Table of Contents

1. Auto-Encoder Neural Network

Question 1.1

In this sub-question, I have plotted 200 randomly chosen images from the dataset, both in RGB and normalized grayscale view. The images are given in Figures 1 and 2. The images are turned into grayscale following the steps provided in the prompt.

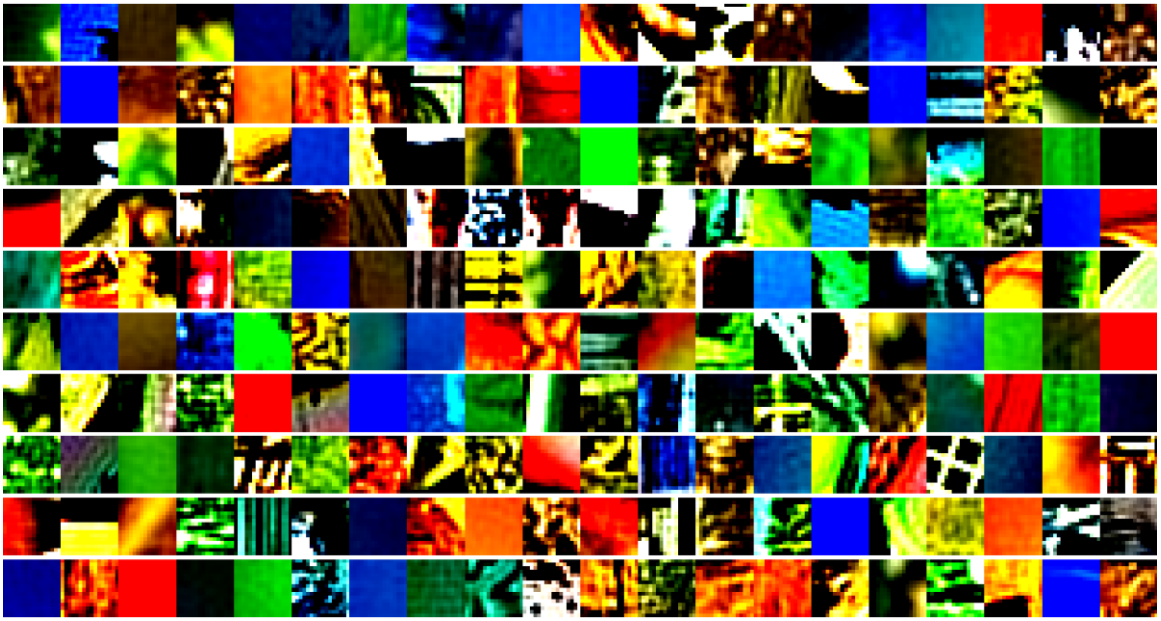

Figure 1: 200 randomly selected images in RGB format

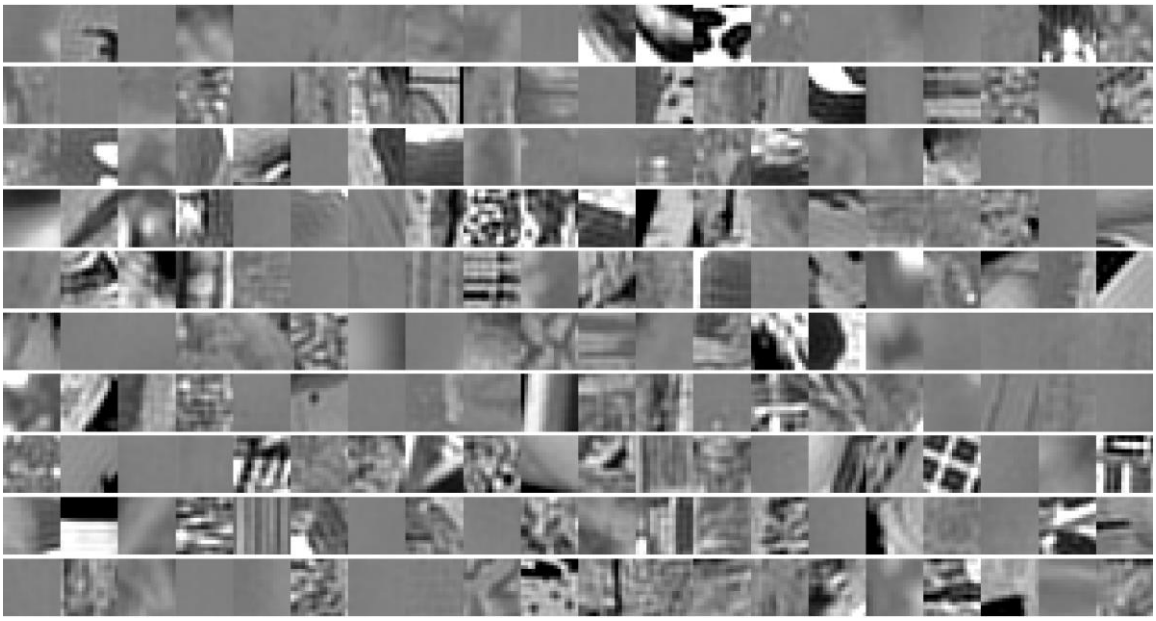

Figure 2: Grayscale versions of the randomly selected images

The cost function that we need to minimize is given in the prompt. The weights are initialized as recommended.

Grayscale images are more suitable as we want to extract the features of the images regardless of their colors. Colors would decrease generalisability and increase the training time to extract the same features.

Question 1.b

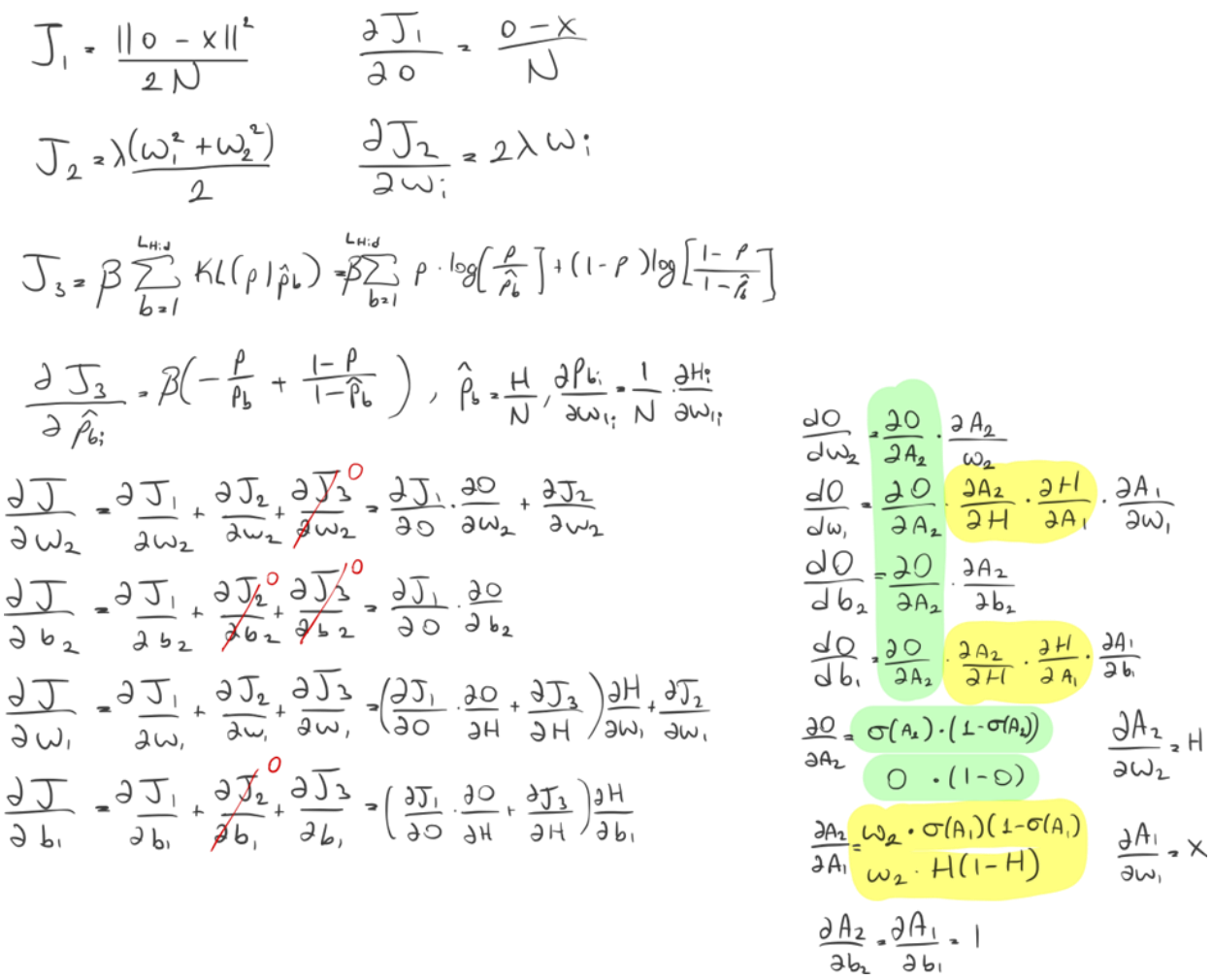

Figure 3: Derivative calculations for weights and biases

The gradients are calculated according to the derivations given in Figure 3. The code is written with an object-oriented approach. The class is structured differently from the recommendation, but the method names are self-descriptive.

The weights are initialized exactly as described in the prompt.

First and second stage weights are transpose of each other. When weight update is performed, the transpose of the gradient for the second weight is added to the first stage weight.

Sigmoid activation is used on both layers. The given loss function is minimized with the gradients. $\hat{p}_b$ is defined as follows:

$$\hat{p}_b = \frac{\sum_{n=0}^{N} H_n}{N}$$

Where H_n corresponds to single hidden units.

Mini-batch stochastic gradient descent with momentum is used to update the weights where several batches are executed in one epoch. After training the network several times, I chose the best-performing parameters by comparing the inputs and outputs.

The optimum values are found as follows for training 64 hidden layer network:

$$L_{hid} = 64$$

$$\lambda = 0.0005$$

$$\rho = 0.4$$

$$\beta = 0.005$$

$$\eta = 0.3$$

$$\alpha = 0.6$$

50 epoch with a batch size of 32

Question 1.c

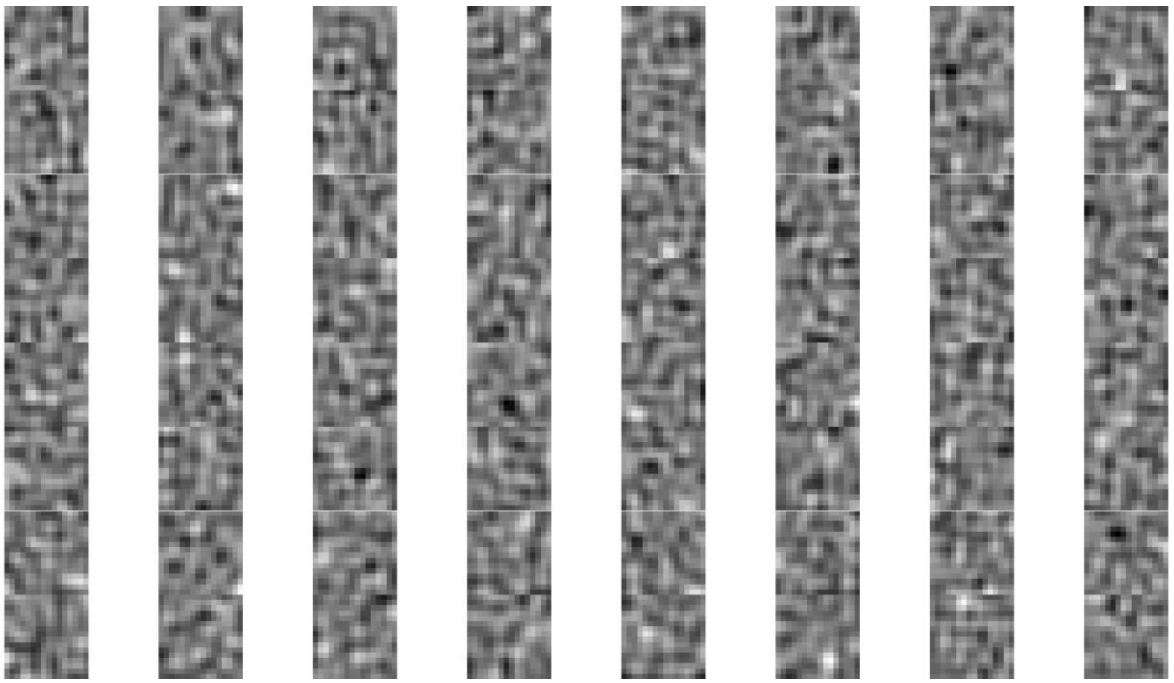

Figure 4: First layer weights plotted after training with parameters given in 1.b

I expected to see edges and more common features but instead found the images in Figure 4. The reason for these weights is probably overfitting. To prove my suspicion, I compared the inputs and outputs of the last batch of the last epoch and found Figures 5 and 6. They look nearly identical and close to perfect reconstruction. I did not try using unseen samples and comparing their output which would have been useful, to be sure. If this is due to overfitting, while the network can identically reconstruct the samples, it shouldn't be able to perform this well with an unseen sample.

Also, notice that some images in Figure 5 have tentackle-like parts that match the images in Figure 4. The auto-encoder might have extracted the small shapes and details instead of the edges, as edges are less common than flat gray and tentacle-like vein images in most of the randomly selected batches.

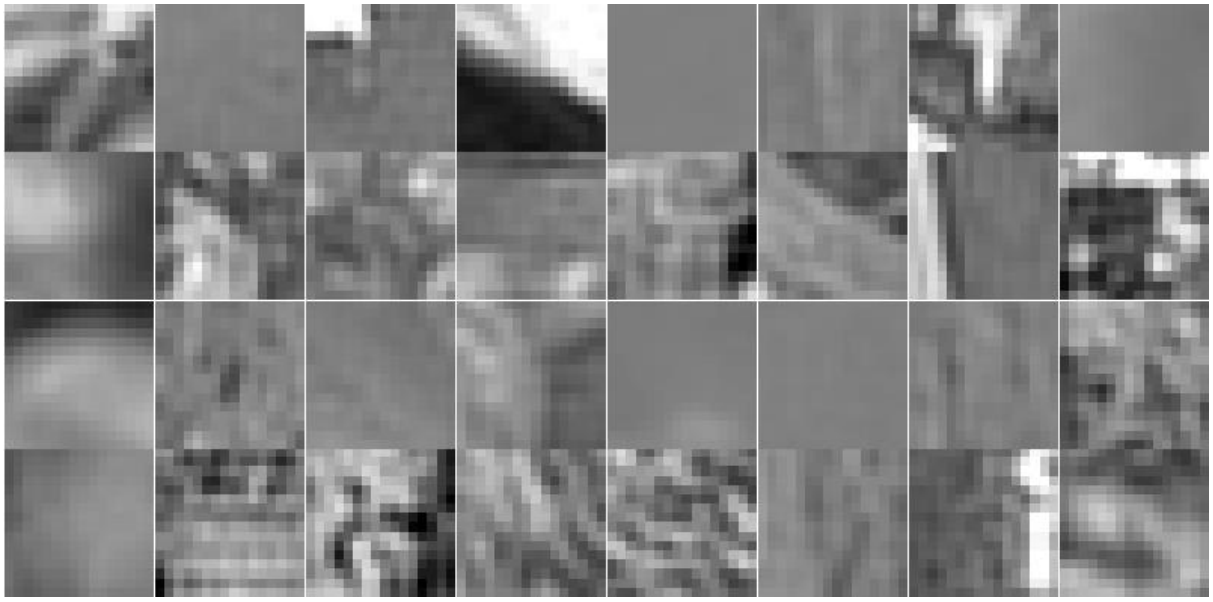

Figure 5: Random samples fed into the network

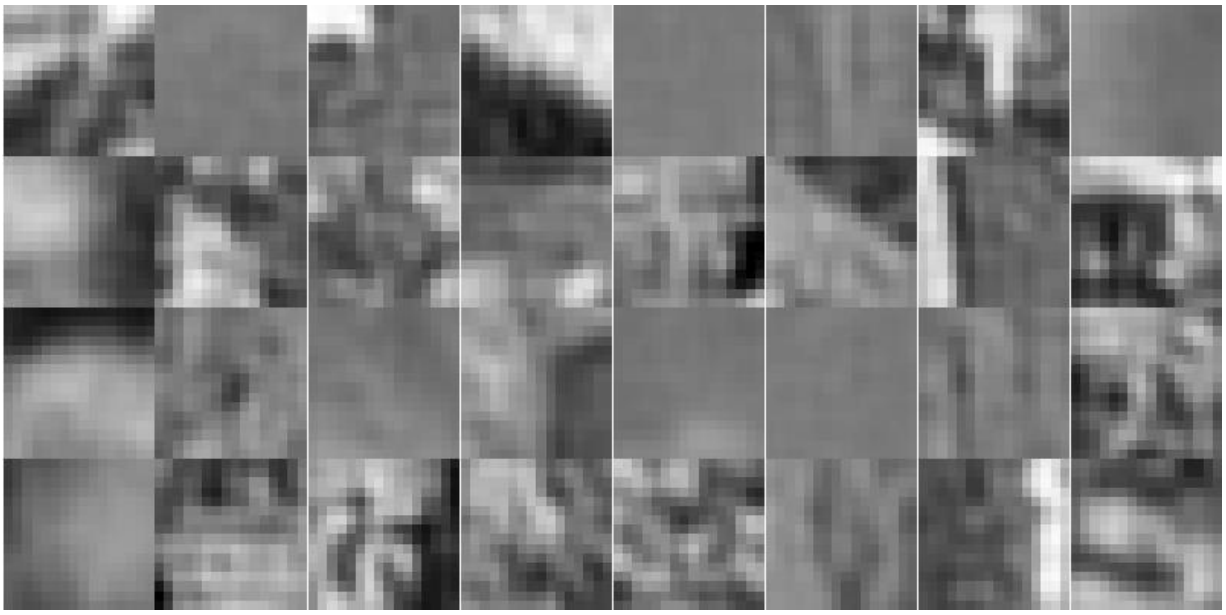

Figure 6: Reconstructed samples

Question 1.d

I have trained 9 networks with the combinations of the following parameters demonstrated on Figures 10 – 12.

$$L_{hid} \in \{16, 49, 81\}, \quad \lambda \in \{0, 3 \cdot 10^{-4}, 10^{-3}\}$$

I have noticed that with increasing L_{hid}, the weights become more detailed. When $L_{hid} = 16$, the extracted features are more thick and simple. When L_{hid} is increased further, the features become more vein-like, thin, and detailed.

I could not see the effect of λ on the weights itself, but I've seen them on the produced outputs. The outputs become less detailed and more blurred with the increased λ. To explain what I see, I trained a network $L_{hid} = 64, \lambda = 0.001$ in the following Figures 7-9.

The details are significantly less compared to outputs in Figure 6. As our task is not specified to create a general algorithm, I do not see overfitting as a problem in this case and prefer higher quality detailed images.

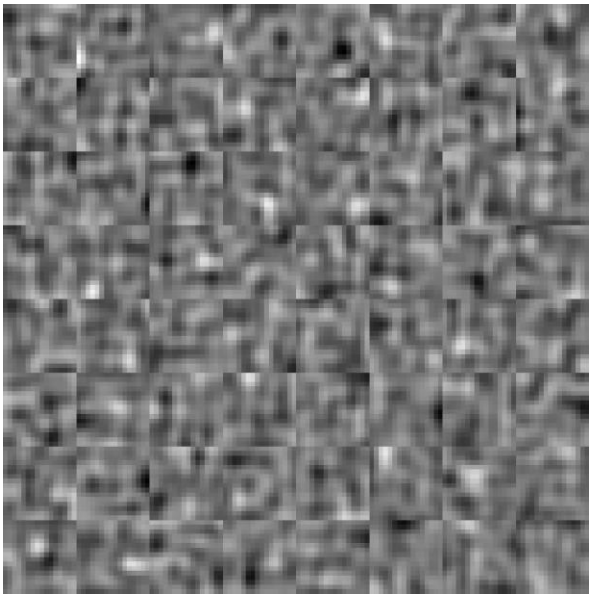

Figure 7: 64 Hidden layers, $\lambda = 0.001$

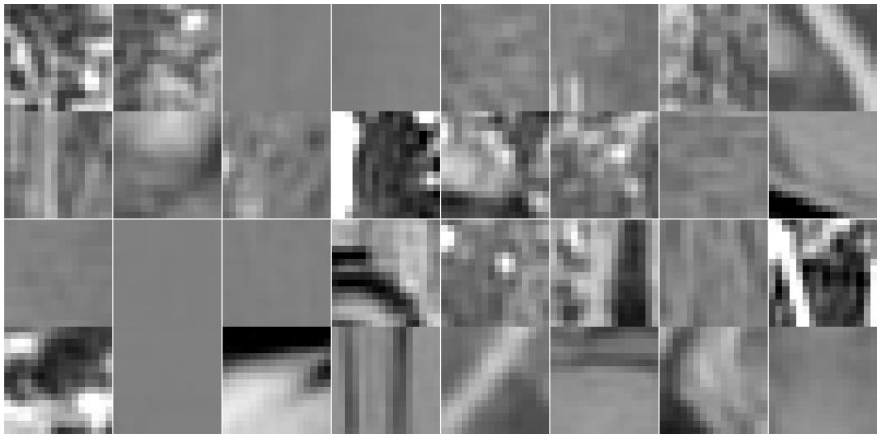

Figure 8: Input to the network

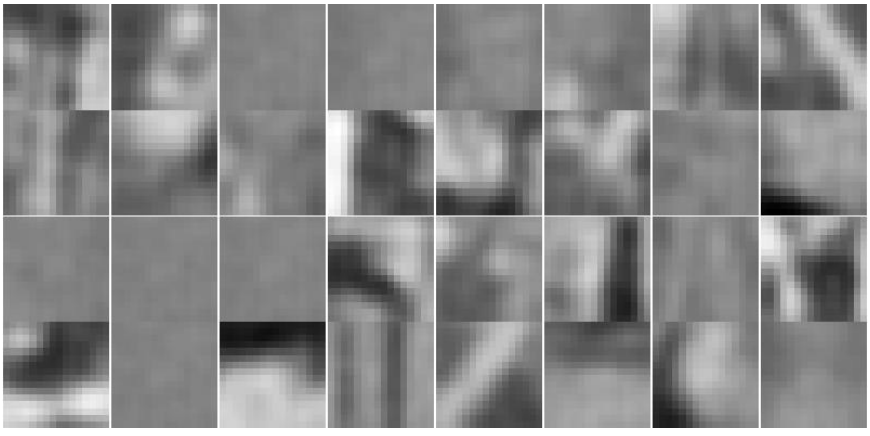

Figure 9: Output of the network

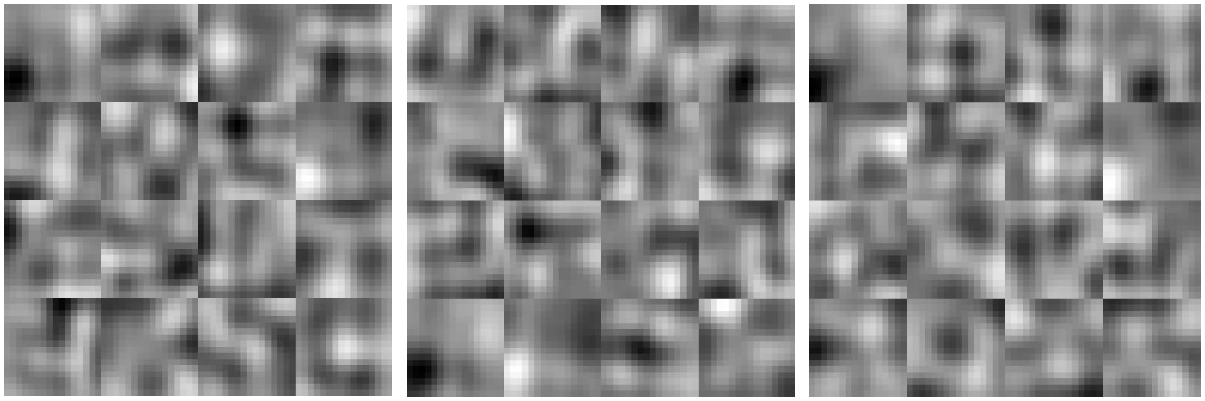

Figure 10: 16 Hidden layers, $\lambda = 0, 0.0003, 0.001$ *repectively from left to right*

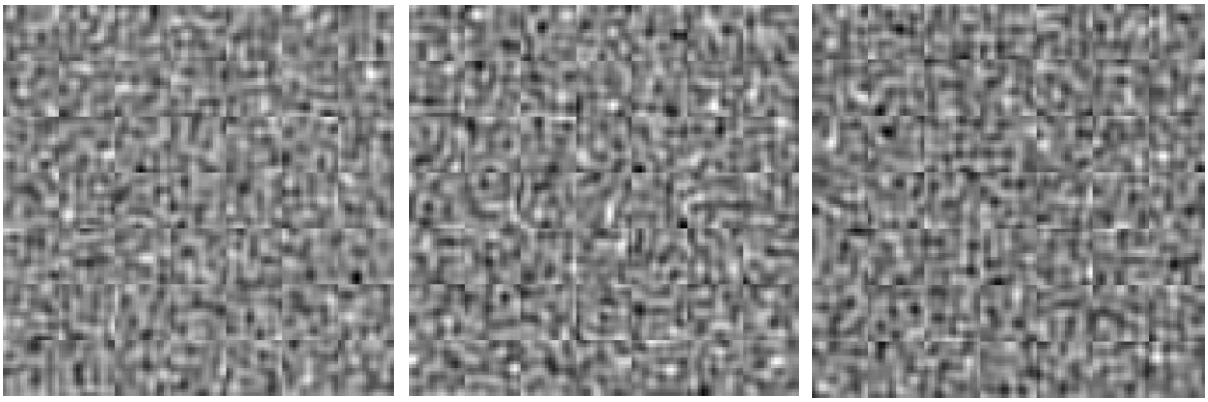

Figure 11: 49 Hidden layers, $\lambda = 0, 0.0003, 0.001$ *repectively from left to right*

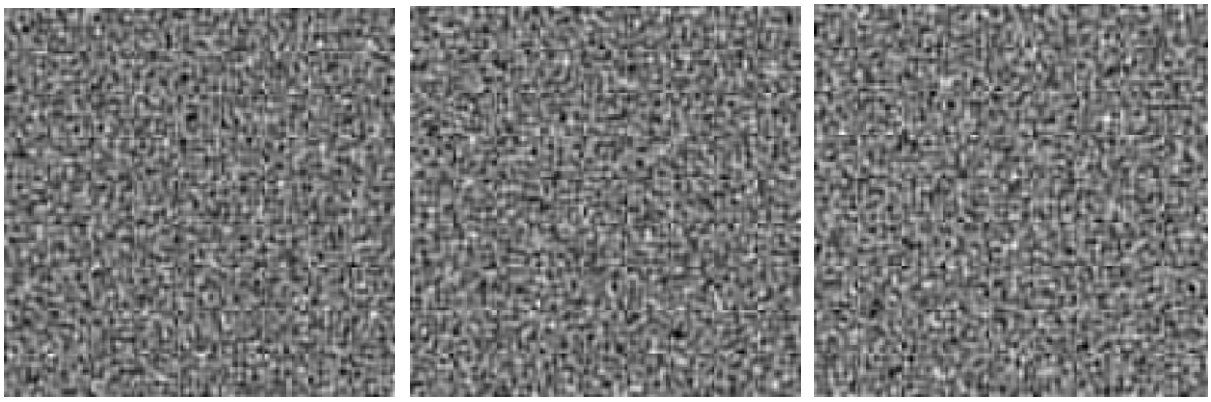

Figure 12: 81 Hidden layers, $\lambda = 0, 0.0003, 0.001$ *repectively, from left to right*

2. N-Gram NLP Model

- The network consists of 4 layers, including the input layer.
- The first layer is a linear transformation that maps the 250 x 1 inputs to D x 1 vector representations.
- Embeddings are then passed through different but P x D sized weights and mapped onto a hidden layer after passing through the sigmoid activation function. This is done as if the same matrix was to be used for three embeddings or if three embeddings were to be added together before being passed to the hidden layer, their location information would be lost, which is necessary in our case.
- The hidden layer is passed to the output layer after being transformed with a 250 x P weight matrix and passed through a softmax function. This gives the probability of the next word following tri-grams.

Gradients are calculated similarly to the first problem, the only differences being the existence of softmax function and cross-entropy loss. The derivative of the softmax function is given as follows:

$$\sigma_i = \frac{e^{x_i}}{\sum_j e^{x_j}}, \frac{\partial \sigma_k}{\partial x_i} = \sigma_k(\delta_{ik} - \sigma_i)$$

$$\frac{\partial E_x}{\partial o_j^x} = \frac{\partial}{\partial o_j^x}\left(-\sum_k |t_k^x \log(o_k^x)|\right) = -\frac{t_k^x}{o_k^x}$$

Note that the training is stopped based on the cross-entropy of the training data instead of validation data. The reason for this choice is explained in the following chapter along with the figures.

Question 2.a

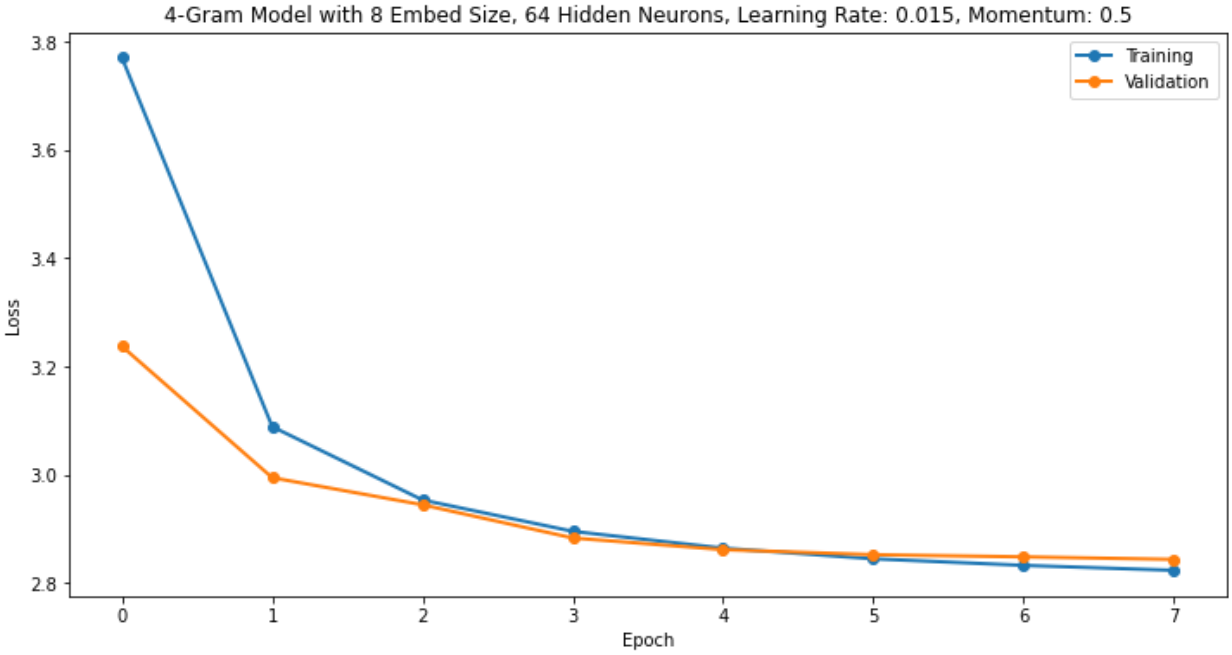

Figure 13: Training and test loss functions vs. epoch graph for (D, P) = (8, 64)

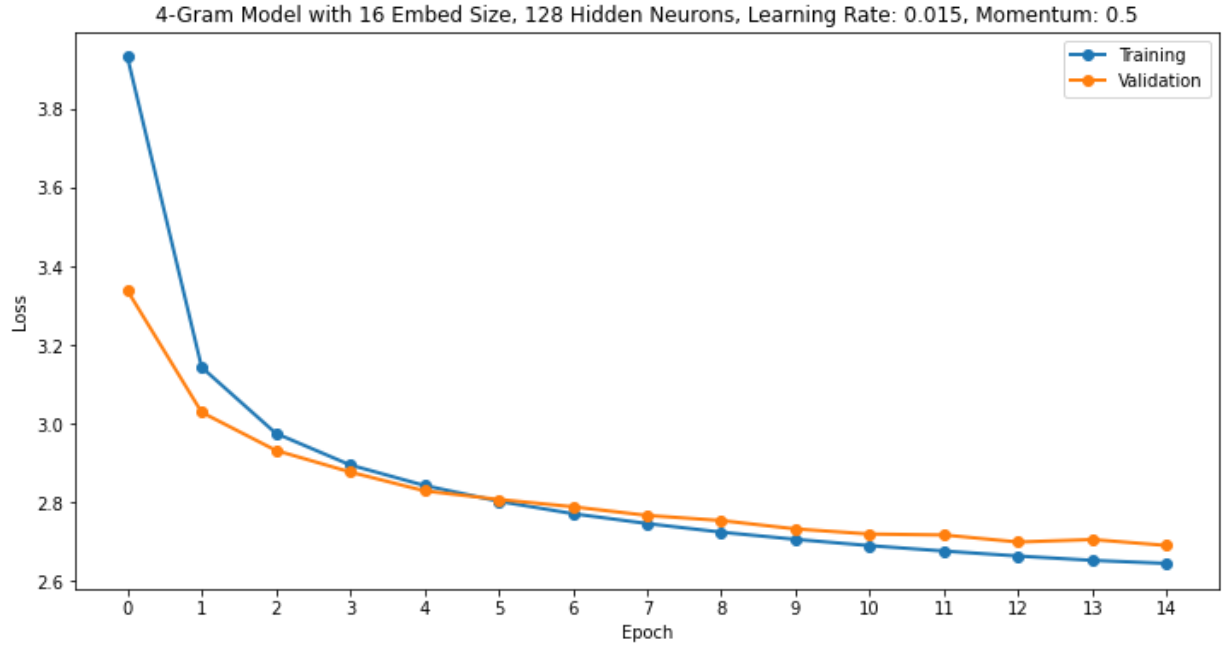

Figure 14: Training and test loss functions vs. epoch graph for (D, P) = (16, 128)

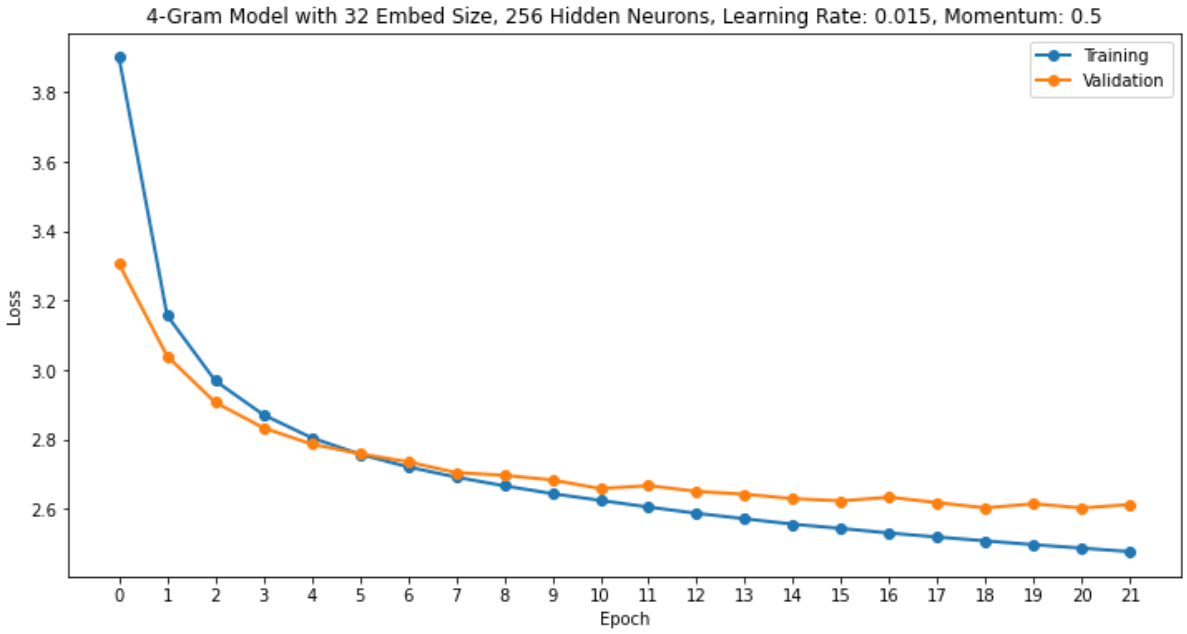

Figure 15: Training and test loss functions vs. epoch graph for (D, P) = (32, 256)

The reason for using training cross-entropy error is it is much smoother compared to validation cross-entropy error. Some spikes in the validation error make understanding the convergence point harder and require a more sophisticated algorithm. The training is stopped if the decrease in the loss function falls below a certain limit, 0.01 in our case.

This approach is better for (D, P) values (16, 128) and (8, 64) but extends the training on the (32, 256) model. Even though it extends the process, it is not a pure time waste as we can see that validation loss slowly decreases until the 18th epoch.

Higher the (D, P) values are, the lesser the final loss function becomes. They behave similarly in initial epochs, but those with smaller (D, P) values converge faster and to a higher loss value.

The downside of the higher (D, P) networks is their training time, as increased dimensionality significantly increases training time.

Question 2.b

The same 10 trigrams are used for estimating the 10 most likely words for each model in part a. Even though this is more elaborate than the prompt, I thought I could see the improvements better as it is possible for correct choice not to be in first then in worse models but see its improvement on different models. Also, most samples end with the '.' character, and as the samples are chosen randomly when the code is run, I wanted to decrease the chance of more than half of selected trigrams ending with punctuations.

Table 1: Output estimates for (D, P) = (8, 64)

Trigrams			Grounds	Estimates (Probability High -> Low)										
last	more	than	[a],	6th	you	that	it	?	the	a	i	this	children	two
just	do	that	[.],	1st	.	,	?	for	to	in	at	now	here	as
nt	have	that	[right],	8th	.	,	at	to	for	in	much	right	before	now
to	the	street	[in],	8th	.	,	?	to	of	's	i	in	and	we
did	the	best	[she],	>10	.	of	,	i	?	we	way	off	man	that
to	do	what	[he],	6th	you	they	we	she	i	he	's	the	it	?
a	new	world	[.],	1st	.	?	for	,	to	and	in	here	with	at
you	do	about	[that],	2nd	it	that	the	them	this	him	me	?	you	a
to	end	it	[.],	10th	's	was	is	has	did	does	could	will	would	.
it	's	against	[the],	1st	the	me	you	it	.	a	him	them	us	that

Table 2: Output estimates for (D, P) = (16, 128)

Trigrams			Grounds	Estimates (Probability High -> Low)										
last	more	than	[a],	4th	that	one	i	a	two	you	we	any	this	five
just	do	that	[.],	1st	.	?	,	for	in	to	at	now	all	today
nt	have	that	[right],	7th	.	,	in	at	before	now	right	today	all	?
to	the	street	[in],	8th	.	?	,	the	now	with	for	in	here	it
did	the	best	[she],	>10	.	i	we	,	you	of	they	part	time	for
to	do	what	[he],	6th	you	we	i	they	's	he	?	it	the	is
a	new	world	[.],	1st	.	for	,	?	to	at	in	and	with	before
you	do	about	[that],	2nd	it	that	this	them	?	.	him	the	what	,
to	end	it	[.],	1st	.	,	all	in	was	at	here	's	?	out
it	's	against	[the],	1st	the	it	me	us	him	them	you	that	.	this

Table 3: Output estimates for (D, P) = (32, 256)

Trigrams			Grounds	Estimates (Probability High -> Low)										
last	more	than	[a],	3rd	that	i	a	you	it	we	they	two	three	the
just	do	that	[.],	1st	.	,	?	for	to	in	now	here	you	the
nt	have	that	[right],	8th	.	,	at	much	in	here	before	right	last	today
to	the	street	[in],	5th	.	,	?	and	in	i	to	we	at	now
did	the	best	[she],	7th	i	we	.	you	that	they	she	he	if	man
to	do	what	[he],	6th	's	we	you	i	they	he	?	is	was	the
a	new	world	[.],	1st	.	,	?	to	for	at	in	is	that	with
you	do	about	[that],	2nd	it	that	this	them	the	my	your	him	what	money
to	end	it	[.],	1st	.	,	's	?	was	than	as	work	that	in
it	's	against	[the],	1st	the	you	us	me	my	him	them	what	it	your

As can be seen in the tables, with increased size and complexity, the network's overall performance is increased. Probability of the correct choice increases in order for more

complex models. The network predictions become more sensible with the increased (D, P) values.

Also, notice that the first two models can't guess the correct answer for the 5th question, but the word 'she' is in the top 10 estimates in the last model.

The n-gram language models usually suffer from the simple processing of the language. 3 words are not enough to guess the 4th model, and even human estimates do not have high accuracy with this task. The content is the most crucial element of the language, yet it is the most difficult to capture. For example, in the 6th sample, "to do what …", the algorithm has estimated 4 pronouns before the correct guess "he," which is also a pronoun. Classifying other pronoun answers as wrong is not completely correct in this case.

Apart from the shortcomings of the n-gram model, I think the network is successful, and estimates are meaningful.

3. RNN, LSTM & GRU Construction

The code is written in the same format for three architectures and trained with the same function. A layer class is present with forward and backward methods. The layer stores the variables and the input when the forward method is called. The layers are stored in a list and then called from last to first during gradient calculation. This is convenient as you do not need to return or feed a lot of variables in this case. You just feed in the data for forward propagation, the derivative of previous hidden and c layers for the backward propagation. A new addition is the tanh activation function, and it is implemented directly with NumPy. Its derivative is very similar to sigmoid, so it was not hard.

I have chosen to implement backpropagation through time truncation, as calculating the gradient without truncating makes the system less efficient, unstable, and harder to train.

Question 3.a

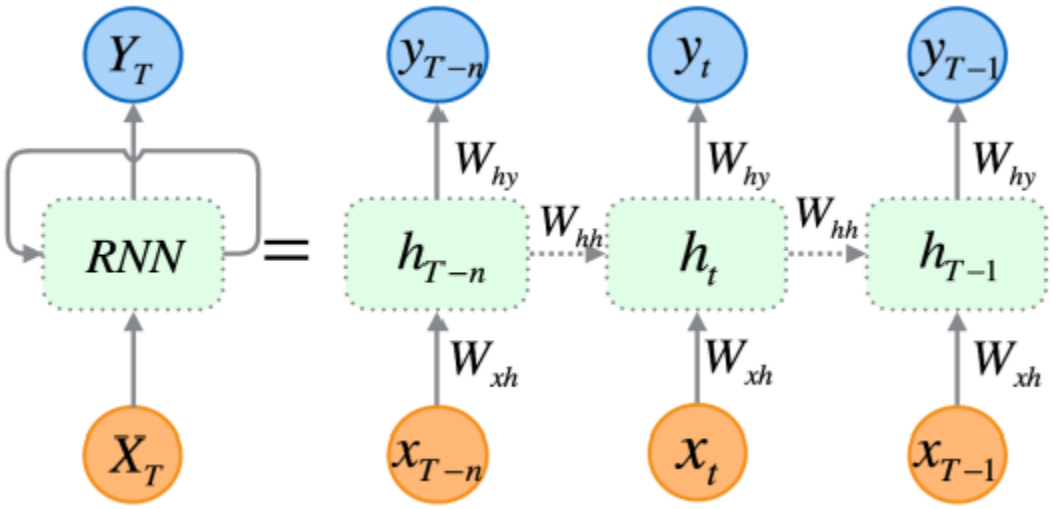

Figure 16: RNN cell architecture [1]

The forward propagation is done by applying the dot products given in Figure 16. The activation function is tanh for all forward propagations except for hidden to output as softmax activation is used for hidden to output on all recurrent networks in this question.

The reason for using training loss to terminate the training is more obvious if we compare the validation and training loss vs. epoch plots. In some cases, validation loss increases on two epochs in a row, and if I have implemented a termination algorithm based on the validation error, it will terminate at that point, but the accuracy increased around 5% after that point. The training terminates once the training loss increases as I have tuned the termination limit after several trials.

BPTT hyper-parameter is set to 10 as full backpropagation is both unnecessary and power/time-consuming. I have also experimented with different BPTT values, and it affects the final accuracy and loss of the model drastically. With a backpropagation truncation of 3, the network is trained significantly faster, but the accuracy increases slower and less. I did not analyze all BPTT values and trained the network for each value, but 10 provides a good balance between speed and performance.

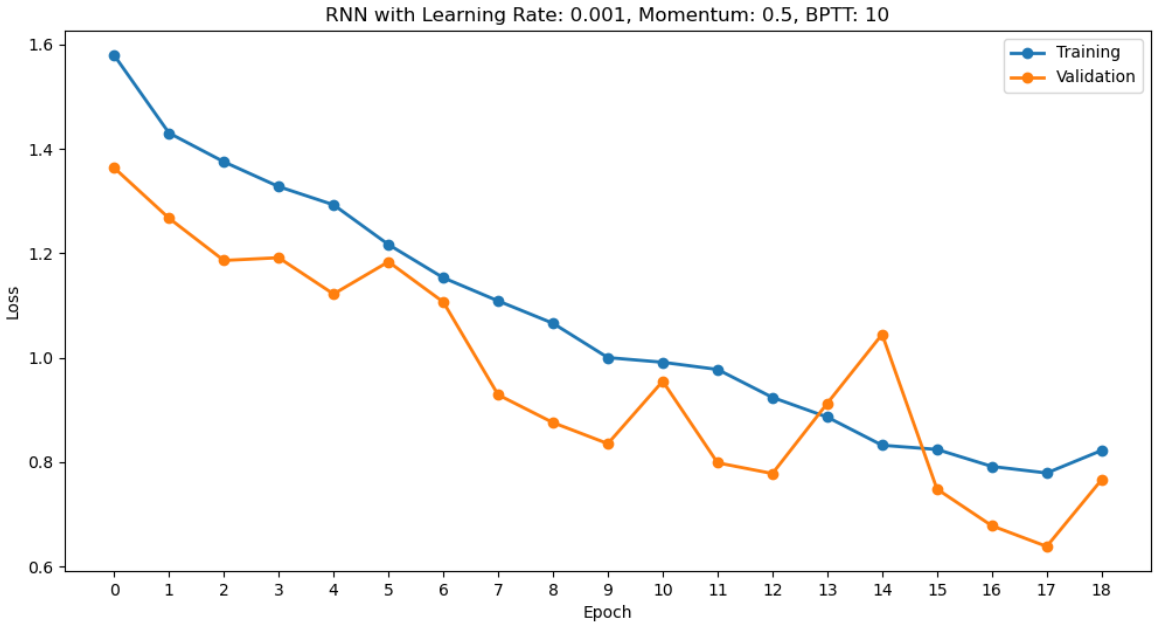

Figure 17: Training and validation loss vs. epoch for RNN

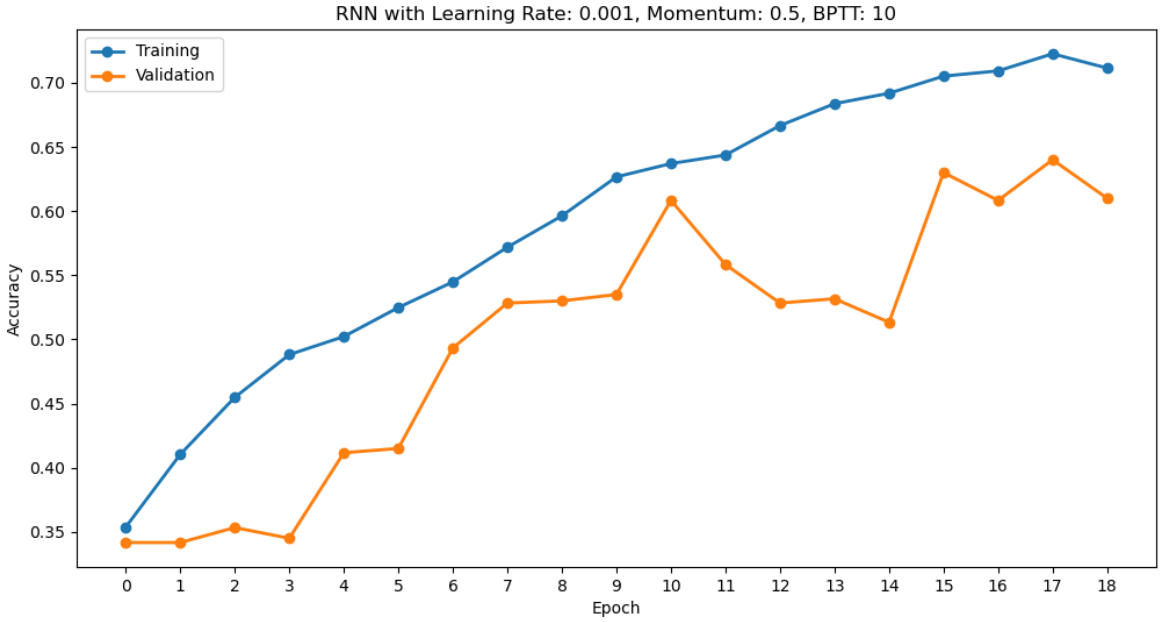

Figure 18: Training and test accuracy vs. epoch for RNN

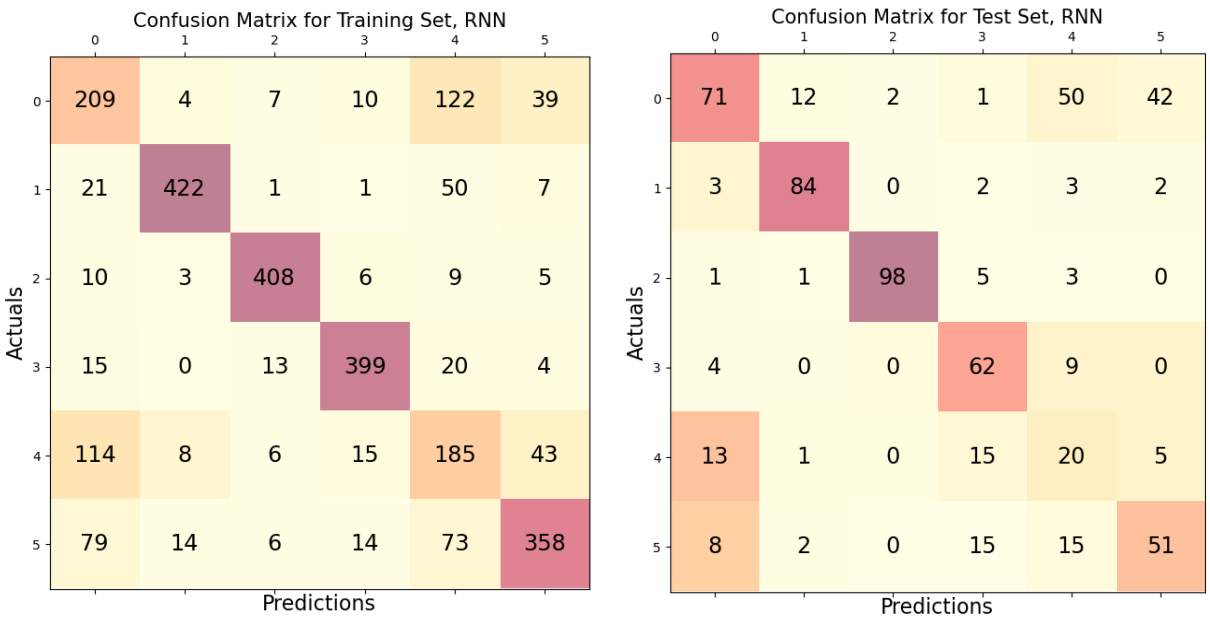

Figure 19: Training and test confusion matrices for RNN

Question 3.b

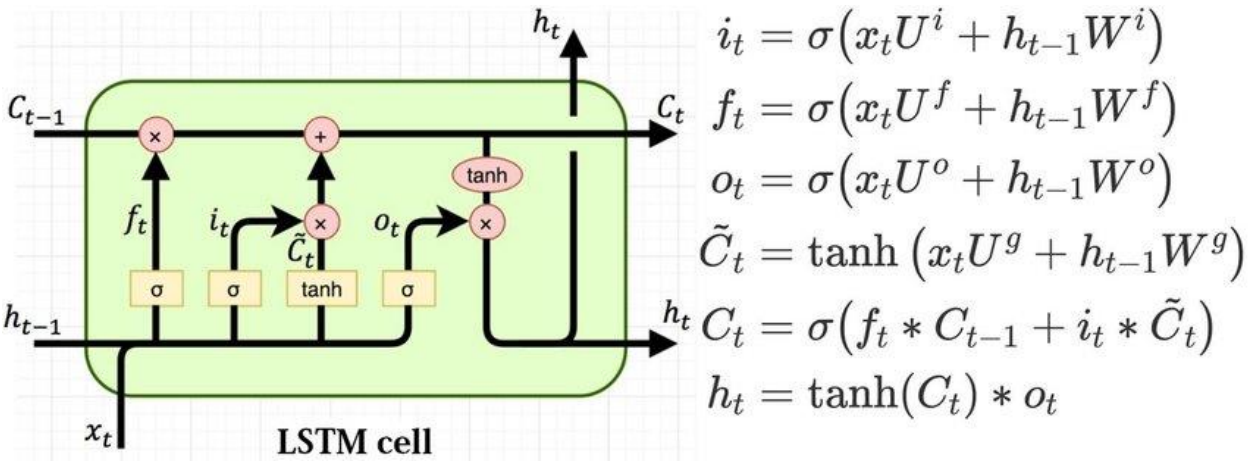

Figure 20: LSTM cell architecture [2]

Forward propagation is done as shown in Figure 20, the only difference being instead of using two separate matrices, U and W, I stacked H and X row-wise and named it Z and used single weights for each stage.

LSTM's BPTT is set to 4 as it was giving overflow errors. Decreasing the learning rate helped with this problem, but overflows occurred at some point. Also, LSTM is the slowest among the three networks, and with a BPTT of 10, it would require an unnecessary amount of time.

Interestingly, LSTM's accuracy rates turned out to be lower than RNN. Note that it required more epochs for RNN to get the same accuracy levels and twice more epochs to get the final loss value which is the same for LSTM and RNN. The difference might be due to different BPTT values. When I trained all of them with 3 BPTT, LSTM performed significantly better than RNN and converged at fewer epochs.

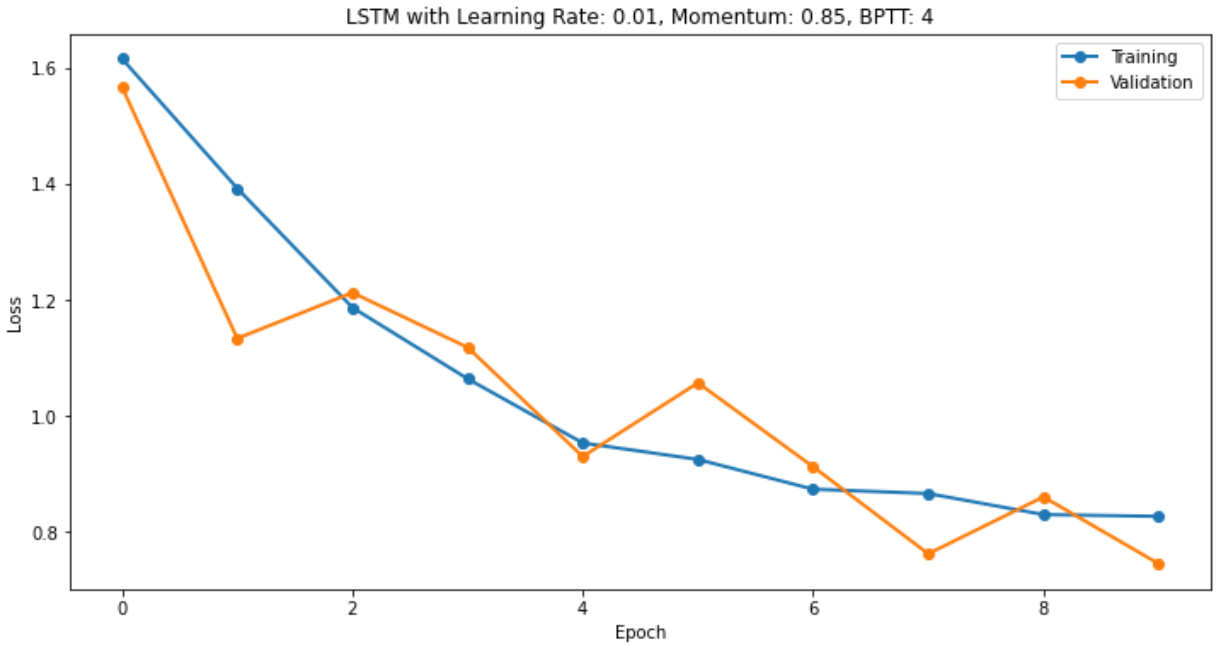

Figure 21: Training and validation loss vs. epoch for LSTM

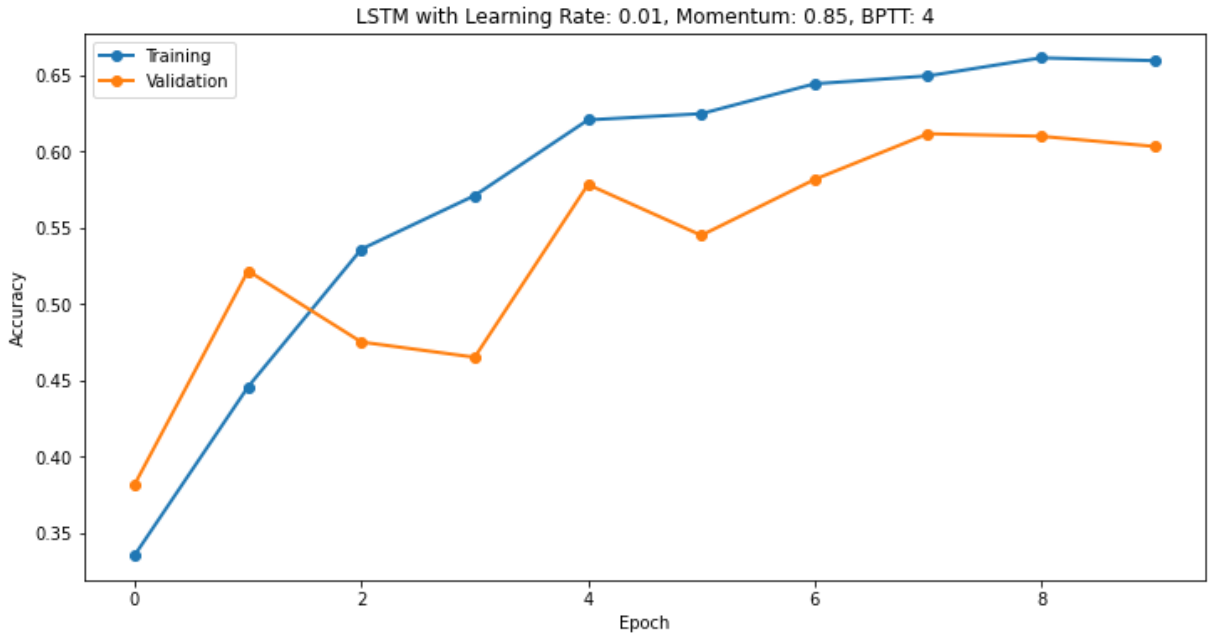

Figure 22: Training and test accuracy vs. epoch for LSTM

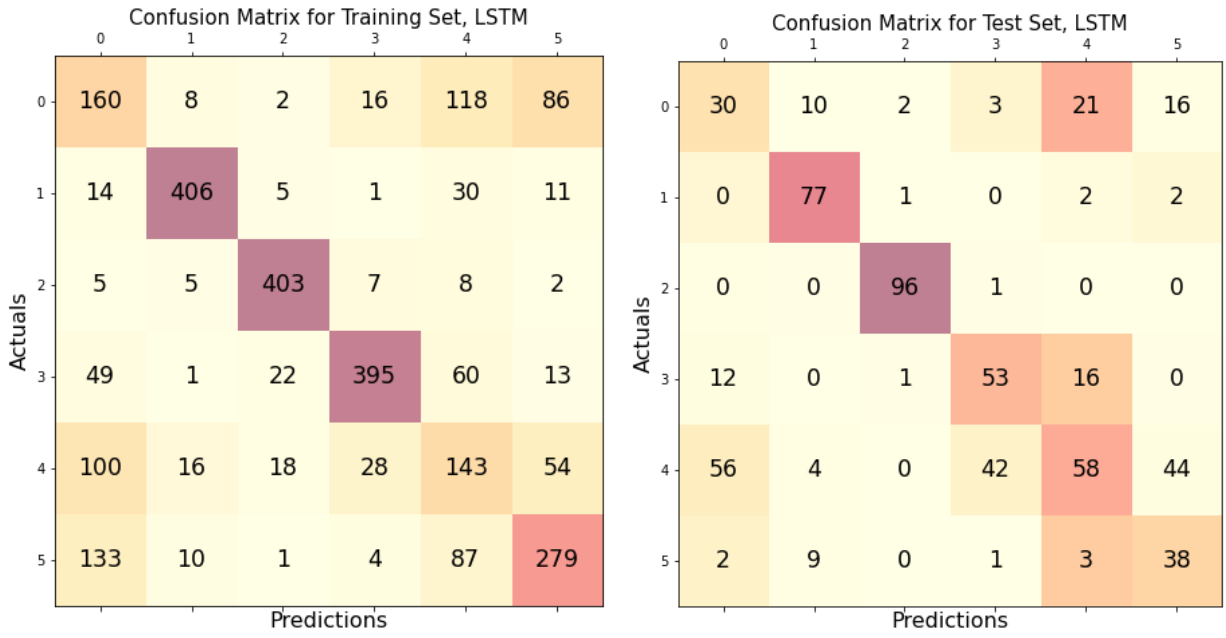

Figure 23: Training and test confusion matrices for LSTM

Question 3.c

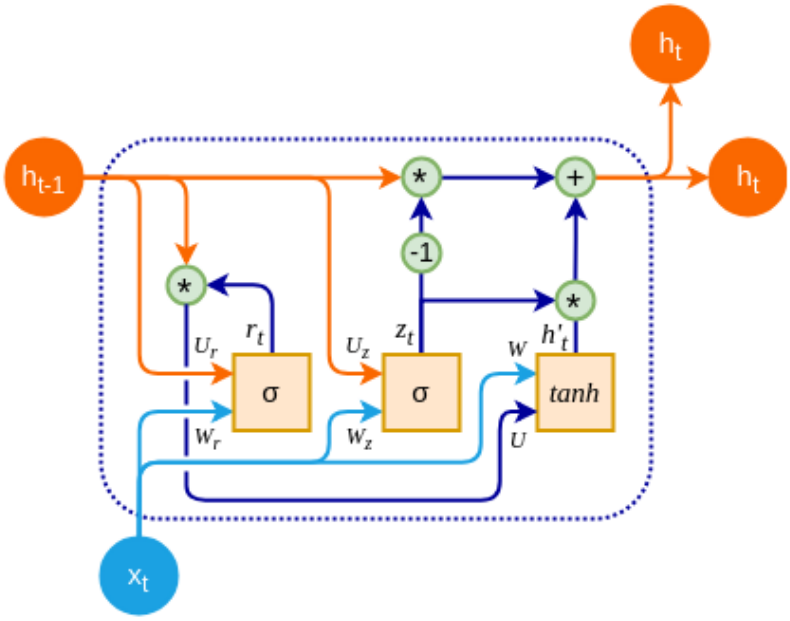

Figure 24: GRU cell architecture [3]

While implementing GRU, I have used the knowledge I have gained while writing the code for the previous two recurrent networks and used Figure 24 as a reference. The stages are similar to LSTM, so I calculated the forward and backward propagation in a similar way.

GRU performed the best and trained in the least amount of time and epochs. It seems that GRU is unparalleled by its rivals for this task. It achieved over 80% accuracy on the test set in 7 complete epochs. GRU stands in the middle of RNN and LSTM at the speed of completing one cycle. It has the best-looking confusion matrix among all 3 recurrent models.

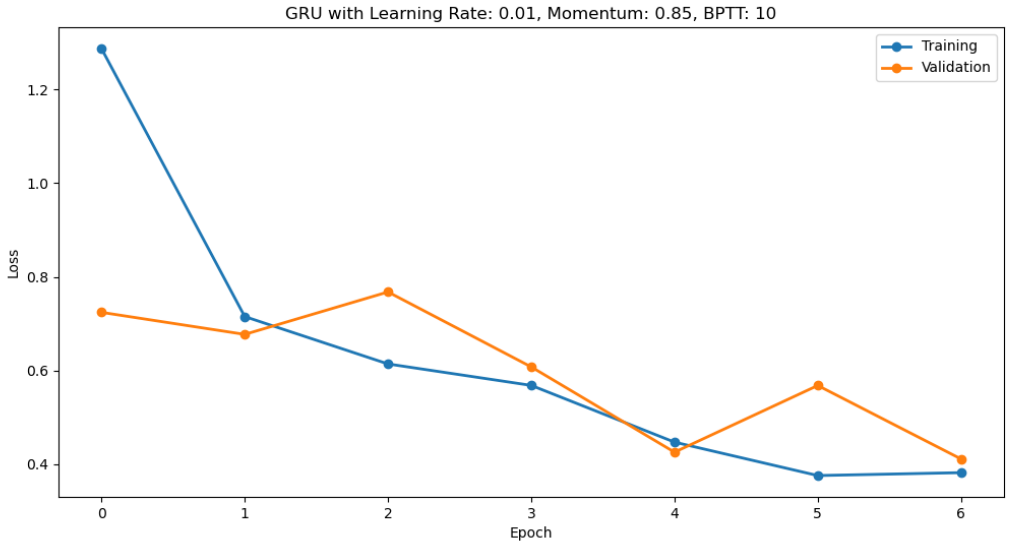

Figure 25: Training and validation loss vs. epoch for GRU

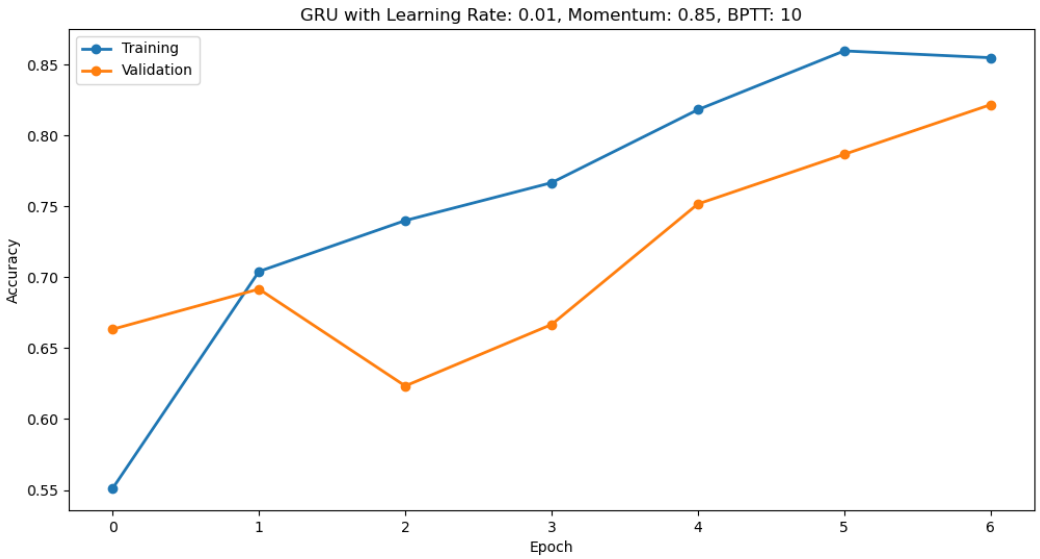

Figure 26: Training and test accuracy vs. epoch for GRU

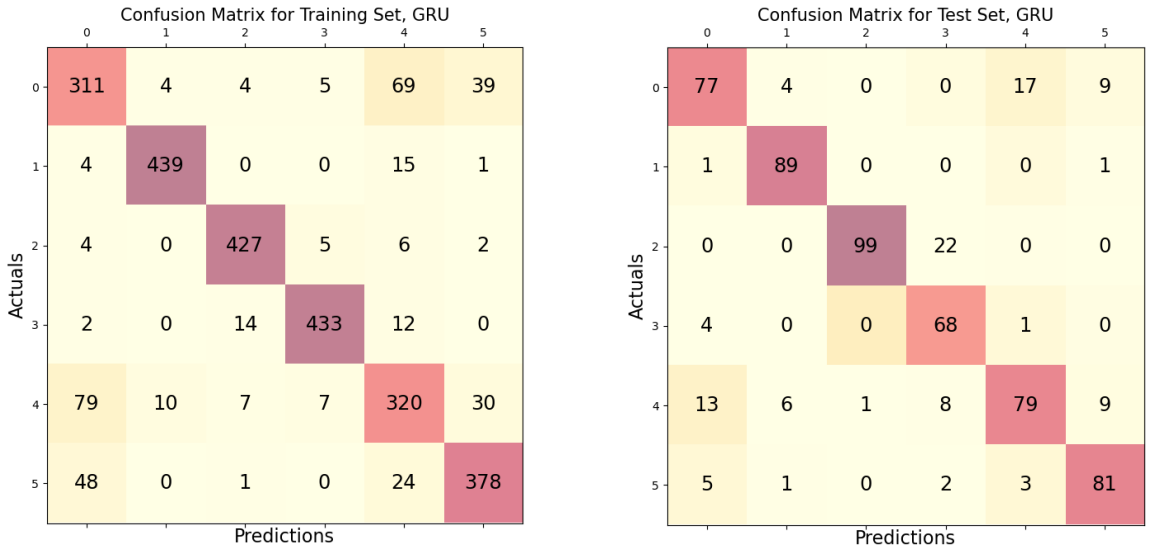

Figure 27: Training and test confusion matrices for GRU

4. References

[1] "What is the difference between Bidirectional RNN and RNN?" *i2tutorials,* Sep. 2019. [Online]. Available at: https://www.i2tutorials.com/what-is-the-difference-between-bidirectional-rnn-and-rnn/. [Accessed: May 8, 2022].

[2] S. Varsamopoulos et al. "Designing neural network based decoders for surface codes." 2018.

[3] "Gated recurrent units" *O'reilly.* [Online]. Available at: https://www.oreilly.com/library/view/python-deep-learning/9781789348460/88bbd930-3e25-47ab-9a50-b7d8c324994f.xhtml. [Accessed: May 8, 2022]