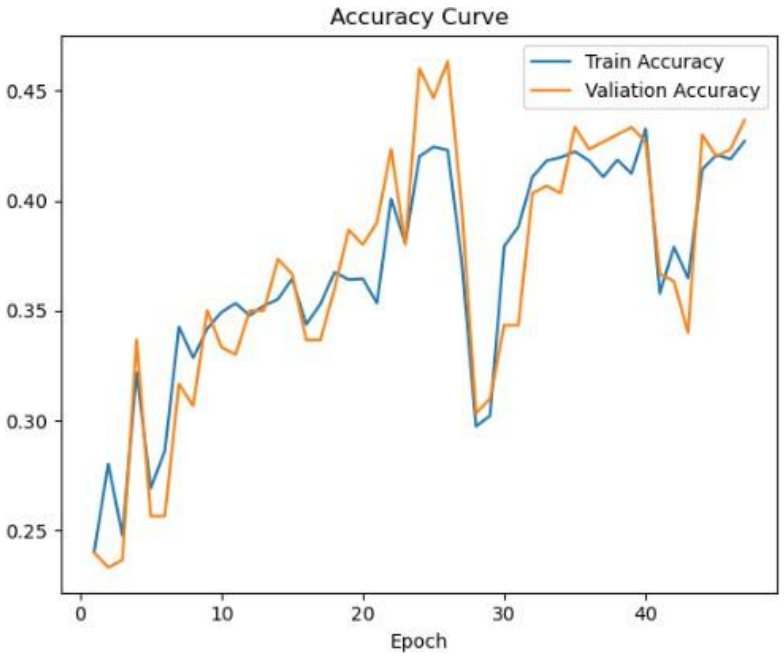

Figure 45: Train and Validation Accuracy Curves for RNN

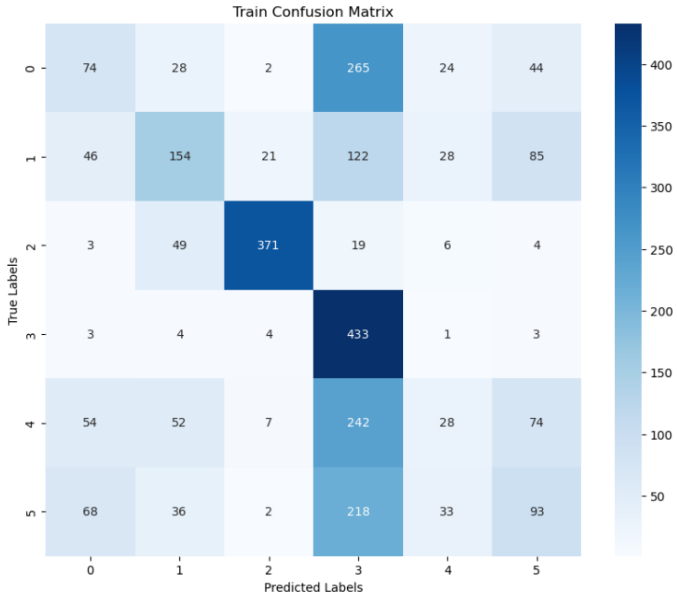

Figure 46: Train Confusion Matrix for RNN

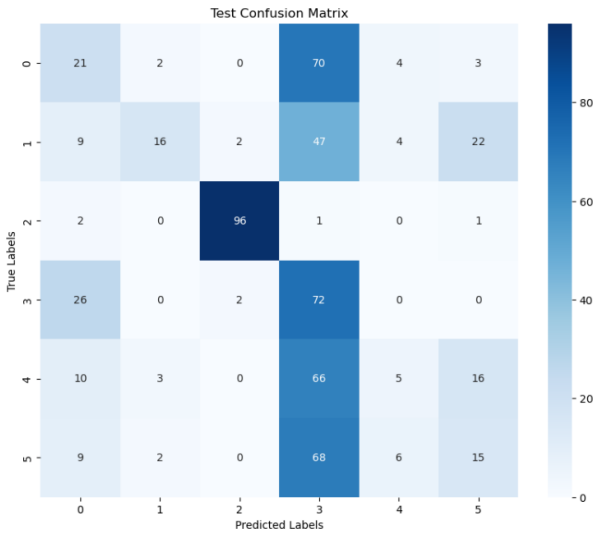

Figure 47: Test Confusion Matrix for RNN

When we look at the results we obtained, we can see that the RNN performs poorly for this task. It gets 37.50% test accuracy which is not sufficient such a multi class task. Moreover, when we look at the training and validation loss curves, we see that while the loss reduces in the long run, it gives some picks at some points which means that the learning process is not stable. Furthermore, when we look at the training and test confusion matrix, we don't see good results. For good networks, we expect that the highest numbers occur in the diagonal of the confusion matrix which means that the predictions match with actual results. However, it is not the case in RNN. If we look at the Figure 47, although the assignments for class with label 2 works well, we cannot say the same thing for class with label 3.

Part B

In this part, it is asked to do same things by using LSTM. First, let's visit the general LSTM architecture.

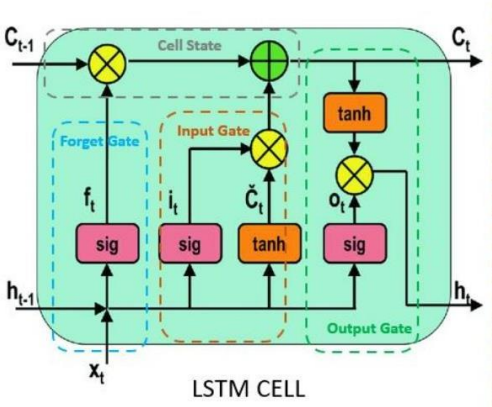

Figure 48: LSTM Architecture [4]

Since we talked about the details of the task before, let's start directly the implementation.

The forward pass of the LSTM is as follows,

$$z_t = X_t W_{\text{in}} + h_{t-1} W_h + b$$

$$f_t = \sigma(z_t[:, :h]), \quad i_t = \sigma(z_t[:, h:2h\}), \quad o_t = \sigma(z_t[:, 2h:3h\}), \quad C_t = \tanh(z_t[:, 3h:4h\})$$

$$C_t = f_t \odot C_{t-1} + i_t \odot C_t$$

$$h_t = o_t \odot \tanh(C_t)$$

where f_t is forget gate, i_t is input gate, o_t is output gate, C_t is cell state and $\tilde{C}_t$ is the candidate cell state.

Then the forward pass of the MLP is as follows,

$$z_{\text{hidden}} = h_T W_{\text{hidden}} + b_{\text{hidden}}, \quad h_{\text{hidden}} = \text{ReLU}(z_{\text{hidden}})$$

$$z_{\text{hidden2}} = h_{\text{hidden}} W_{\text{hidden2}} + b_{\text{hidden2}}, \quad h_{\text{hidden2}} = \text{ReLU}(z_{\text{hidden2}})$$

$$z_{\text{out}} = h_{\text{hidden2}} W_{\text{out}} + b_{\text{out}}, \quad Y_{\text{pred}} = \text{Softmax}(z_{\text{out}})$$

After completing the forward pass, we can calculate the loss function like as follows,

$$J = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{c} Y_{ij} \log(Y_{\text{pred},ij})$$

After completing these steps, we can calculate the gradients. First for the MLP,

$$\delta_{\text{out}} = \frac{1}{m}(Y_{\text{pred}} - Y)$$

$$\frac{\partial J}{\partial W_{\text{out}}} = h_{\text{hidden2}}^{\mathsf{T}} \delta_{\text{out}}, \quad \frac{\partial J}{\partial b_{\text{out}}} = \sum_{i=1}^{m} \delta_{\text{out},i}$$

$$\delta_{\text{hidden2}} = (\delta_{\text{out}} W_{\text{out}}^{\mathsf{T}}) \odot \text{ReLU}'(z_{\text{hidden2}})$$

$$\frac{\partial J}{\partial W_{\text{hidden2}}} = h_{\text{hidden}}^{\mathsf{T}} \delta_{\text{hidden2}}, \quad \frac{\partial J}{\partial b_{\text{hidden2}}} = \sum_{i=1}^{m} \delta_{\text{hidden2},i}$$

$$\delta_{\text{hidden1}} = (\delta_{\text{hidden2}} W_{\text{hidden2}}^{\mathsf{T}}) \odot \text{ReLU}'(z_{\text{hidden}})$$

$$\frac{\partial J}{\partial W_{\text{hidden}}} = h_T^{\mathsf{T}} \delta_{\text{hidden1}}, \quad \frac{\partial J}{\partial b_{\text{hidden}}} = \sum_{i=1}^{m} \delta_{\text{hidden1},i}$$

Then for the LSTM, (Again I have put the figure because when I write tanh in word document, word document stops working)

$$\delta_{h_t} = \delta_{\text{hidden1}} W_{\text{hidden}}^{\top} + \delta_{h_{t+1}} W_h^{\top}$$

$$\delta_{C_t} = \delta_{h_t} \odot o_t \odot \tanh'(C_t) + \delta_{C_{t+1}} \odot f_{t+1}$$

$$\delta_{o_t} = \delta_{h_t} \odot \tanh(C_t) \odot \sigma'(o_t)$$

$$\delta_{f_t} = \delta_{C_t} \odot C_{t-1} \odot \sigma'(f_t)$$

$$\delta_{i_t} = \delta_{C_t} \odot \tilde{C}_t \odot \sigma'(i_t)$$

$$\delta_{\tilde{C}_t} = \delta_{C_t} \odot i_t \odot \tanh'(\tilde{C}_t)$$

$$\frac{\partial J}{\partial W_{\text{in}}} = \sum_{t=1}^{T} X_t^{\top} \delta_t$$

$$\frac{\partial J}{\partial W_h} = \sum_{t=1}^{T} h_{t-1}^{\top} \delta_t$$

$$\frac{\partial J}{\partial b} = \sum_{t=1}^{T} \delta_t$$

All workflow for the LSTM followed by MLP network is like above.

After completing these, let's move onto the results,

```
Early stopping triggered at epoch 15, best validation loss: 0.5893
Test Accuracy: 54.67%
```

Figure 49: Early Stopping for LSTM

As it can be seen in Figure 49, the test accuracy for LSTM is 54.67%

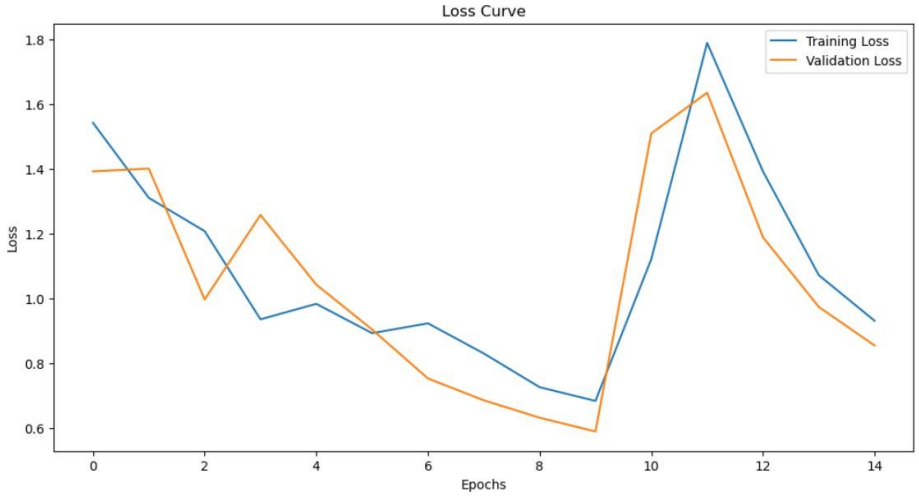

Figure 50: Train and Validation Loss Curves for LSTM

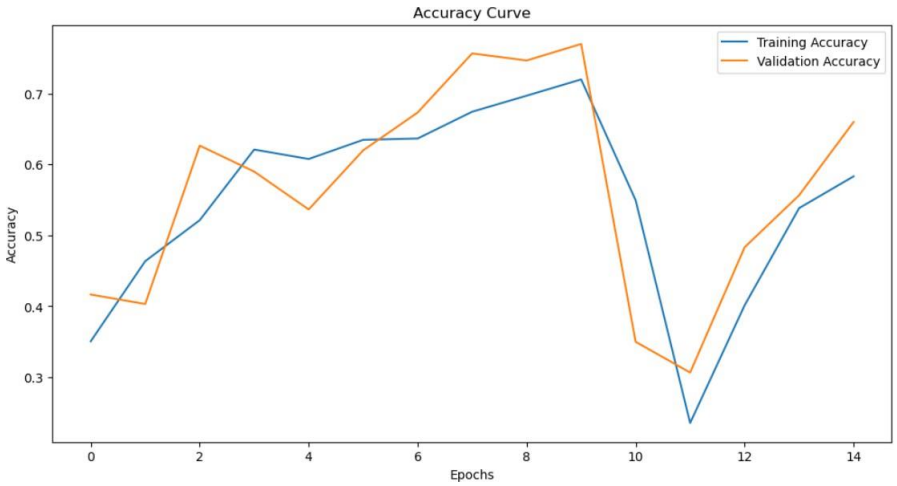

Figure 51: Train and Validation Accuracy Curves for LSTM

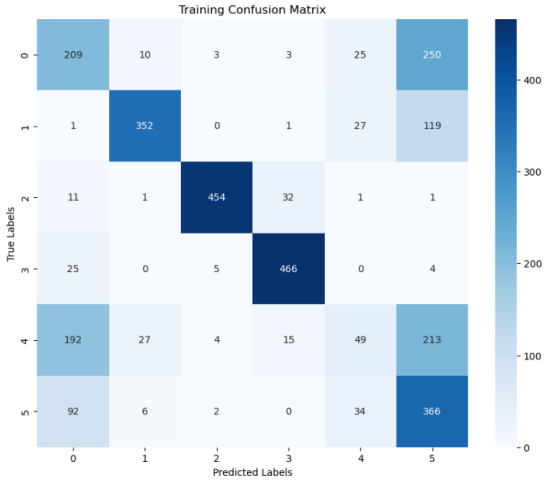

Figure 52: Train Confusion Matrix for LSTM

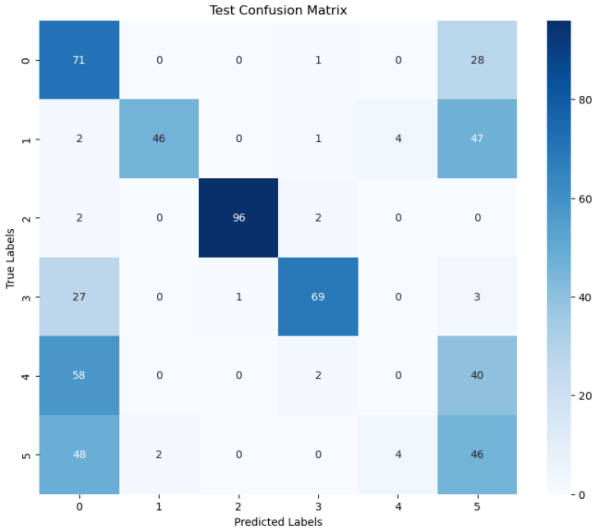

Figure 53: Test Confusion Matrix for LSTM

When we look at the results, we can observe that the test accuracy of the LSTM which is 54.67% is higher than the test accuracy of the RNN which is 37.50. Moreover, when we look at the training and validation loss curve, we observe that LSTM decreases more stable than the RNN does. There are several reasons why LSTM works better than the RNN. If we look at the architectures of the RNN and LSTM which is given in Figure 41 and Figure 48 respectively, we observe that the LSTM uses gates which are called forget gate, input gate, output gate and cell state which are responsible for forgetting the information and storing the information. While these gates control the information flow, cell state behaves like memory and carry the information between different time steps. Moreover, these components make the backpropagation effective and provides more stable learning and also give a chance network to learning longer contexts compared to RNN. Furthermore, LSTMs give a chance to eliminate the vanishing/exploding gradient problems we face in RNNs. If we look at the confusion matriecs of the LSTM, we observe that the diagonals of the confusion metrics have mostly the greater numbers which is good. However, there are still significant numbers in the blocks other than diagonal. When we put all these together, LSTM gives better performance compared to RNNs.

Part C

In this part, it is asked to the similar work for GRU. Let's first visit the general structure of the GRU.

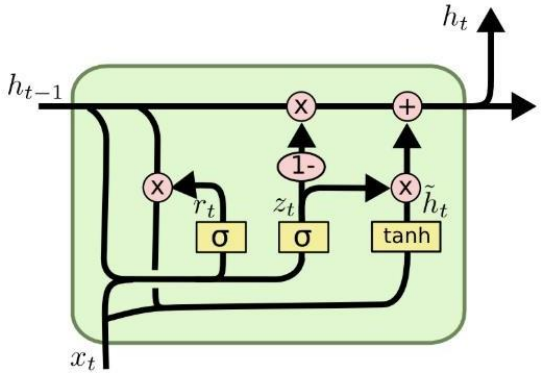

Figure 54: GRU Architecture [5]

Since we are familiar with the details of the task, let's do forward pass. First for GRU,

$$z = X_t W_{\text{in}} + h_{t-1} W_h + b$$

$$z_t = \sigma(z[:, :\text{hidden\_size}]) \quad (\text{update gate})$$

$$r_t = \sigma(z[:, \text{hidden\_size}: 2 \cdot \text{hidden\_size}]) \quad (\text{reset gate})$$

$$\hbar_t = \tanh(z[:, 2 \cdot \text{hidden\_size}: 3 \cdot \text{hidden\_size}]) \quad (\text{candidate hidden state})$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hbar_t$$

Then for the MLP,

$$z_{\text{hidden}} = h_T W_{\text{hidden}} + b_{\text{hidden}}$$

$$h_{\text{hidden}} = \text{ReLU}(z_{\text{hidden}})$$

$$z_{\text{hidden2}} = h_{\text{hidden}} W_{\text{hidden2}} + b_{\text{hidden2}}$$

$$h_{\text{hidden2}} = \text{ReLU}(z_{\text{hidden2}})$$

$$\text{logits} = h_{\text{hidden2}} W_{\text{out}} + b_{\text{out}}$$

$$y_{\text{pred}} = \text{softmax}(\text{logits})$$

After completing the forward pass part, let's calculate the loss function

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_i^{(c)} \log y_{\text{pred}}^{(c)}$$

Now, let's move onto the gradients. First for the MLP,

$$\delta_{\text{out}} = \frac{1}{N}(y_{\text{pred}} - y_{\text{true}})$$

$$\frac{\partial J}{\partial W_{\text{out}}} = h_{\text{hidden2}}^{\mathsf{T}} \delta_{\text{out}}$$

$$\frac{\partial J}{\partial b_{\text{out}}} = \sum_{i=1}^{N} \delta_{\text{out}}$$

$$\delta_{\text{hidden2}} = \delta_{\text{out}} W_{\text{out}}^{\mathsf{T}} \odot \text{ReLU}'(z_{\text{hidden2}})$$

$$\frac{\partial J}{\partial W_{\text{hidden2}}} = h_{\text{hidden}}^{\mathsf{T}} \delta_{\text{hidden2}}$$

$$\frac{\partial J}{\partial b_{\text{hidden2}}} = \sum_{i=1}^{N} \delta_{\text{hidden2}}$$

$$\delta_{\text{hidden}} = \delta_{\text{hidden2}} W_{\text{hidden2}}^{\mathsf{T}} \odot \text{ReLU}'(z_{\text{hidden}})$$

$$\frac{\partial J}{\partial W_{\text{hidden}}} = h_T^{\mathsf{T}} \delta_{\text{hidden}}$$

$$\frac{\partial J}{\partial b_{\text{hidden}}} = \sum_{i=1}^{N} \delta_{\text{hidden}}$$

Then for the GRU,

$$\delta_{\tilde{h}_t} = \delta_t \odot z_t$$

$$\delta_{z_t} = \delta_t \odot (\tilde{h}_t - h_{t-1})$$

$$\delta_{\text{concat}} = \left[\delta_{z_t} \odot \sigma'(z_t),\ \delta_{r_t} \odot \sigma'(r_t),\ \delta_{\tilde{h}_t} \odot \tanh'(\tilde{h}_t)\right]$$

$$\frac{\partial J}{\partial W_{\text{in}}} = \sum_{t=1}^{T} X_t^{\mathsf{T}} \delta_{\text{concat}}$$

$$\frac{\partial J}{\partial W_h} = \sum_{t=1}^{T} h_{t-1}^{\mathsf{T}} \delta_{\text{concat}}$$

$$\frac{\partial J}{\partial b} = \sum_{t=1}^{T} \delta_{\text{concat}}$$

All the workflow for GRU followed by MLP is as above. Now, let's move onto the obtained results,

```
Epoch 38/50, Train Loss: 0.2005, Train Acc: 93.48%, Val Loss: 0.3185, Val Acc: 91.33 %
Early stopping triggered at epoch 38, best validation loss: 0.2484
Test Accuracy: 86.50%
```

Figure 55: Early Stopping for GRU

As it can be seen in the Figure 55, the test accuracy for GRU is 86.50%

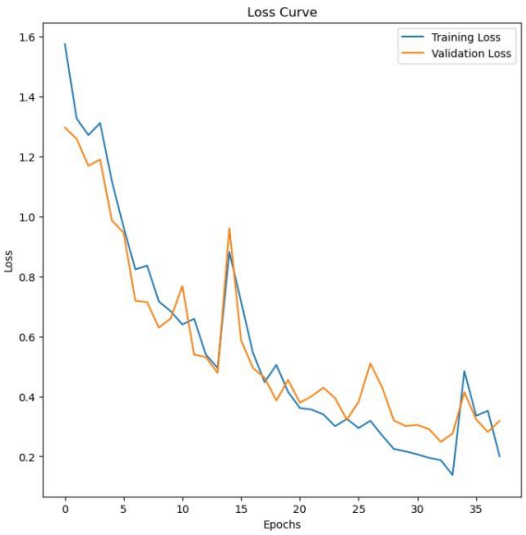

Figure 56: Training and Validation Loss Curve for GRU

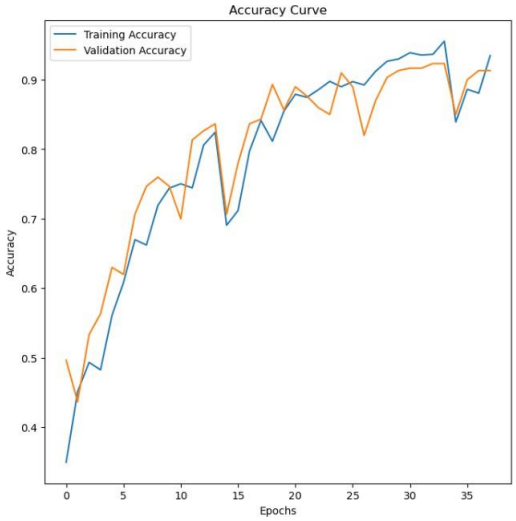

Figure 57: Training and Validation Accuracy Curve for GRU

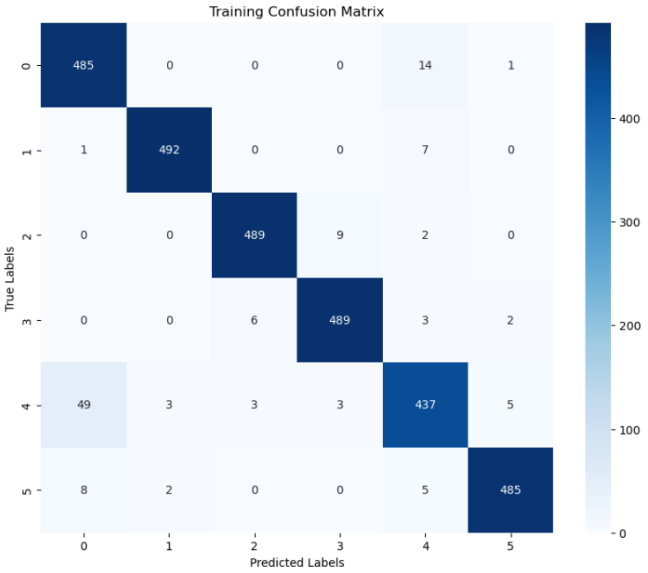

Figure 58: Train Confusion Matrix for GRU

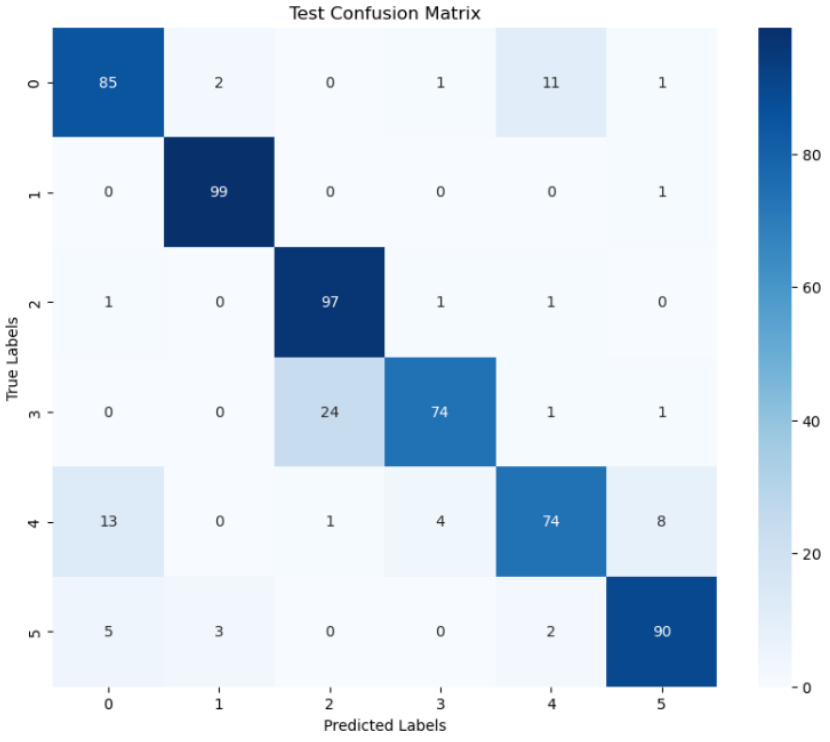

Figure 59: Test Confusion Matrix for GRU

When we look at the results of the GRU, we see that the test accuracy of the GRU which is %86.50 is significantly higher than the test accuracies obtained in RNN and LSTM which are 37.50% and 54.67% respectively. Moreover, when we look at the training and validation curves of the GRU, we observe that stable learning. Also, similar things can be said about the training and validation accuracy curves. When we look at the accuracy curves of the models, we can clearly observe that the accuracy increases more stable than the RNN or LSTM does. There are several reasons for them. GRU uses a

similar architecture with LSTM but with less parameters. Therefore, GRU is easier to train and more efficient in applications. Similar to the LSTMs, GRUs have a mechanism which decides whether the information will be stored or forgotten. Moreover, since the GRUs has less parameters than other models, they cannot easily overfit the data which is good for us. As a result, simplification in the architecture and computations while keeping the functionality similar, makes the GRUs better compared to RNNs and LSTMs.

REFERENCES

[1] A. Bharadwaj, "Introduction to Machine Learning," *Achyuthabharadwaj's Blog*. [Online]. Available: https://achyuthabharadwaj.github.io/Intro-to-ML-4/. [Accessed: Dec. 22, 2024]

[2] A. Bose, "Whirlwind Tour of RNNs," *Towards AI*, Apr. 2023. [Online]. Available: https://pub.towardsai.net/whirlwind-tour-of-rnns-a11effb7808f. [Accessed: Dec. 22, 2024]

[3] S. Shah, "Let's Understand the Problems With Recurrent Neural Networks," *Analytics Vidhya*, Jul. 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/07/lets-understand-the-problems-with-recurrent-neural-networks/. [Accessed: Dec. 22, 2024]

[4] "Introduction to LSTM Units in RNN," *Pluralsight*, [Online]. Available: https://www.pluralsight.com/resources/blog/guides/introduction-to-lstm-units-in-rnn. [Accessed: Dec. 22, 2024]

[5] "GRU (Gated Recurrent Unit)," *Papers with Code*, [Online]. Available: https://paperswithcode.com/method/gru. [Accessed: Dec. 22, 2024]