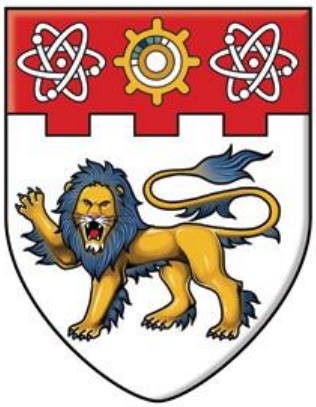

NANYANG TECHNOLOGICAL UNIVERSITY

Project 2 Report

CZ4042: Neural Network & Deep Learning

Student Name: Chua Wen Kai

Matriculation Number: U1722307B

Student Name: Chew Jing Wei

Matriculation Number: U1720459J

1 Table of Contents

2 Part A: Object Recognition Problem

2.1 Introduction

The CIFAR-10 dataset is a dataset of RGB images with size 32x32 and 10 categories in total. It is commonly used to benchmark object recognition models in the field of Computer Vision.

The experiment explores the visualization of the feature maps from each layer, followed by finding the optimal number of filters for each convolutional layer, and finally using different types of optimizers for the training and comparing their relative performances.

Training is done on 10000 random images and testing is done on 2000 random images from the CIFAR-10 dataset.

2.2 Methods

The original model was designed trained using 3000 epochs. For plotting training cost and testing accuracy, the cost and accuracy at each epoch is computed and stored in a list so that it can be shown on a graphical figure later.

For the visualization of the feature maps, the output of the tensors corresponding with each layer is kept in their own separate variables. As there are numerous filters in each convolutional layer and thus numerous channels, the output of each channel must be displayed separately as a grayscale image as a subplot. The output from each channel in the layer can then be plotted together as an overall figure with multiple subplots.

To compare the results from the grid search, it was decided that the different configurations will be compared based on the maximum test accuracy it can achieve within a certain number of epochs.

An attempt to create a grid of parameters and train each model using the original epoch count of 3000 at each of these combinations was attempted. However, as this took too long, more economical approaches were then devised.

One approach explored was to repeat the experiment using a much smaller number of epochs. This reduced number was chosen through the below process:

1. Performing the above experiment on a small subset of the parameters to try
2. Truncating the test accuracy results at a smaller epoch count
3. Sort the results from each configuration and compare if the top five best performing combinations from the truncated results is the same as the top five best performing combinations chosen based on the original set of results.

Through this process, it was decided that 2000 epochs was enough for each model to be trained adequately without affecting the comparison process.

Also, to reduce the complexity further, a recursive grid search was devised. This works by starting with a large range and a large step size, finding the neighbourhood in which the optimal number of filters for each layer is found, before reducing the step size and narrowing the search around that neighbourhood.

During the experiment, it was found out that the original number of filters to use for each layer is too small, as each search iteration constantly returned the maximum number of filters devised for each layer. Therefore, the grid had to be expanded until the test accuracy started to decrease, before continuing with the original plan to narrow down the number range to search on.

Once the optimal number of filters is found, the network is then modified to train using either momentum, RMSProp or the Adam optimizer based on a parameter. The option to introduce dropout in the model after the fully connected layer is also

introduced. The training loss and test accuracy from each variant is then stored as a variable and then plotted onto the same figure.

Finally, to compare the accuracy of all the models, the same strategy of comparing them based on the maximum test accuracy each of the models can achieve is used.

Their performances are then compared based on the number of parameters in the model, which will give us the relative time complexity of running a forward pass through the model, on top of the maximum test accuracy.

2.3 Experiment & Results

2.3.1 Question 1

Design a convolutional neural network consisting of:

• An Input layer of 3x32x32 dimensions

• A convolution layer $C1$ of 50 filters of window size 9x9, VALID padding, and ReLU neurons. A max pooling layer $S1$ with a pooling window of size 2x2, with stride = 2 and padding = 'VALID'.

• A convolution layer $C2$ of 60 filters of window size 5x5, VALID padding, and ReLU neurons. A max pooling layer $S2$with a pooling window of size 2x2, with stride = 2 and padding = 'VALID'.

• A fully connected layer $F3$ of size 300.

• A softmax layer $F4$ of size 10.

Train the network by using mini-batch gradient descent learning. Set batch size =128, and learning rate α=0.001. Images should be scaled.

2.3.1.1 a. Plot the training cost and the test accuracy against learning epochs.

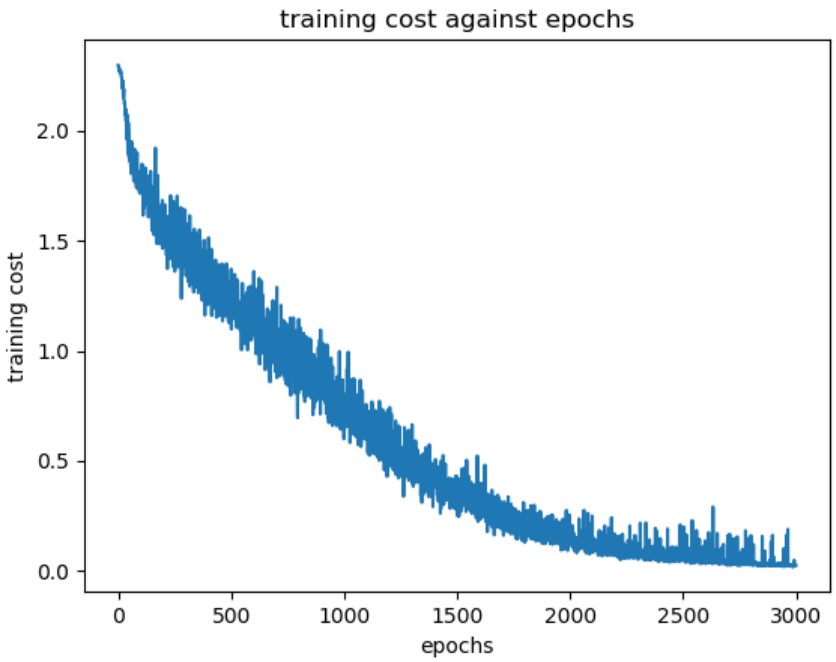

Figure 1.1: Graph of training cost against learning epochs

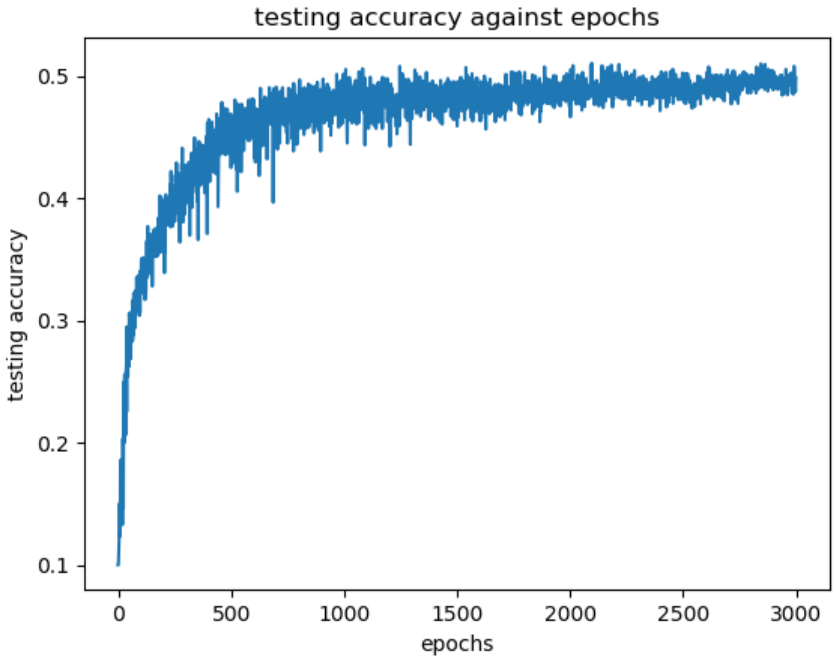

Figure 1.2: Graph of testing accuracy against learning epochs

2.3.1.2 b. For any two test patterns, plot the feature maps at both convolution layers ($C1$ and $C2$) and pooling layers ($S1$ and $S2$) along with the test patterns.

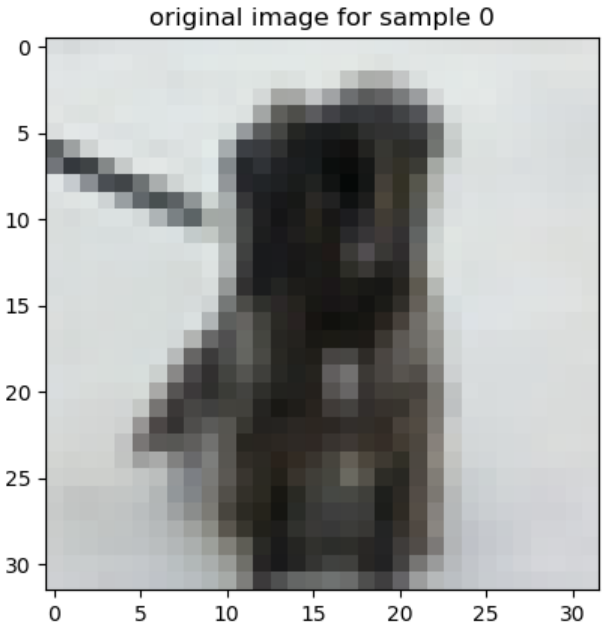

Figure 1.3: The original image of the first sample to be fed, belonging to a dog.

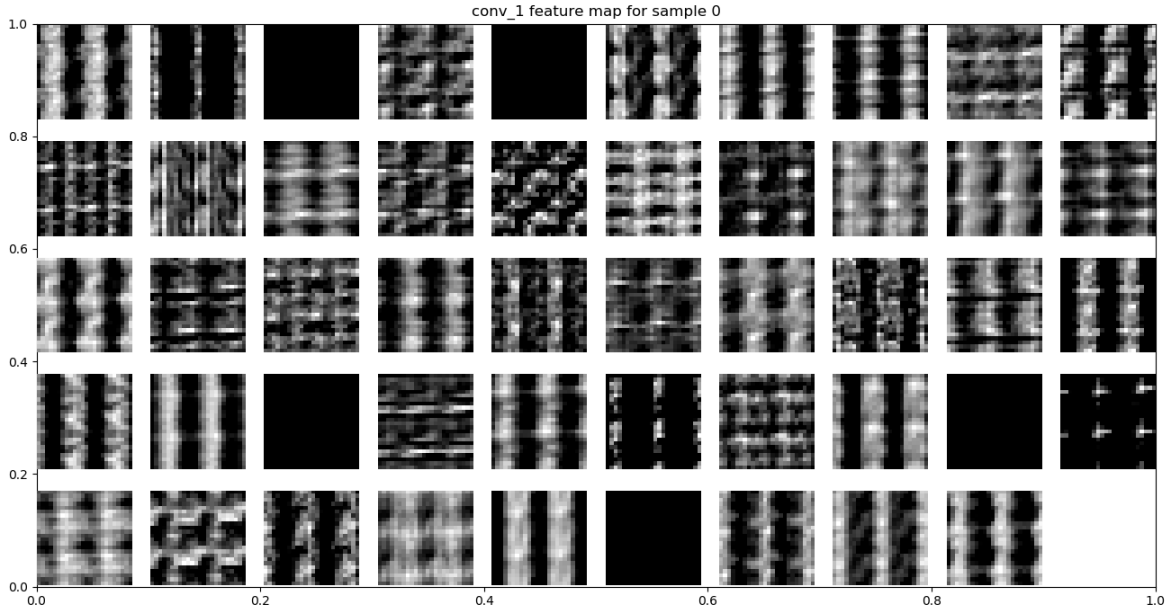

Figure 1.4: The feature map from the first convolutional layer after feeding the first sample image

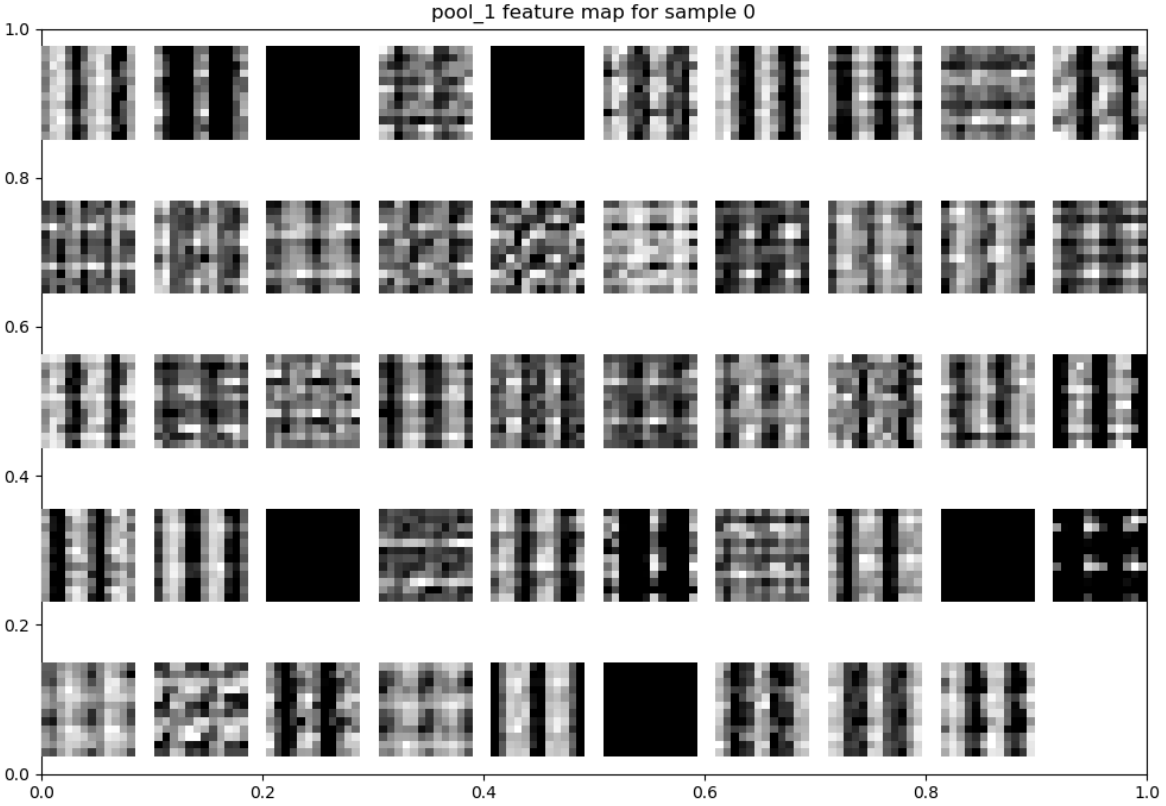

Figure 1.5: The feature map from the first pooling layer after feeding the first sample image

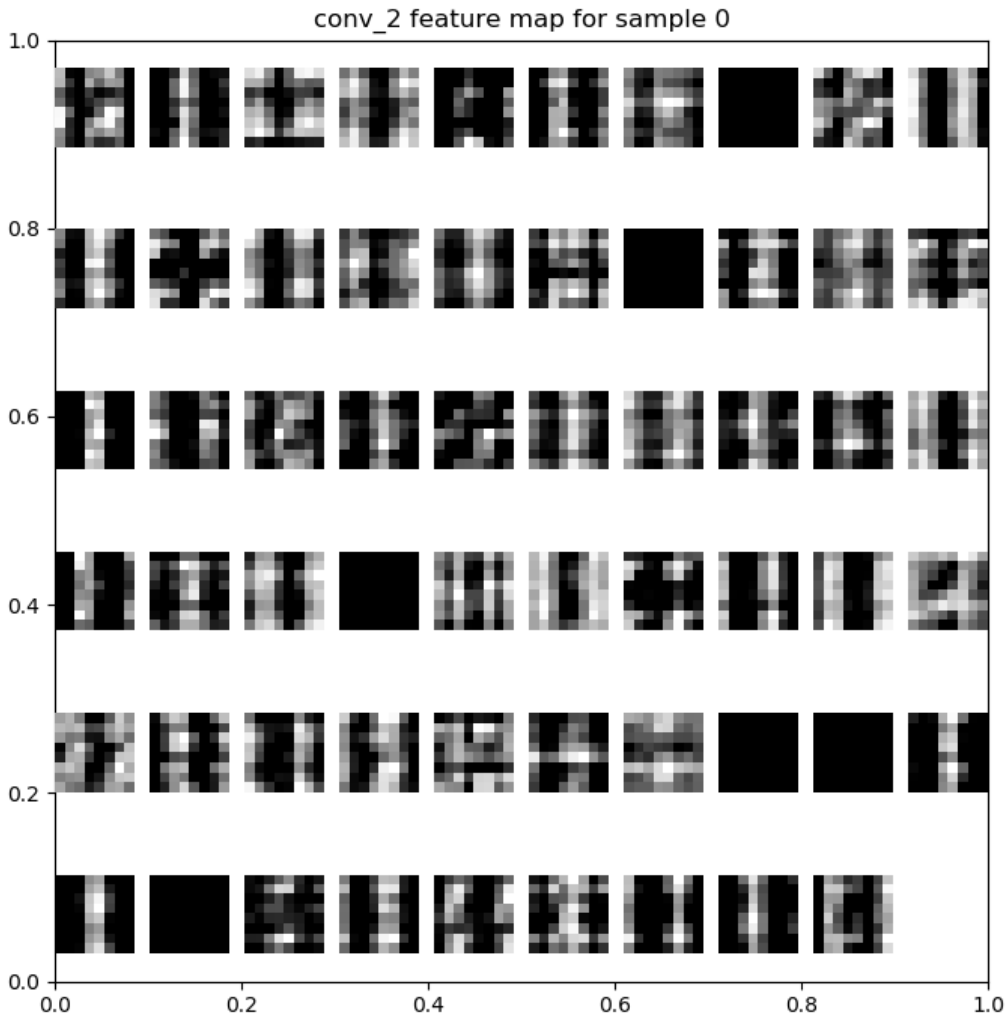

Figure 1.6: The feature map from the second convolutional layer after feeding the first sample image

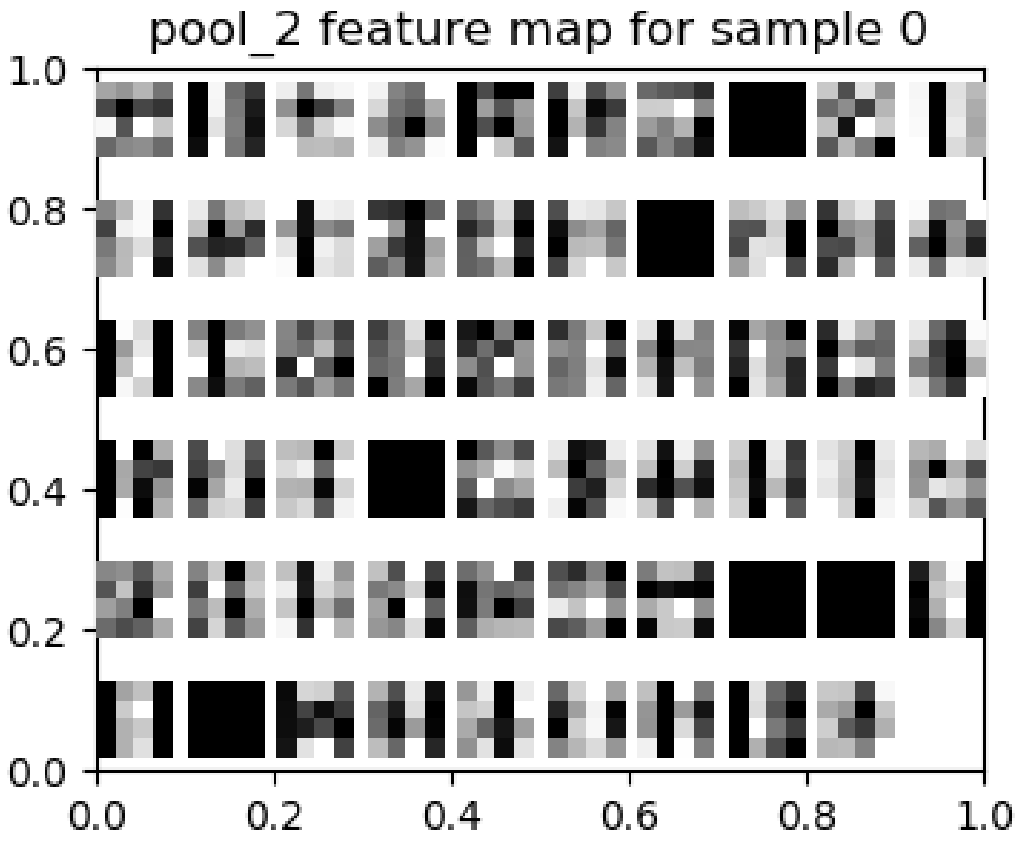

Figure 1.7: The feature map from the second pooling layer after feeding the first sample image

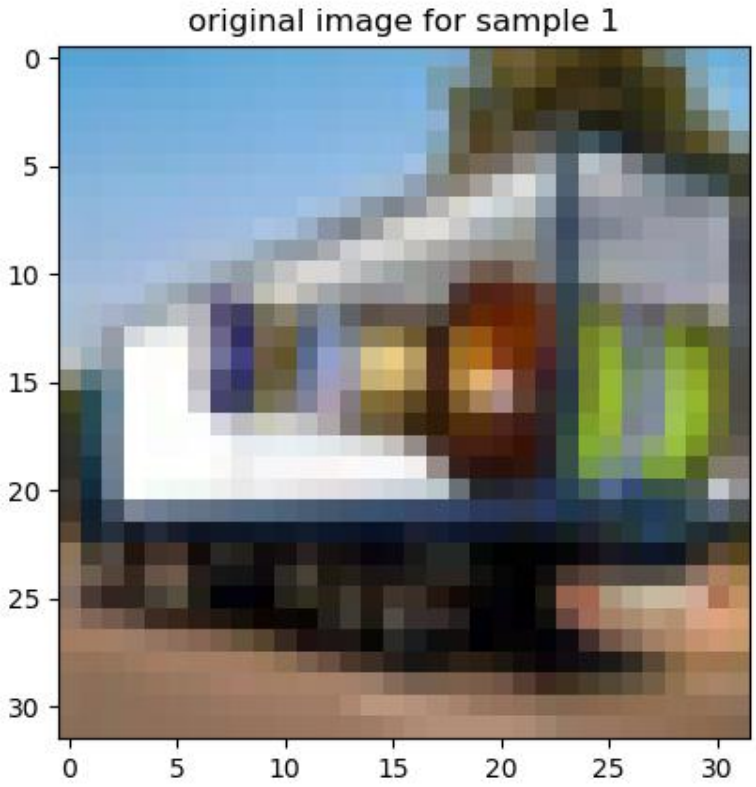

Figure 1.8: The original image of the second sample to be fed, belonging to a truck.

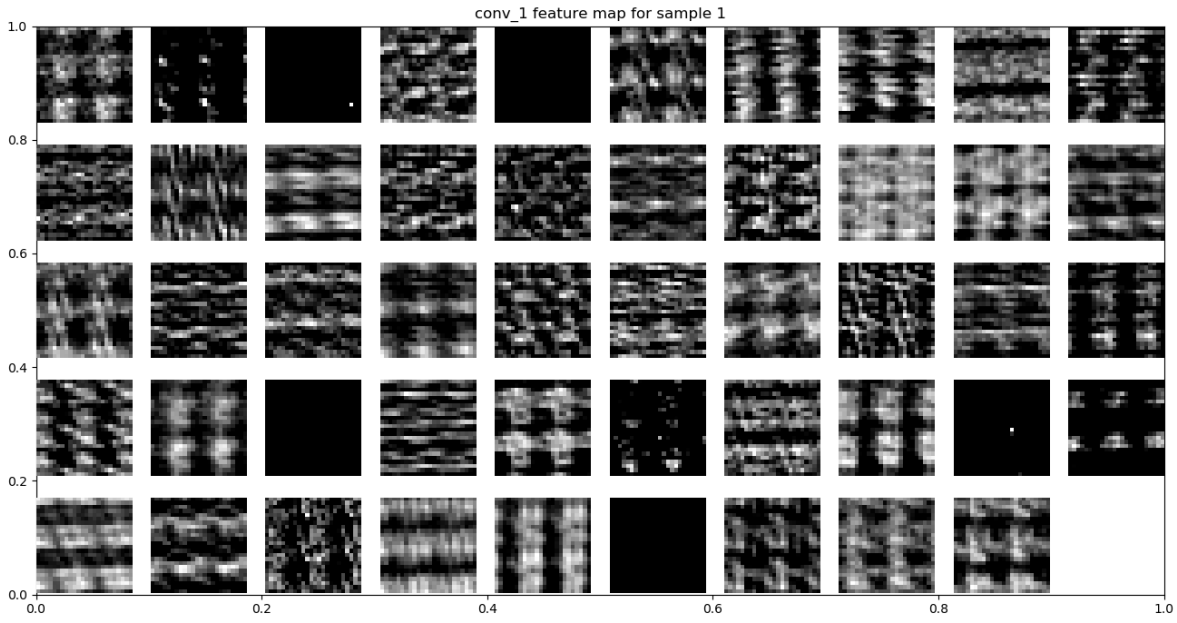

Figure 1.9: The feature map from the first convolutional layer after feeding the second sample image

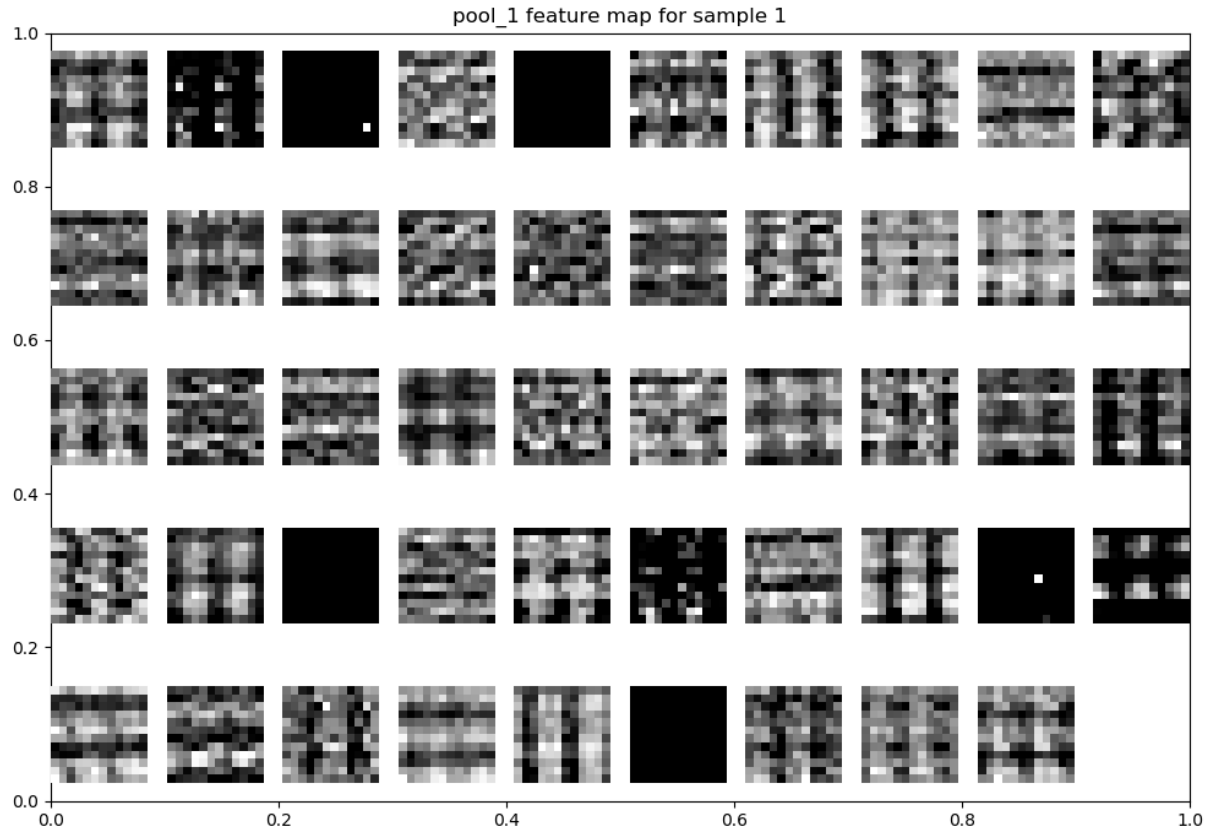

Figure 1.10: The feature map from the first pooling layer after feeding the second sample image

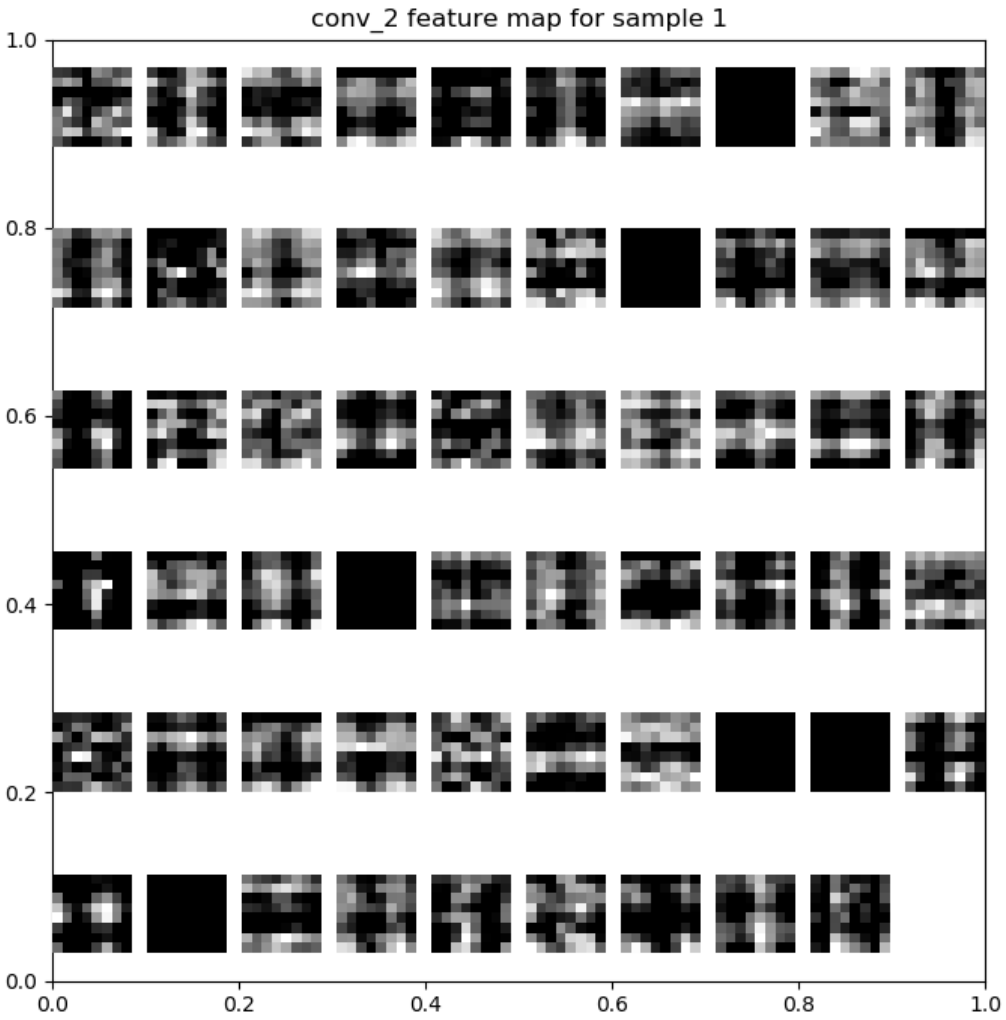

Figure 1.11: The feature map from the second convolutional layer after feeding the second sample image

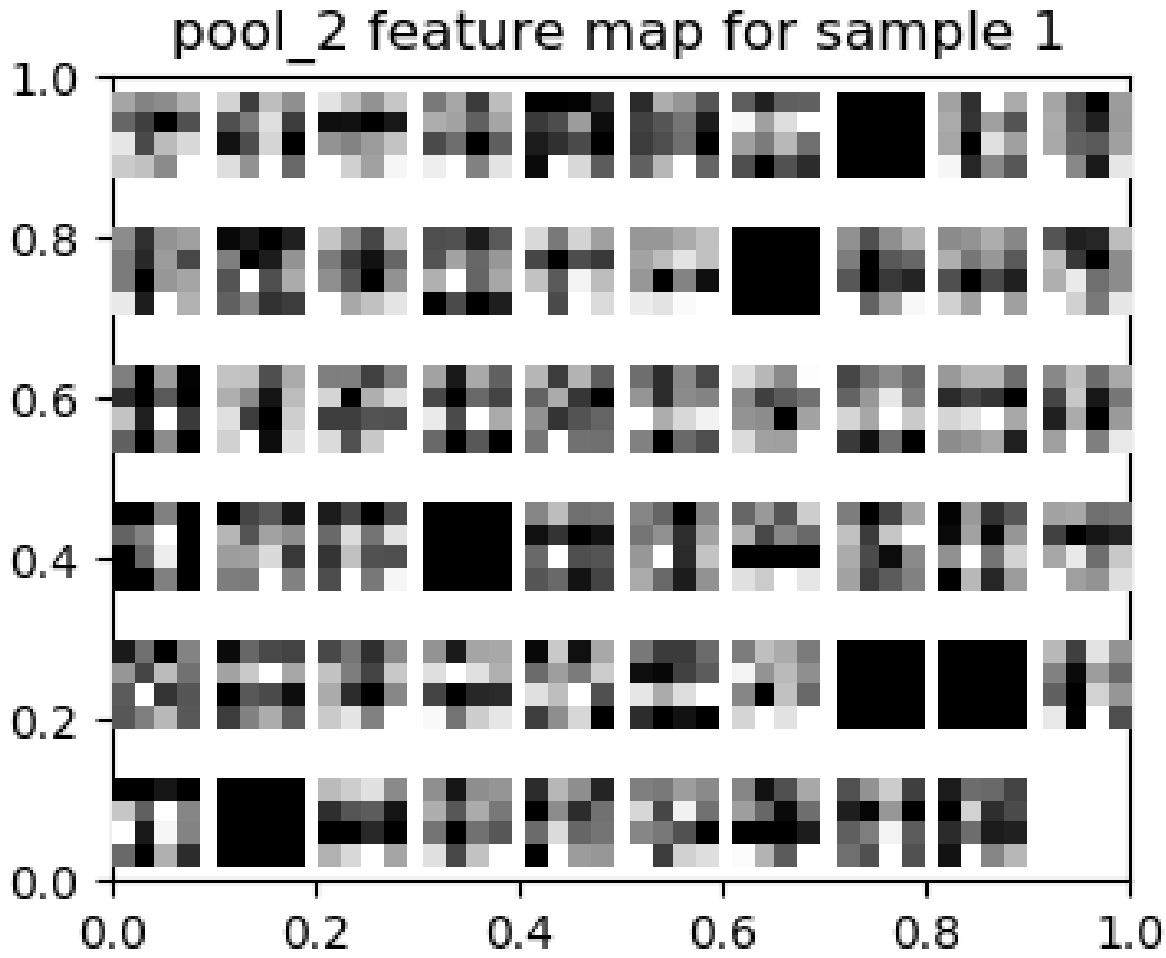

Figure 1.12: The feature map from the second pooling layer after feeding the second sample image

2.3.2 Question 2

Using a grid search, find the optimal numbers of feature maps for part (1) at the convolution layers. Use the test accuracy to determine the optimal number of feature maps.

Each of these tables show the maximum test accuracy achieved by that combination.

Interestingly, through the different combinations, it is noticed that increasing the number of filters in C1 had a bigger effect on test accuracy than increasing the number of filters in C2.

First iteration:

Step size = 36

Number of filters in C1 to try = [16, 52, 88]

Number of filters in C2 to try = [16, 52, 88]

	Number of filters in C2			
Number of filters in C1		16	52	88
	16	0.48800	0.49500	0.49500
	52	0.50350	0.52100	0.51999
	88	0.52450	0.51950	0.52799

Second Iteration:

Step size = 18

Number of filters in C1 to try = [70, 88, 106]

Number of filters in C2 to try = [70, 88, 106]

	Number of filters in C2			
Number of filters in C1		70	88	106
	70	0.51249	0.51700	0.52499
	88	0.52249	0.52799	0.52499
	106	0.51550	0.52499	0.53450

Third Iteration (non-diagonal combinations skipped for economic reasons):

Step size = 54

Number of filters in C1 to try = [124, 178, 214, 250]

Number of filters in C2 to try = [124, 178, 214, 250]

	Number of filters in C2				
Number of filters in C1		124	178	214	250
	124	0.52549	-	-	-
	178	-	0.55049	-	-
	214	-	-	0.55449	-
	250	-	-	-	0.55049

Therefore, the optimal number of filters is 214 for both C1 and C2.

2.3.3 Question 3

Using the optimal number of filters found in part (2), train the network by:

a. Adding the momentum term with momentum γ=0.1.

b. Using RMSProp algorithm for learning

c. Using Adam optimizer for learning

d. Adding dropout to the layers

Plot the training costs and test accuracies against epochs for each case.

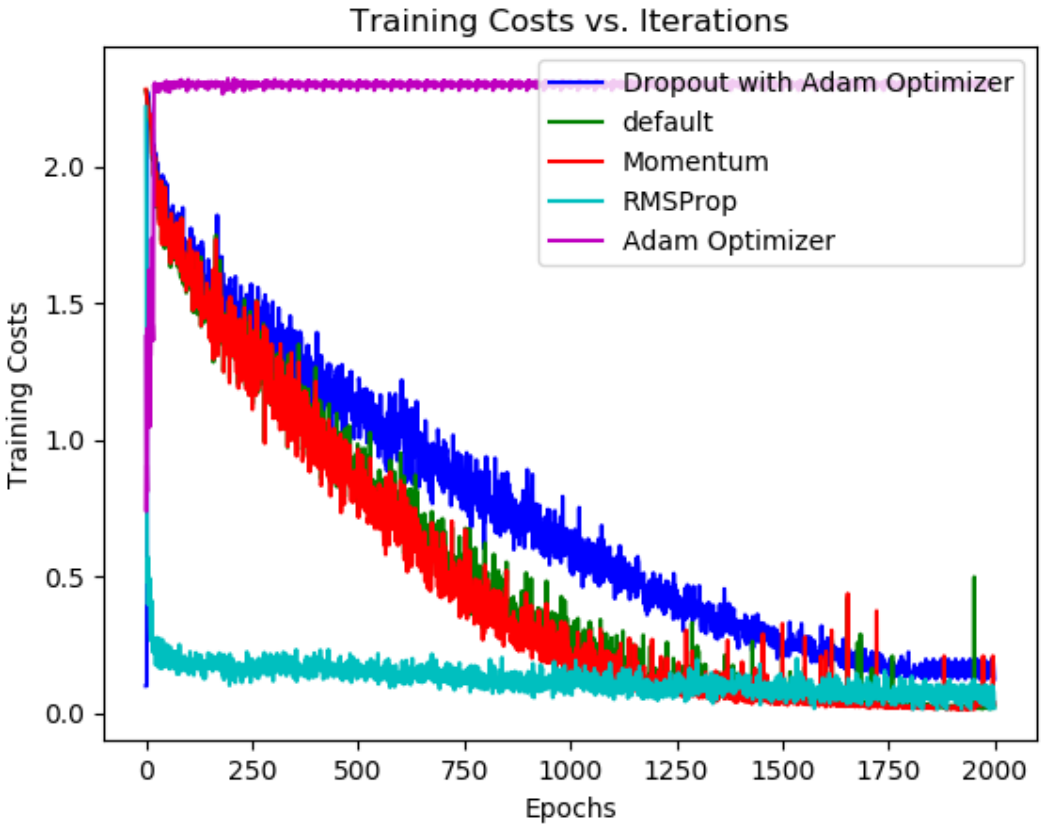

Figure 3.1: The plot of the training cost against epochs

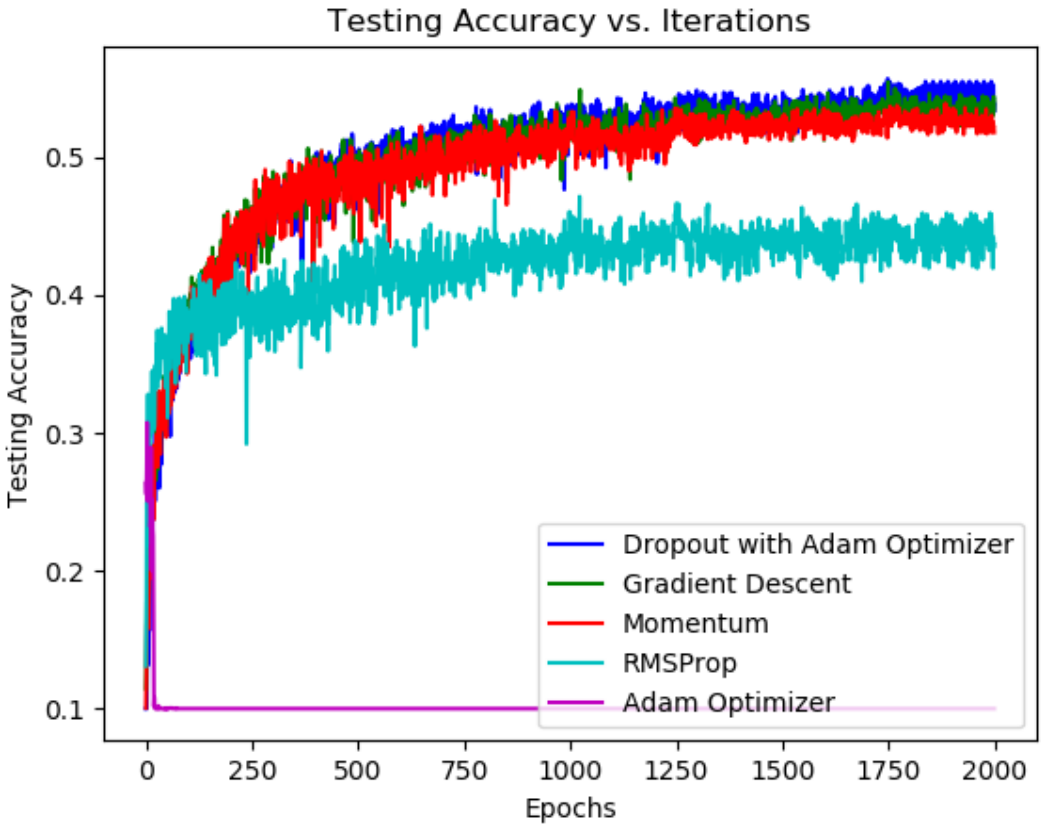

Figure 3.2: The plot of testing accuracy against epochs

2.3.4 Question 4

Compare the accuracies of all the models from parts (1) - (3) and discuss their performances.

Model	Max Test Accuracy
Gradient Descent Original Model	0.5099999904632568
Gradient Descent 214 Filters at both layers	0.5544999837875366
Momentum and 214 Filters at both layers	0.5389999747276306
RMSProp and 214 Filters at both layers	0.4715000092983246
Adam and 214 Filters at both layers	0.3075000047683716
Dropout of 0.5 and 214 Filters at both layers	0.5569999814033508

Therefore, the model trained on Gradient Descent with a dropout of 0.5 and 214 filters at both the first convolutional layer and the second convolutional layer performed the best with the highest test accuracy.

3 Part B: Text Classification Problem

3.1 Introduction

The aims are to implement CNN and RNN layers at the word and character levels for the classification of texts in the paragraphs of the dataset.

The dataset contains the first paragraphs collected from Wikipage entries and the corresponding labels about their category.

The dataset is split into two, training and test datasets with the training dataset containing 5600 records and the test dataset containing 700 records.

Each record will be labelled with one of the fifteen categories such as people, company, schools, etc.

3.2 Methods

Data Pre-processing

For the characters, as they have no meaning to a computer, it is processed into meaningful values through the function below:

```
char_processor = tf.contrib.learn.preprocessing.ByteProcessor(MAX_DOCUMENT_LENGTH)
```

After which, it will be converted into one-hot encoding to match the 256 different characters that can potentially appear.

```
char_vectors = tf.one_hot(x, NUM_CHARS)
```

For the words, the computer is unable to recognise a series of characters to form a word, hence it is processed into meaningful values through the function below.

```
vocab_processor = tf.contrib.learn.preprocessing.VocabularyProcessor(MAX_DOCUMENT_LENGTH)

x_transform_train = vocab_processor.fit_transform(x_train)
x_transform_test = vocab_processor.transform(x_test)
```

Before the words can be used as the training input, it must go through an embedding layer before it can be fed.

```
word_vectors = tf.contrib.layers.embed_sequence(x, vocab_size=num_words, embed_dim=EMBEDDING_SIZE)
```

Dropouts

To perform dropouts on the CNN, the probability that a node is kept is specified by the keep probability.

```
if dropout > 0:
  pool1 = tf.nn.dropout(pool1, keep_prob=dropout)
```

To perform dropouts on the RNN, the probability that a node is used as a input and given out as a output is specified by the respective keep probability.

```
if dropout > 0:
  cell = tf.contrib.rnn.DropoutWrapper(cell, input_keep_prob=dropout, output_keep_prob=dropout)
```

CNN Models

The CNN model is specified using a 2d convolutional layer with its respective parameters as shown below.

```
conv1 = tf.layers.conv2d(
    input_layer,
    filters=N_FILTERS,
    kernel_size=FILTER_SHAPE1,
    padding='VALID',
    activation=tf.nn.relu)
```

Additionally, a max pooling layer is used as well.

```
pool1 = tf.layers.max_pooling2d(
    conv1,
    pool_size=POOLING_WINDOW,
    strides=POOLING_STRIDE,
    padding='SAME')
```

RNN Models

The RNN model can be specified using the function below.

```
if model == 'gru':
 cell = tf.nn.rnn_cell.GRUCell(HIDDEN_SIZE)
elif model == 'vanilla':
 cell = tf.nn.rnn_cell.BasicRNNCell(HIDDEN_SIZE)
elif model == 'lstm':
 cell = tf.nn.rnn_cell.LSTMCell(HIDDEN_SIZE)
```

As for a multi-layered RNN, the function below is used.

```
num_units = [HIDDEN_SIZE, HIDDEN_SIZE]
cell = [tf.nn.rnn_cell.GRUCell(num_units=n) for n in num_units]
stacked_rnn_cell = tf.nn.rnn_cell.MultiRNNCell(cell)
```

Clippings

Below is how clipping is performed for the models.

```
gvs = optimizer.compute_gradients(entropy)
capped_gvs = [(tf.clip_by_value(grad, -2, 2.), var) for grad, var in gvs]
train_op = optimizer.apply_gradients(capped_gvs)
```

Mini-batch gradient descent learning

Mini-batch gradient descent learning is to utilise the benefits of training using batch descent learning and gradient descent learning by allowing the weights of the model to be updated by a mini-batch instead of the entire training data.

Parameters used

Learning Rate, α =0.01
Batch size = 128
Maximum length of characters / words = 100
Optimizer = Adam
Output layer = Softmax
Unique characters = 256

3.3 Experiment & Results

3.3.1 Question 1

In order to determine the optimal model, a total of 1500 epochs of training have been tested with checkpoints of 100 epochs. The result shown below is for 1500 epochs:

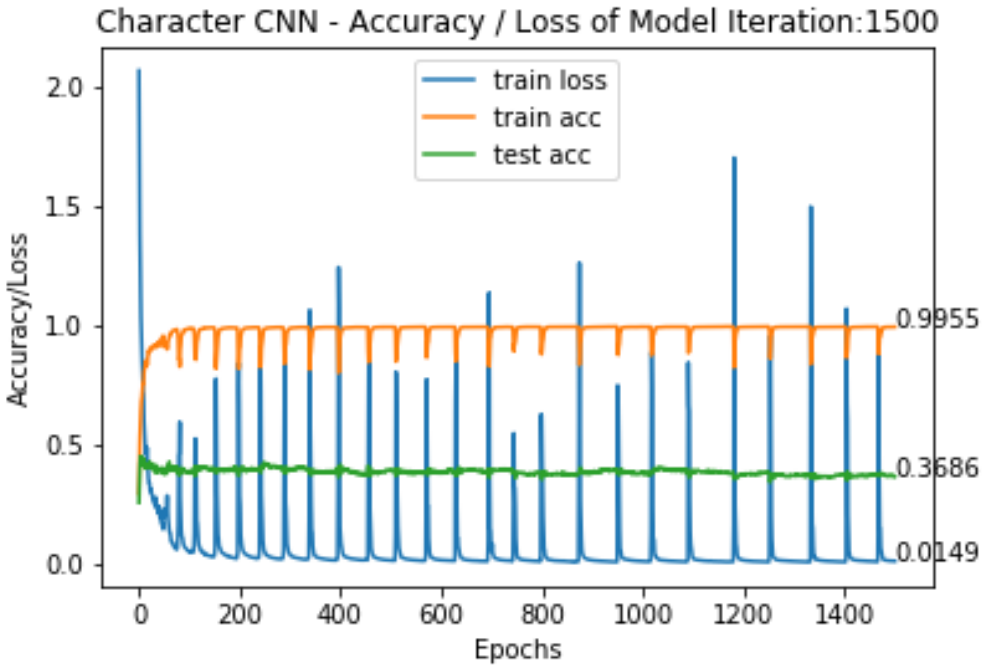

Figure 1.1 Accuracy/Loss of Model at 1500 epochs

As seen in the image above, there is an exploding gradient phenomenon as the training loss peaks towards one and above and quickly descends at random points.

Despite the peaks in the training loss, an interesting inference is the fact that the model manages to get its training loss to decrease even with the random peaks occurring with the number of epochs trained.

Below is a table showing the lowest training loss with the respective test accuracy within the range of 100 from 0 epoch up till 1500 epochs seen in Figure 1.1.

Epochs	Training Loss (4 s.f.)	Test Accuracy (4 s.f.)
0 – 99 (99)	0.05570	0.3829
100 -199 (192)	0.02602	0.4000
200- 299 (281)	0.02355	0.4057
300 – 399 (390)	0.01826	0.3886
400 – 499 (499)	0.01789	0.3943
500 – 599 (563)	0.01620	0.3814
600 – 699 (688)	0.01376	0.3914
700 – 799 (791)	0.01454	0.3829
800 – 899 (868)	0.01196	0.3729
900 – 999 (946)	0.01164	0.3943
1000 – 1099 (1086)	0.01207	0.3871
1100 – 1199 (1175)	0.01138	0.3814
1200 – 1299 (1245)	0.01172	0.3857
1300 – 1399 (1331)	0.01127	0.3771
1400 – 1500 (1402)	0.01172	0.3829

At epochs 1331, it provides us with the lowest entropy cost of 0.01127 on the training data and the accuracy of 37.71% on the testing data.

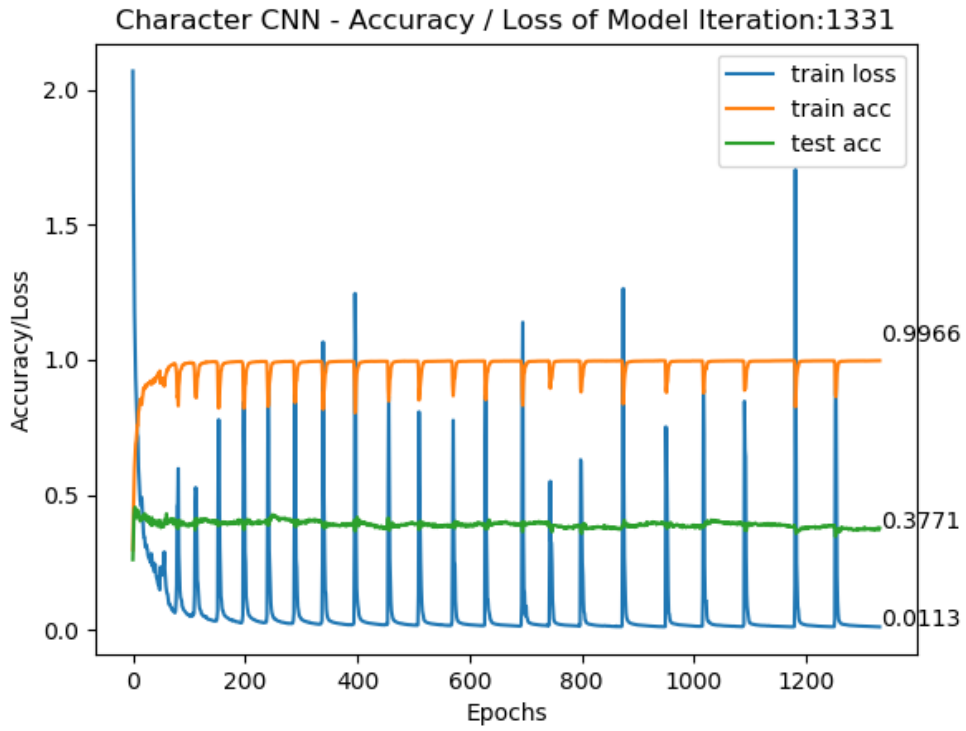

Figure 1.2 Accuracy/Loss of Optimal Model at 1331 epochs

The optimal model is shown above looks like it has been overfitted as the train accuracy is around 99.6% whereas the test accuracy is only 37.7%.

3.3.2 Question 2

Similar to the previous question, in order to determine the optimal model, a total of 1500 epochs of training have been tested with checkpoints of 100 epochs. The result shown below is for 1500 epochs:

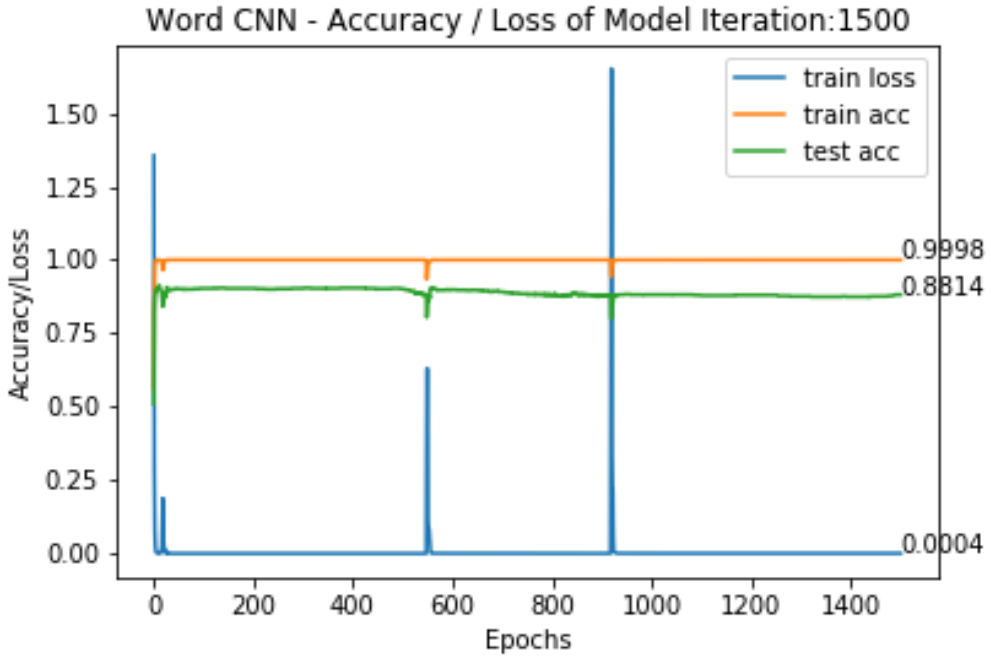

Figure 2.1 Accuracy/Loss of Model at 1500 epochs

As seen in the figure above, the exploding gradient phenomenon is not as frequent with only a few peaks in the graph. An observation made is that the training loss seems to have converge around less than 100 epochs as there are no significant improvement that can be observed.

Below is the table showing the lowest training loss achieved during the intervals of peaks observed in Figure 2.1.

Epochs	Training Loss (4 s.f.)	Test Accuracy (4 s.f.)
0 – 18 (12)	1.89E-4	0.9086
19 -549 (545)	2.554E-10	0.8843
550- 918 (917)	2.488E-4	0.8814
919 – 1500 (1497)	4.017E-4	0.8814

At epochs 545, it provides us with the lowest entropy cost of 2.554E-10 on the training data and the accuracy 88.43% on the testing data.

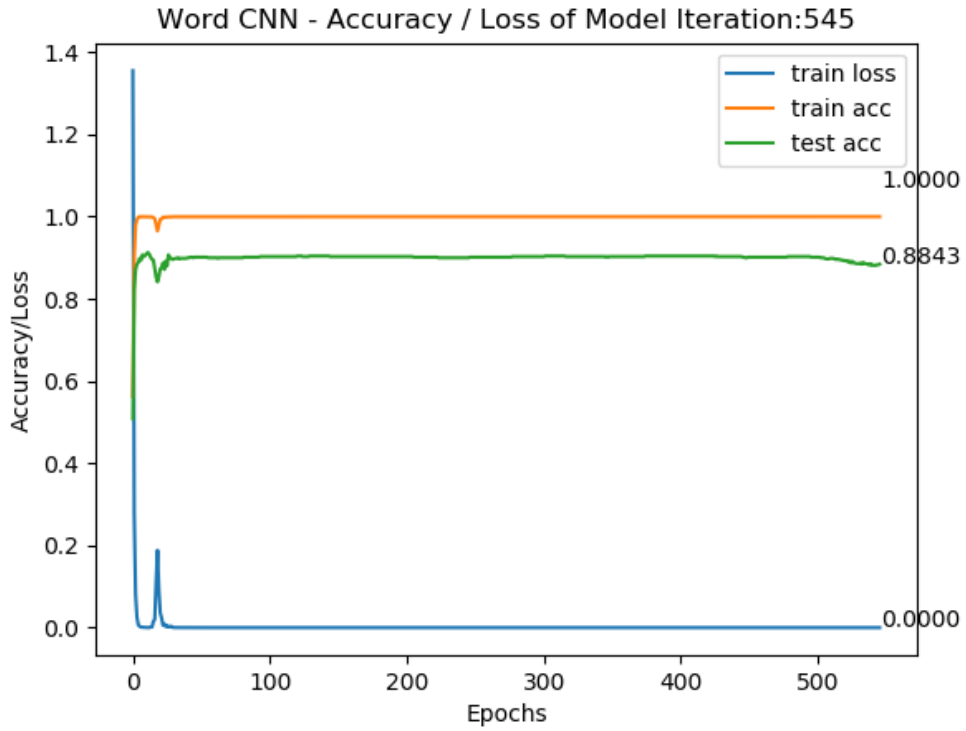

Figure 2.2 Accuracy/Loss of Optimal Model at 545 epochs

The optimal model is shown above seems to have generalize well with at least a test accuracy of around 88.4% even though it has a train accuracy of 100%.

3.3.3 Question 3

Similar to the previous question, in order to determine the optimal model, a total of 1500 epochs of training have been tested with checkpoints of 100 epochs. The result shown below is for 1500 epochs:

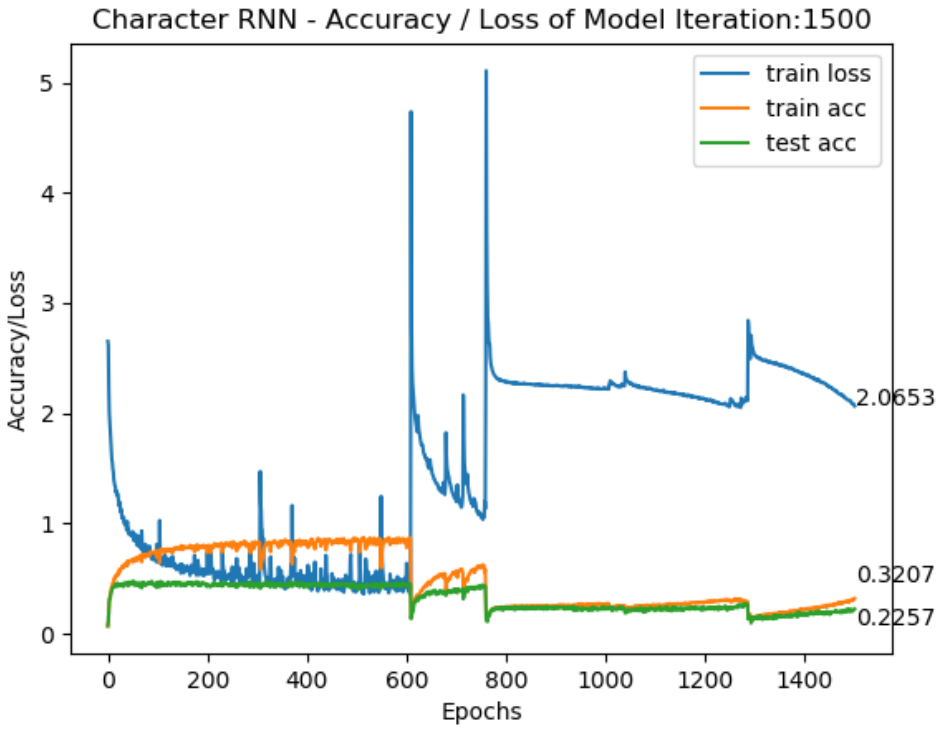

Figure 3.1 Accuracy/Loss of Model at 1500 epochs

For this model, the exploding gradient phenomenon is quite obvious which causes the training loss to diverge consistently above one after around epoch 600. After epoch 600, the model seems to have lost traction on the training data which shows irrelevant learning of weights and biases.

Below is the table showing the training loss achieved during the last intervals of 100 epochs from epoch 0 to epoch 600 observed in Figure 3.1.

Epochs	Training Loss (4 s.f.)	Test Accuracy (4 s.f.)
100	0.6600	0.4686
200	0.7385	0.4314
300	0.4946	0.4414
400	0.4696	0.45
500	0.3991	0.4257
600	0.6401	0.4314

At epochs 500, it provides us with the lowest entropy cost of 0.3991 on the training data and the accuracy of 42.57% on the testing data.

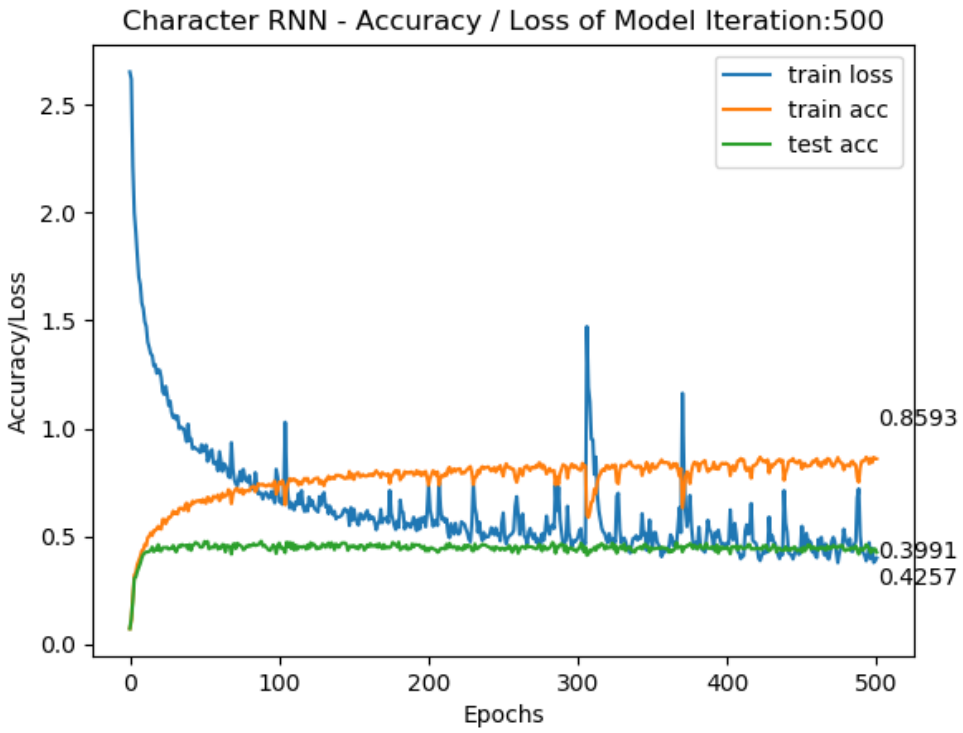

Figure 3.2 Accuracy/Loss of Optimal Model at 500 epochs

The optimal model is shown above seems to be severely underfitted with a training accuracy of around 39.7% but has generalize well as the test accuracy is around 42.6% which is a small difference of around 3%.

3.3.4 Question 4

Similar to the previous question, in order to determine the optimal model, a total of 1500 epochs of training have been tested with checkpoints of 100 epochs. The result shown below is for 1500 epochs:

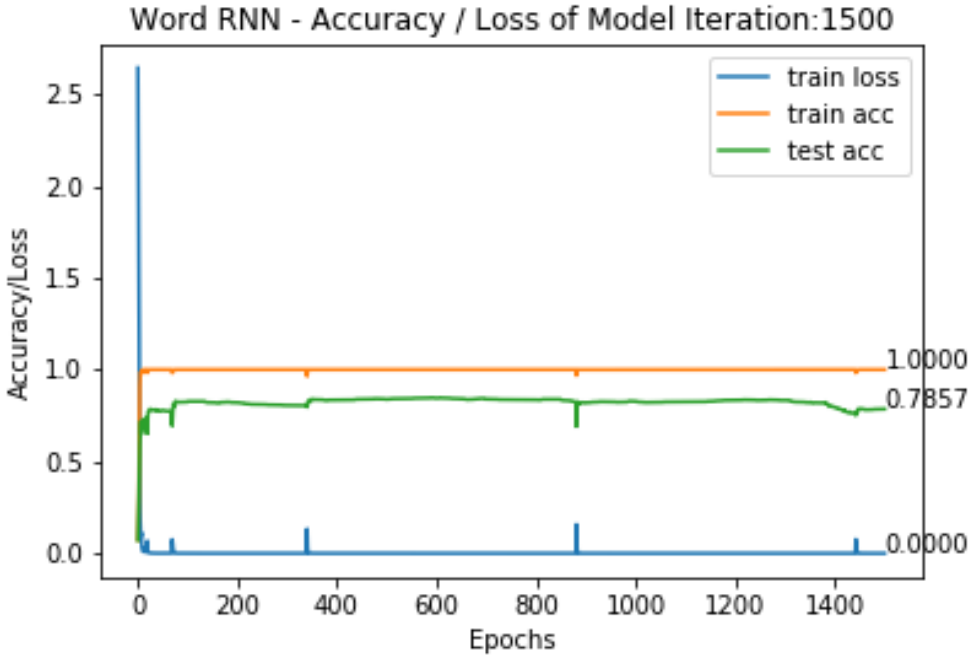

Figure 4.1 Accuracy/Loss of Model at 1500 epochs

As seen in the figure above, the exploding gradient phenomenon rarely occurs with small peaks below 0.5 loss in the graph. An observation made is that the training loss seems to have converge around less than 100 epochs as there are no significant improvement that can be observed.

Below is the table showing the lowest training loss achieved during the intervals of small peaks observed in Figure 4.1.

Epochs	Training Loss (4 s.f.)	Test Accuracy (4 s.f.)
0 – 68 (67)	1.560E-4	0.7714
69 - 399 (338)	3.899E-7	0.8043
340 - 880 (838)	3.576E-9	0.8400
881 – 1441 (1416)	4.683E-9	0.7786
1442 – 1500 (1500)	2.642E-6	0.7857

At epochs 838, it provides us with the lowest entropy cost of 3.576E-9 on the training data and the accuracy of 84% on the testing data.

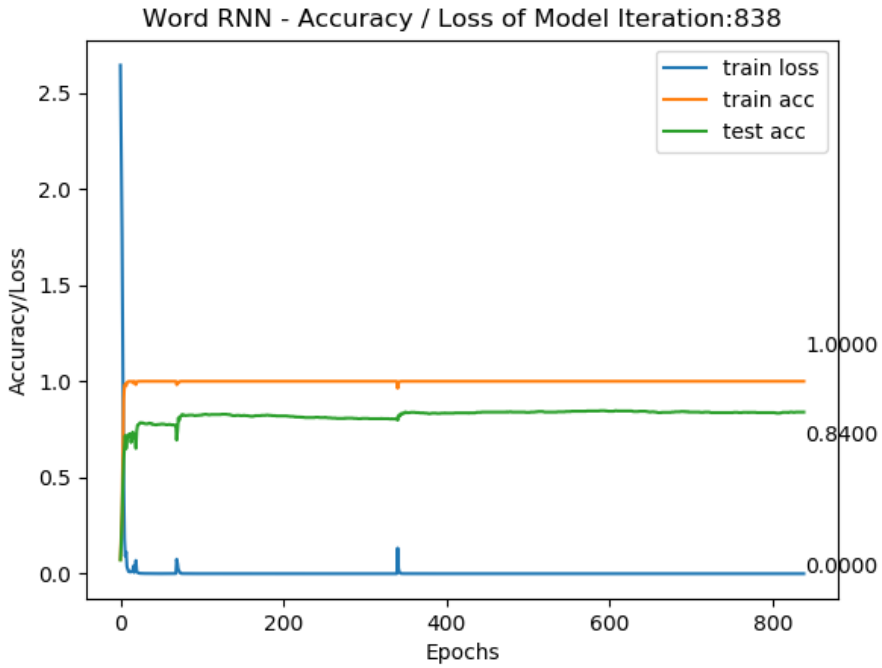

Figure 4.2 Accuracy/Loss of Optimal Model at 838 epochs

The optimal model is shown above seems to have generalize well with at least a test accuracy of around 84% even though it has a train accuracy of 100%.

3.3.5 Question 5

The table below shows the running time for each of the models from epoch 0 to 1500 in the intervals of 100 epochs. The timing highlighted in green corresponds to the optimal model chosen in questions 1 – 4.

Running Time Comparisons				
Epochs	Character CNN	Character RNN	Word CNN	Word RNN
100	32s	2m 15s	13s	2m 27s
200	1m	4m 23s	26s	4m 49s
300	1m 29s	6m 30s	39s	7m 10s
400	1m 57s	8m 39s	52s	9m 32s
500	2m 26s	10m 48s	1m 5s	11m 53s
600	2m 55s	12m 59s	1m 17s	14m 16s
700	3m 24s	15m 9s	1m 30s	16m 38s
800	3m 53s	17m 20s	1m 43s	19m
900	4m 21s	19m 32s	1m 56s	21m 31s
1000	4m 50s	21m 45s	2m 09s	23m 58s
1100	5m 19s	24m 5s	2m 21s	26m 20s
1200	5m 47s	26m 19s	2m 34s	28m 41s
1300	6m 16s	28m 31s	2m 47s	31m 02s
1400	6m 45s	30m 42s	3m	33m 24s
1500	7m 13s	32m 53s	3m 13s	35m 45s

In general, the time taken for the model to process ranked in terms of fastest to slowest is as follows:

1. Word CNN
2. Character CNN
3. Character RNN
4. Word RNN

RNN is shown to take much longer than CNN models as they rank 3rd and 4th in terms of the running time. However, an interesting fact is that the models which receives word ids for CNN is much faster than its counterpart which receives character ids, whereas this is the opposite for RNN models.

The difference in timing between training a CNN and RNN model is quite significant with more than double the time needed to train for the same number of epochs.

As for the networks implemented in questions 1 – 4, the ranking in terms of fastest to slowest is as follows:

1. Word CNN (1400 epochs) – 1m 17s
2. Character CNN (600 epochs) – 6m 45s
3. Character RNN (500 epochs) – 10m 48s
4. Word RNN (900 epochs) – 21m 31s

Despite having to train CNN models for longer epochs to achieve optimal models, the running time taken to train CNN models is much lesser than training RNN models.

The table below shows the test accuracies for the networks implemented in questions 1 – 4.

Test Accuracy Comparisons (3 significant figures)				
	Character CNN	Character RNN	Word CNN	Word RNN
Epochs	1331	500	545	838
Accuracy	37.714%	42.571%	88.429%	83.999%

Between the two models which takes in character ids, the RNN model is shown to perform slightly better with a higher accuracy of around 5% as compared to the CNN model.

Whereas between the two models which takes in word ids, the CNN model performs slightly better than the RNN model with a higher accuracy of around 4%.

RNN models are trained to recognise patterns across time whereas CNN models are trained to recognise patterns across space. The higher accuracy in a character based RNN could be due to RNN's ability to make relations with previous steps characters to form meaning.

Whereas the higher accuracy in a word based CNN could be due to CNN's ability to locate local and position-invariant features of words.

For the experiment of dropouts to the layers of the model in questions 1 – 4, the **percentage of keep probability used is 0.2, 0.4, 0.6 and 0.8.** Hence, **dropout = 1 - <Keep Probability>.**

3.3.5.1 Character CNN with dropout

Dropout of 0.8

The results for the probability that each element is dropped by 20%, 40%, 60% and 80% for the character CNN is shown below.

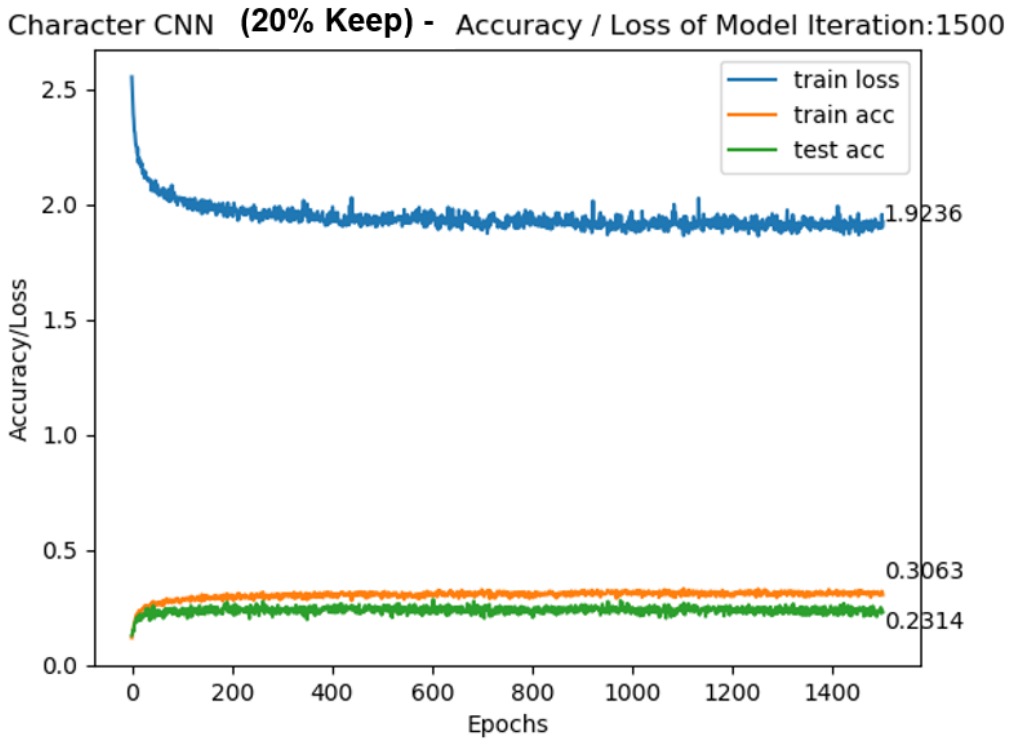

Figure 5.1 Character - Accuracy/Loss of Model with keep = 0.2 at 1500 epochs

At epochs 1251, it provides us with the lowest entropy cost of 1.8623 on the training data and the accuracy of 23.29% on the testing data. The optimal model is shown below.

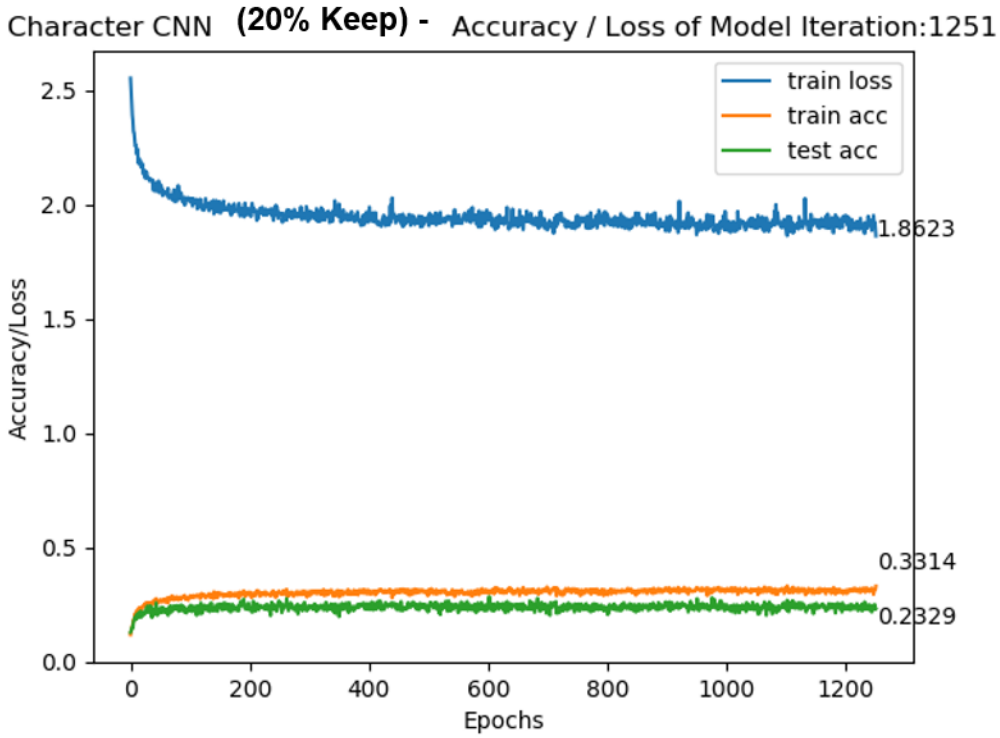

Figure 5.2 Character - Accuracy/Loss of Optimal Model with keep = 0.2 at 1251 epochs

Dropout of 0.6

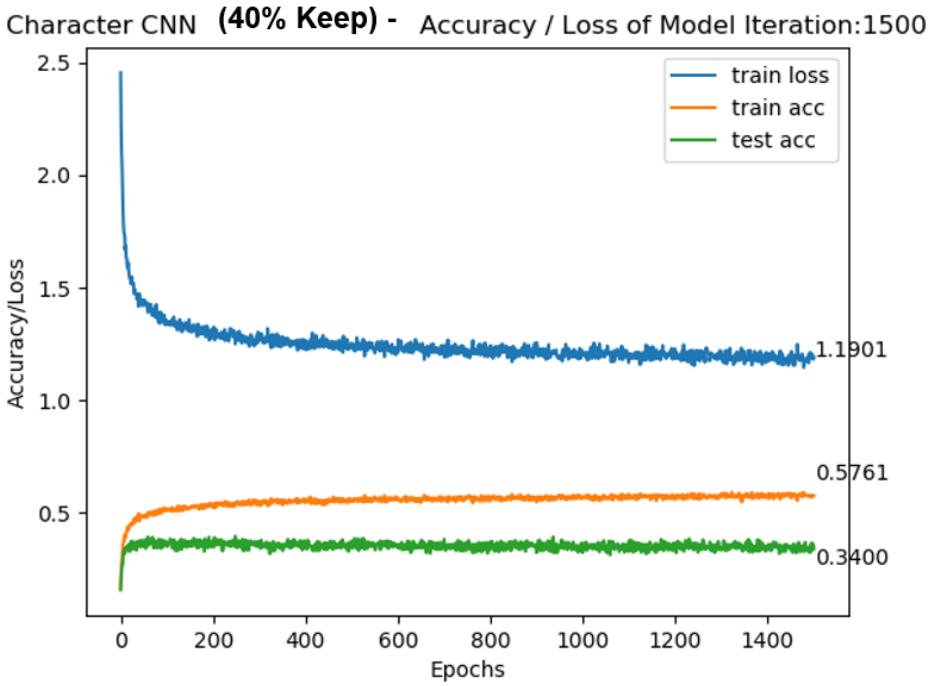

Figure 5.3 Character - Accuracy/Loss of Model with keep = 0.4 at 1500 epochs

At epochs 1478, it provides us with the lowest entropy cost of 1.1451 on the training data and the accuracy of 32.71% on the testing data. The optimal model is shown below.

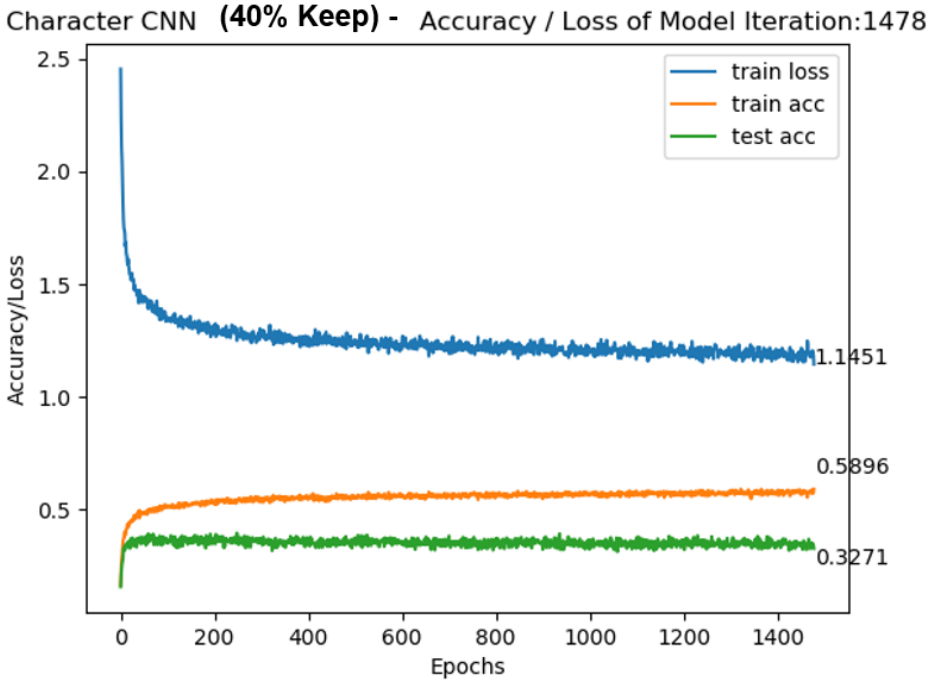

Figure 5.4 Character - Accuracy/Loss of Optimal Model with keep = 0.4 at 1478 epochs

Dropout of 0.4

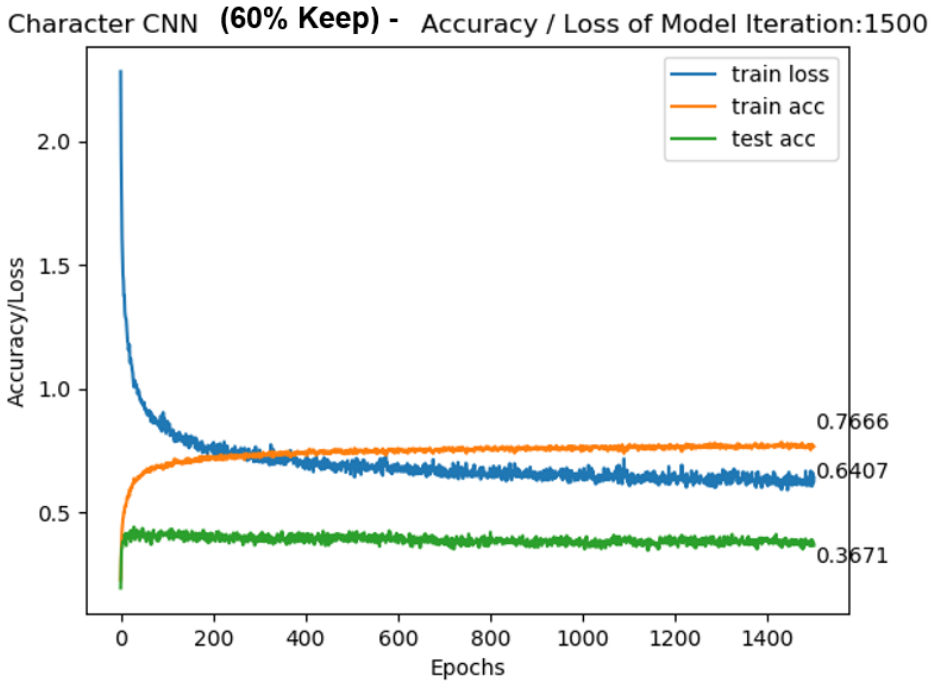

Figure 5.5 Character - Accuracy/Loss of Model with keep = 0.6 at 1500 epochs

At epochs 1443, it provides us with the lowest entropy cost of 0.5905 on the training data and the accuracy of 38% on the testing data. The optimal model is shown below.

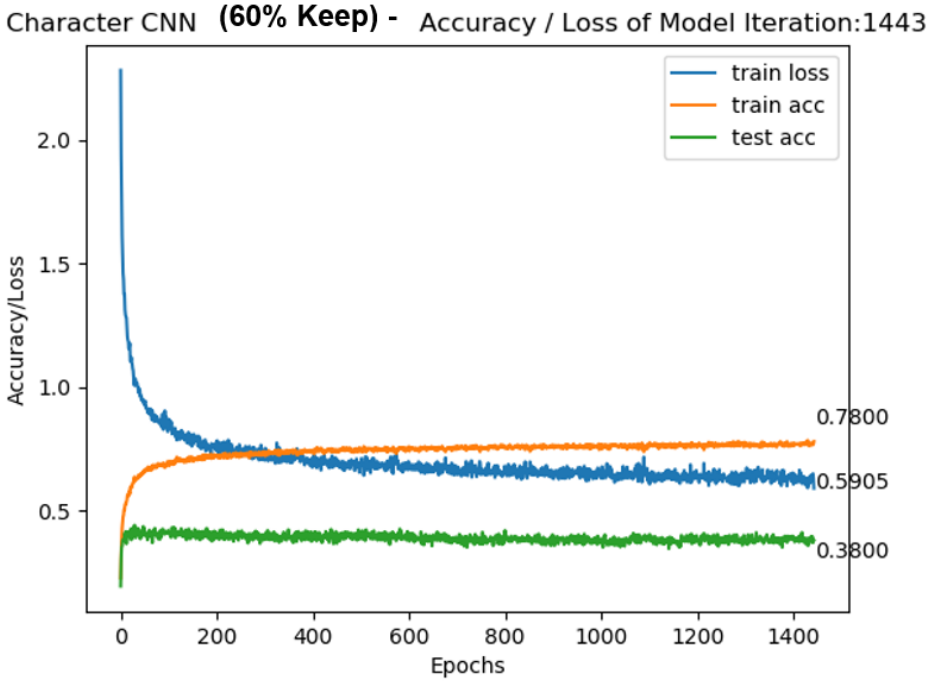

Figure 5.6 Character - Accuracy/Loss of Optimal Model with keep = 0.6 at 1443 epochs

Dropout of 0.2

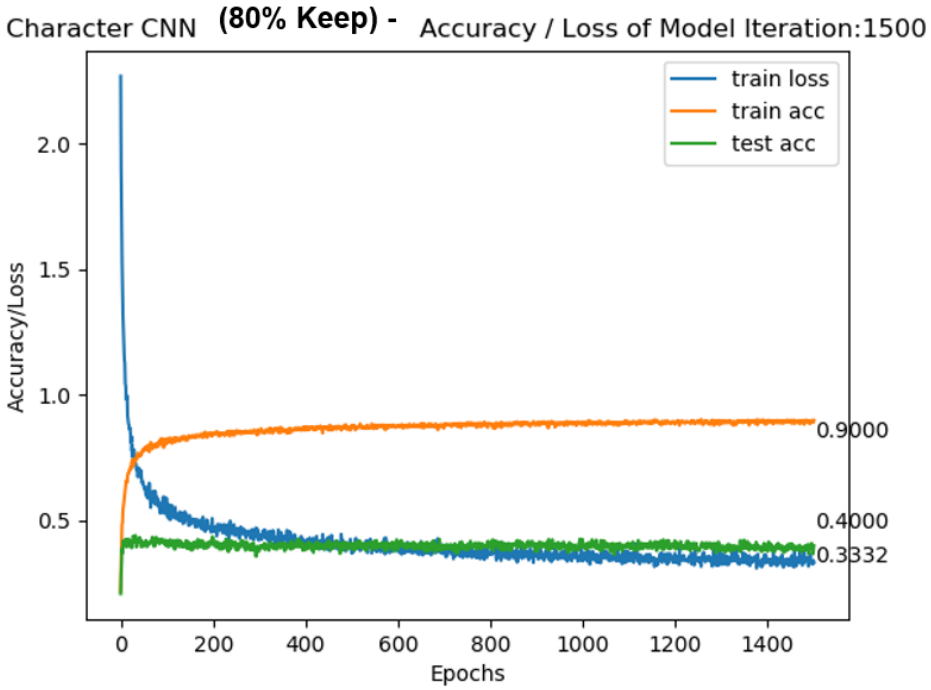

Figure 5.7 Character - Accuracy/Loss of Model with keep = 0.8 at 1500 epochs

At epochs 1376, it provides us with the lowest entropy cost of 0.3101 on the training data and the accuracy of 38.57% on the testing data.

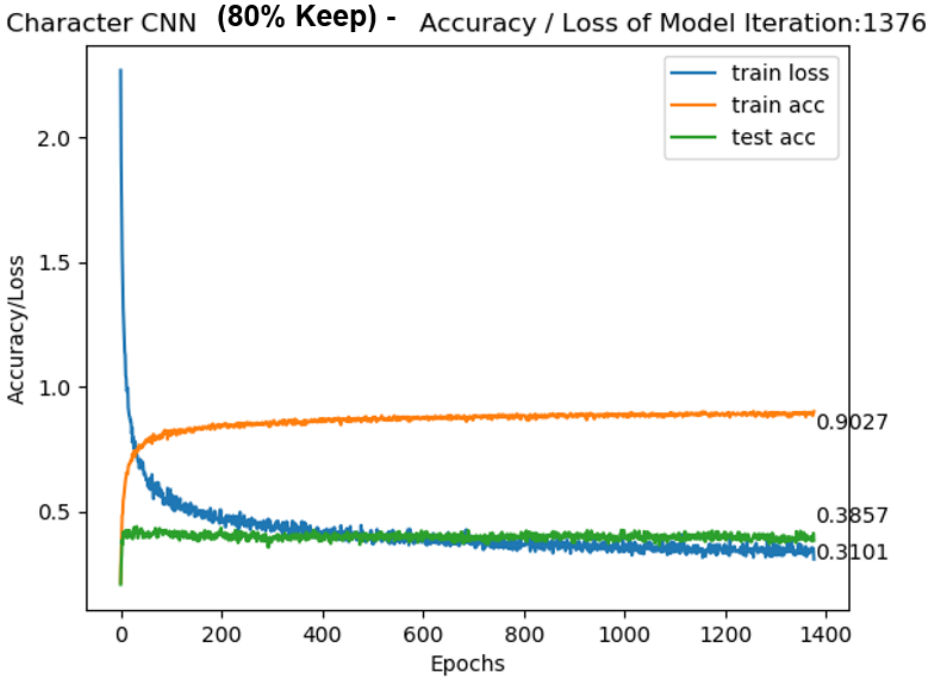

Figure 5.8 Character - Accuracy/Loss of Optimal Model with keep = 0.8 at 1376 epochs

Character CNN Dropout Percentage Comparisons (3 significant figures)					
	0	0.8	0.6	0.4	0.2
Epochs	1331	1251	1478	1443	1376
Train Loss	0.0113	1.862	1.145	0.591	0.310
Test Accuracy	37.714%	23.286%	32.714%	38%	38.571%

From the table shown above, if more than 50% of each pooling layer is dropout, the overall performance of the model decreases as seen in the dropout of 80% with a test accuracy of 23.29% (~-14%) and 60% with a test accuracy of 32.71% (~-5%).

However, if less than 50% of each pooling layer is dropout, the overall performance of the model increases as seen in the dropout of 40% with a test accuracy of 38% (~+0.5%) and 20% with a test accuracy of 38.57% (~+1%).

Dropouts are meant to encourage the network to learn a sparse representation however in this case the models are all underfitted hence not much changes will occur.

3.3.5.2 Word CNN with dropout

Dropout of 0.8

The results for the probability that each element is dropped by 20%, 40%, 60% and 80% for the word CNN is shown below.

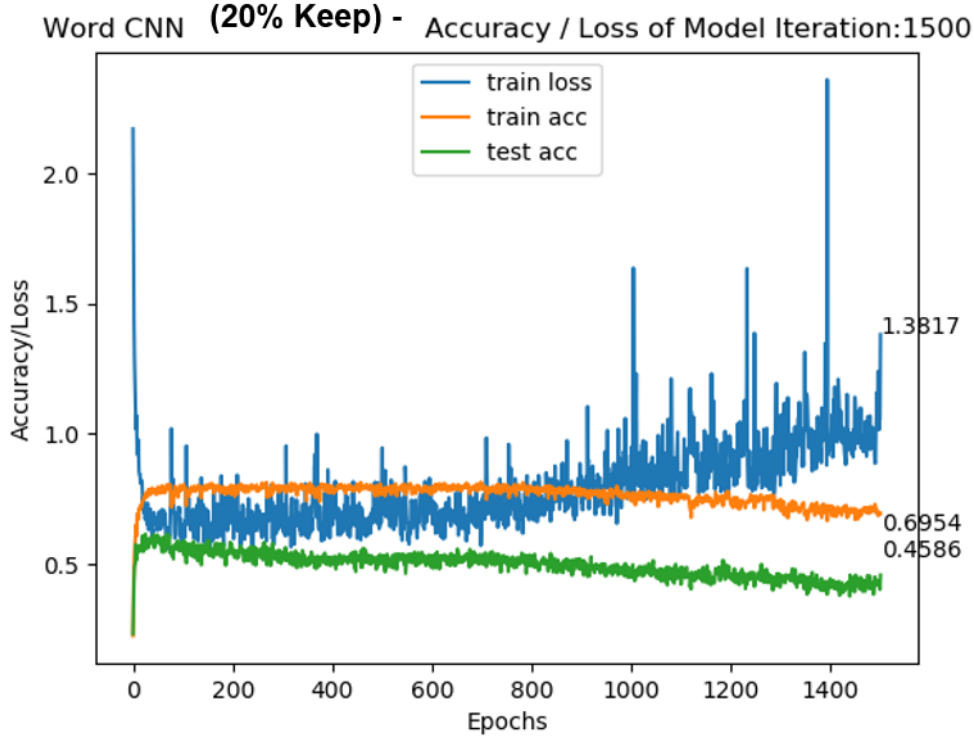

Figure 5.9 Word - Accuracy/Loss of Model with keep = 0.2 at 1500 epochs

At epochs 68, it provides us with the lowest entropy cost of 0.5586 on the training data and the accuracy of 58.71% on the testing data.

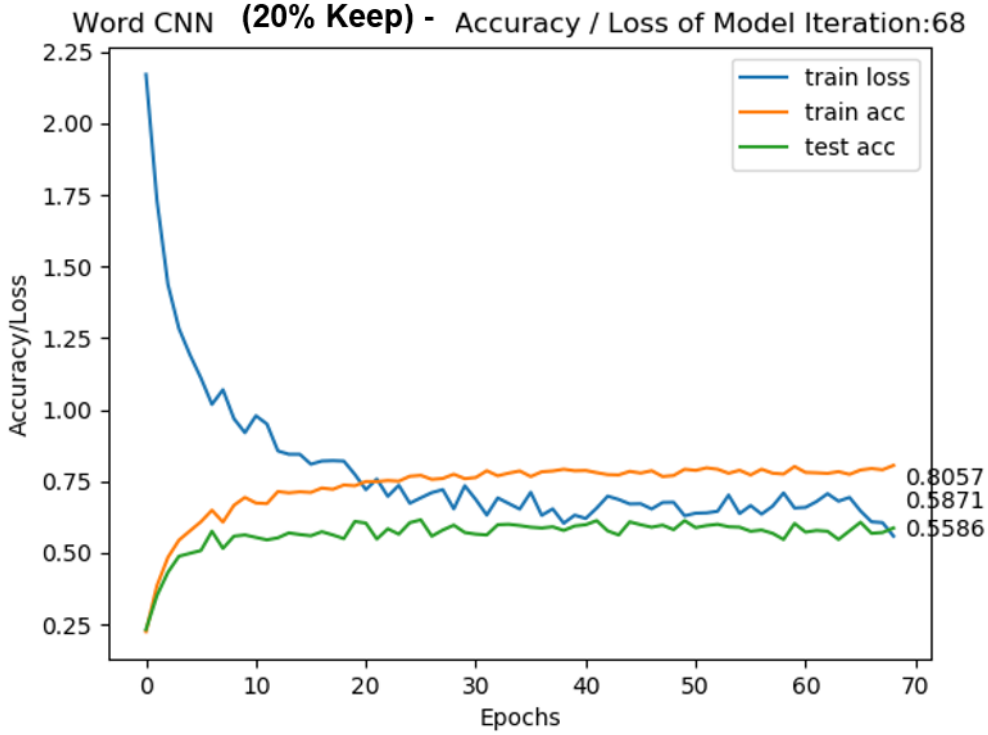

Figure 5.10 Word - Accuracy/Loss of Optimal Model with keep = 0.2 at 68 epochs

Dropout of 0.6

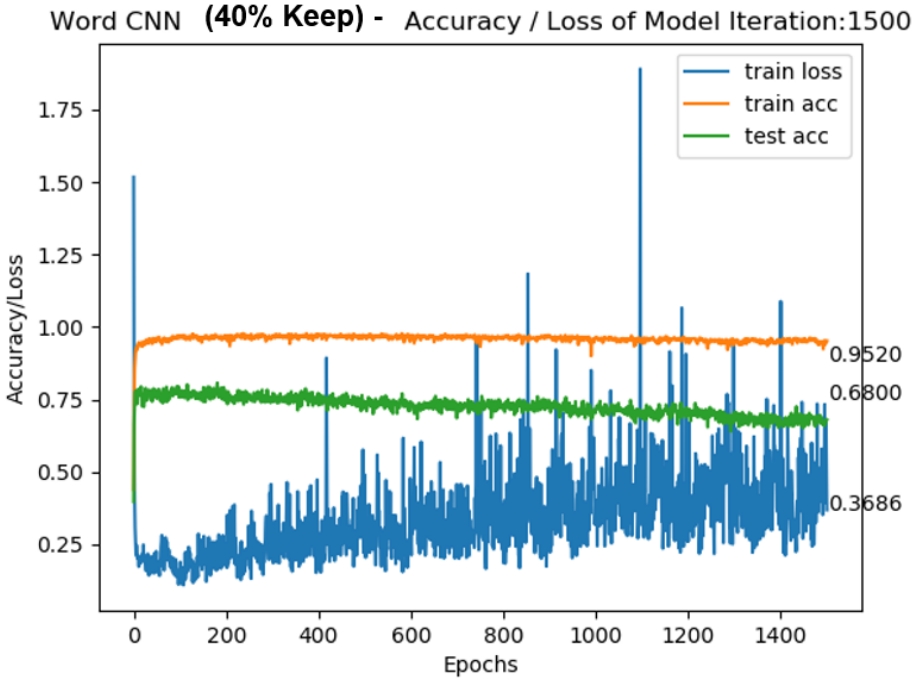

Figure 5.11 Word - Accuracy/Loss of Model with keep = 0.4 at 1500 epochs

At epochs 108, it provides us with the lowest entropy cost of 0.1095 on the training data and the accuracy of 75.86% on the testing data.

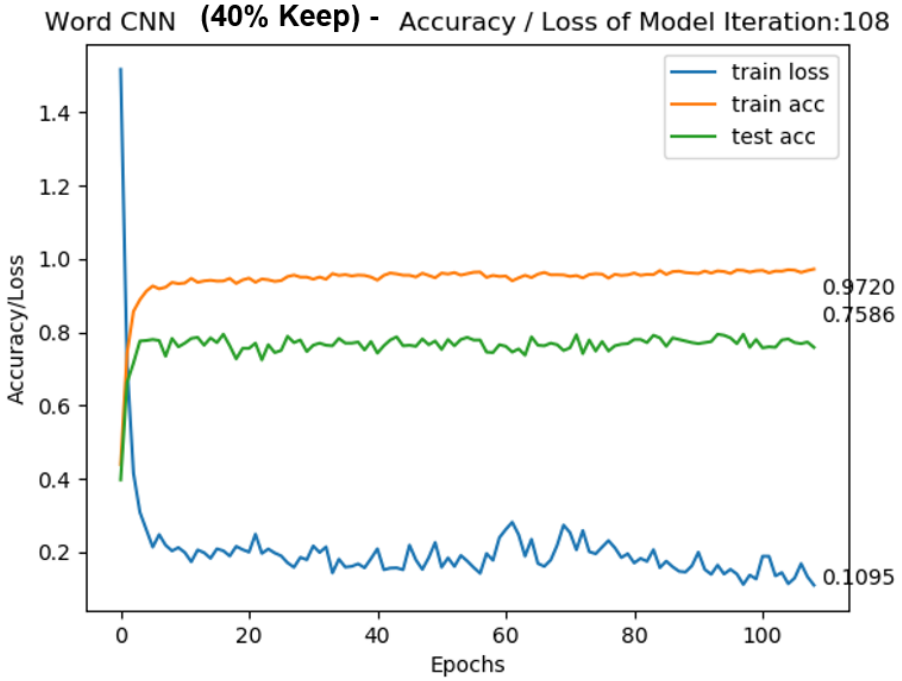

Figure 5.12 Word - Accuracy/Loss of Optimal Model with keep = 0.4 at 108 epochs

Dropout of 0.4

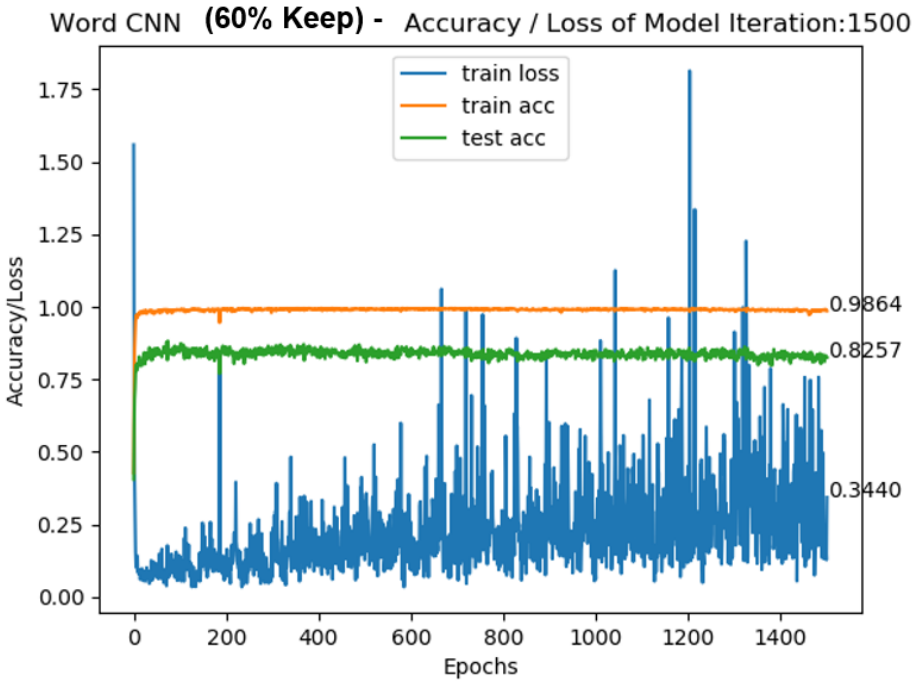

Figure 5.13 Word - Accuracy/Loss of Model with keep = 0.6 at 1500 epochs

At epochs 234, it provides us with the lowest entropy cost of 0.03441 on the training data and the accuracy of 82.71% on the testing data.

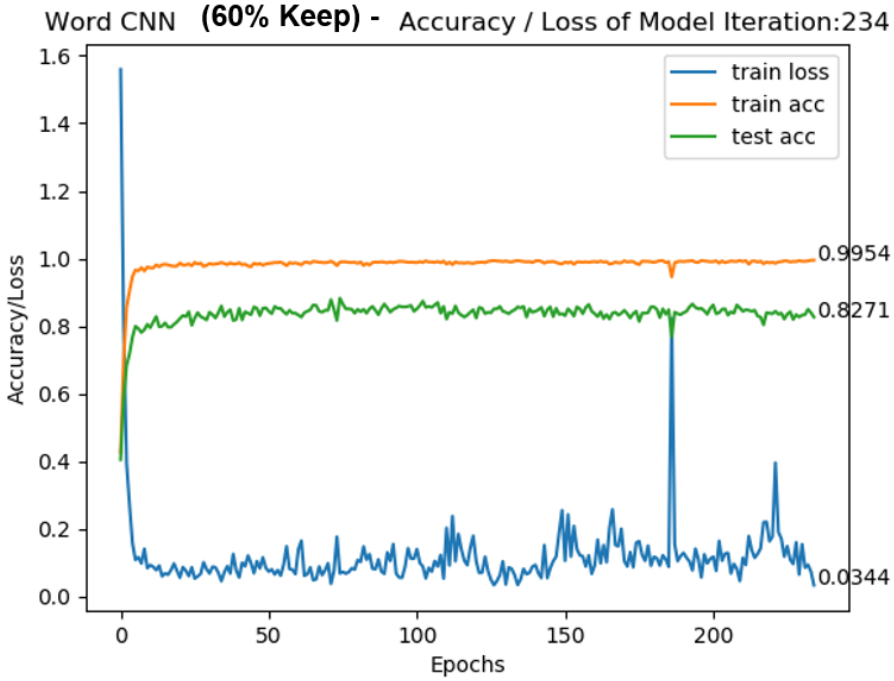

Figure 5.14 Word - Accuracy/Loss of Optimal Model with keep = 0.6 at 234 epochs

Dropout of 0.2

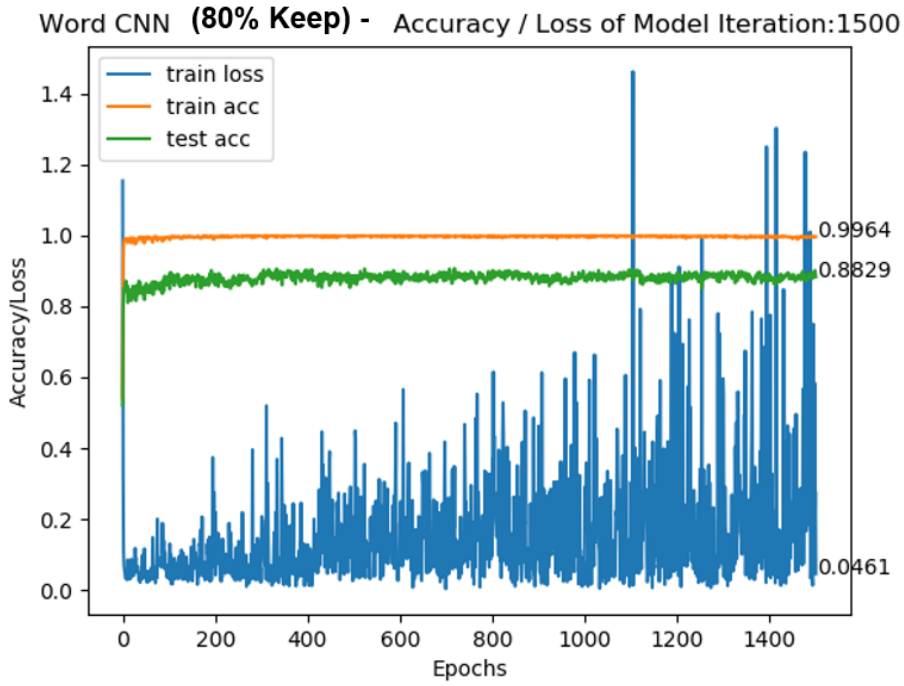

Figure 5.15 Word - Accuracy/Loss of Model with keep = 0.8 at 1500 epochs

At epochs 699, it provides us with the lowest entropy cost of 0.003644 on the training data and the accuracy of 89.57% on the testing data.

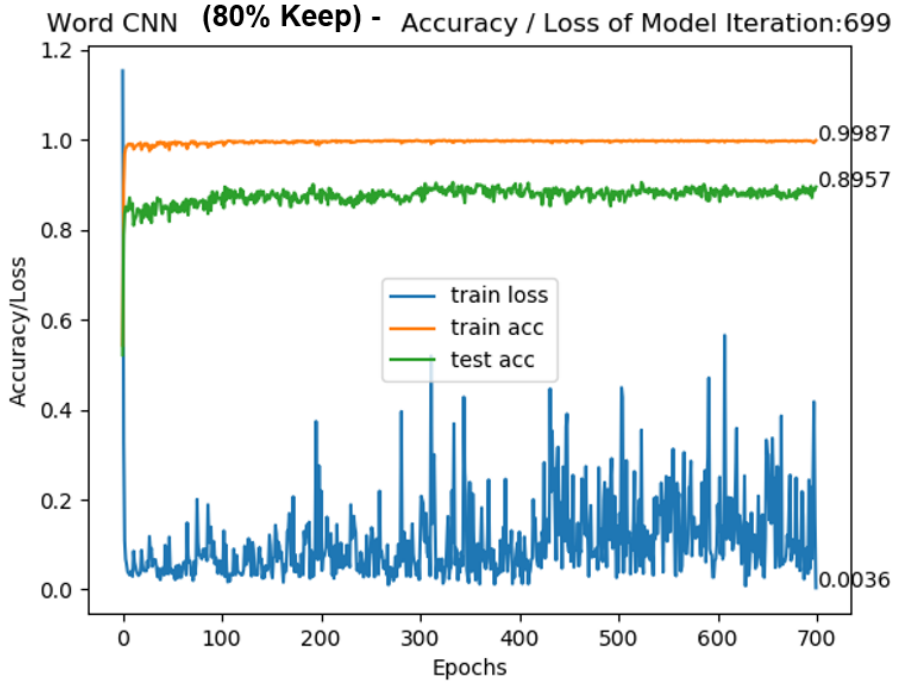

Figure 5.16 Word - Accuracy/Loss of Optimal Model with keep = 0.8 at 699 epochs

Word CNN Dropout Percentage Comparisons (3 significant figures)					
	0	0.8	0.6	0.4	0.2
Epochs	545	68	108	234	699
Train Loss	2.554E-10	0.5586	0.1095	0.03441	0.003644
Test Accuracy	88.428%	58.714%	75.857%	82.714%	89.571%

From the table shown above, if 40% or more of each pooling layer is dropout, the overall performance of the model decreases as seen in the dropout of 80% with a test accuracy of 58.71% (~-30%), 60% with a test accuracy of 75.86% (~-13%) and 40% with a test accuracy of 82.71% (~-6%).

However, if less than 40% of each pooling layer is dropout, the overall performance of the model increases as seen in the dropout of 20% with a test accuracy of 89.57% (~+1%).

Dropouts are meant to encourage the network to learn a sparse representation and with small amount would allow the model to learn better by providing noisier environment to better generalize the model.

3.3.5.3 Character RNN with dropout

Dropout of 0.8

The results for the probability that each element is dropped by 20%, 40%, 60% and 80% for the character RNN is shown below.

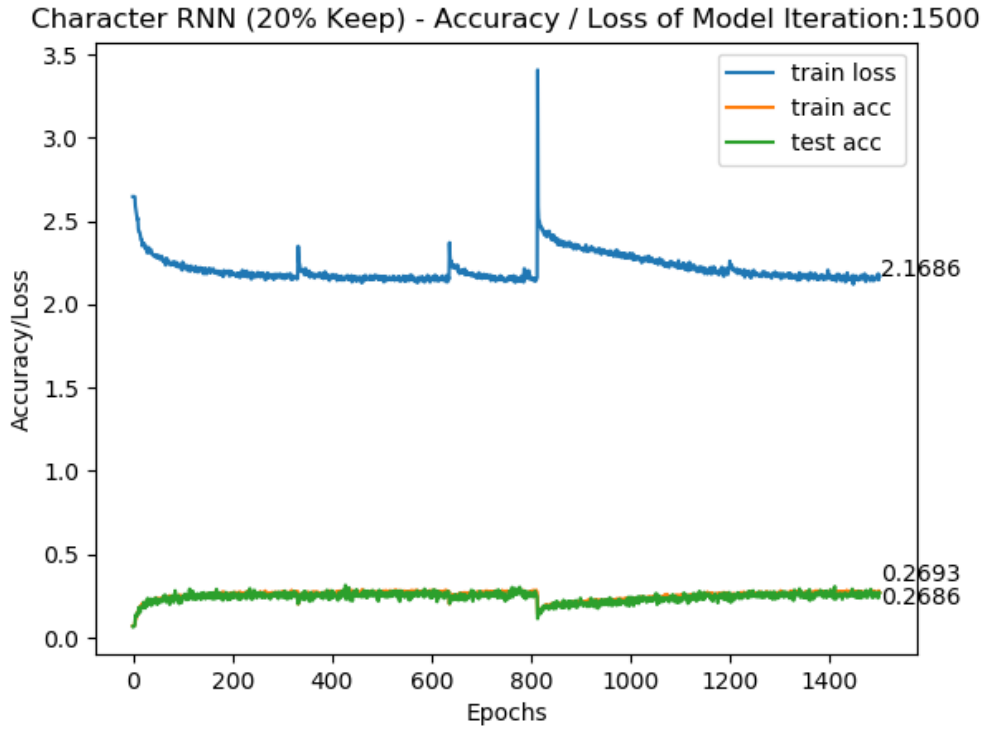

Figure 5.17 Character - Accuracy/Loss of Model with keep = 0.2 at 1500 epochs

At epochs 1447, it provides us with the lowest entropy cost of 2.1203 on the training data and the accuracy of 27.71% on the testing data.

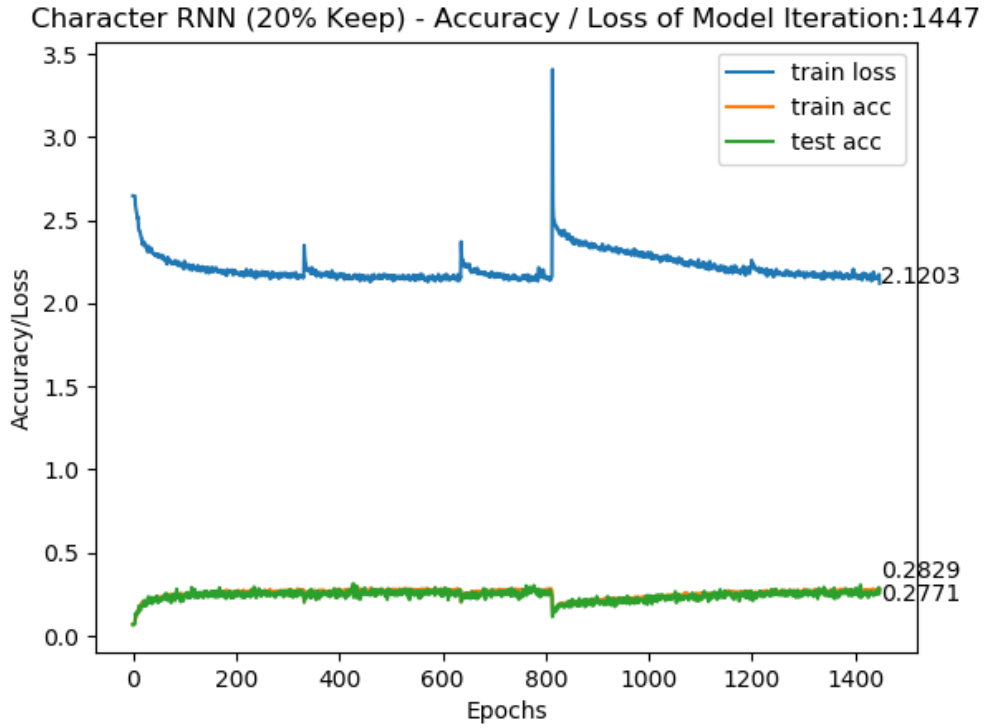

Figure 5.18 Character - Accuracy/Loss of Optimal Model with keep = 0.2 at 1447 epochs

Dropout of 0.6

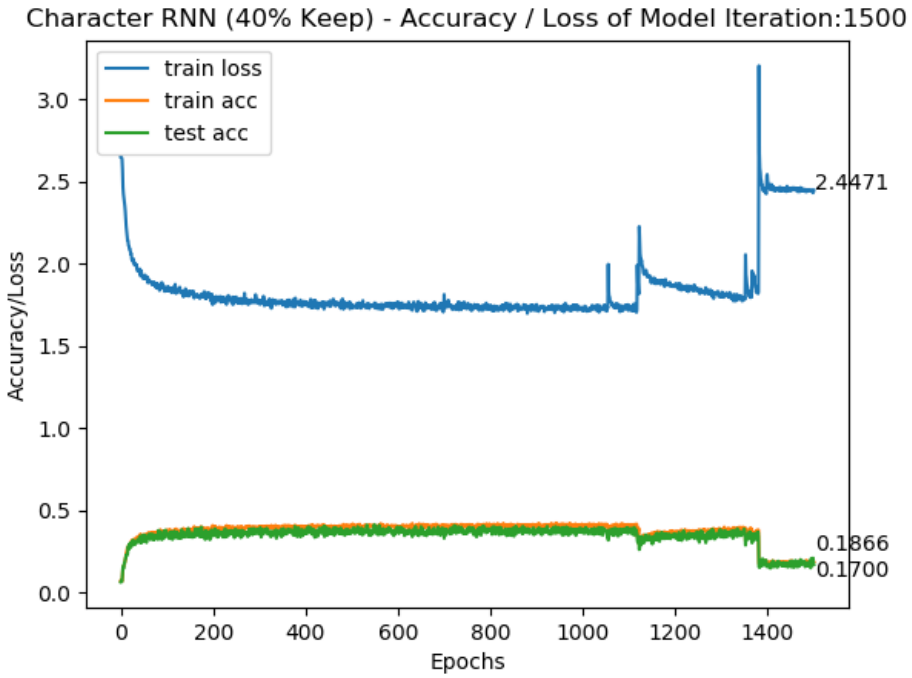

Figure 5.19 Character - Accuracy/Loss of Model with keep = 0.4 at 1500 epochs

At epochs 880, it provides us with the lowest entropy cost of 1.6971 on the training data and the accuracy of 38.57% on the testing data.

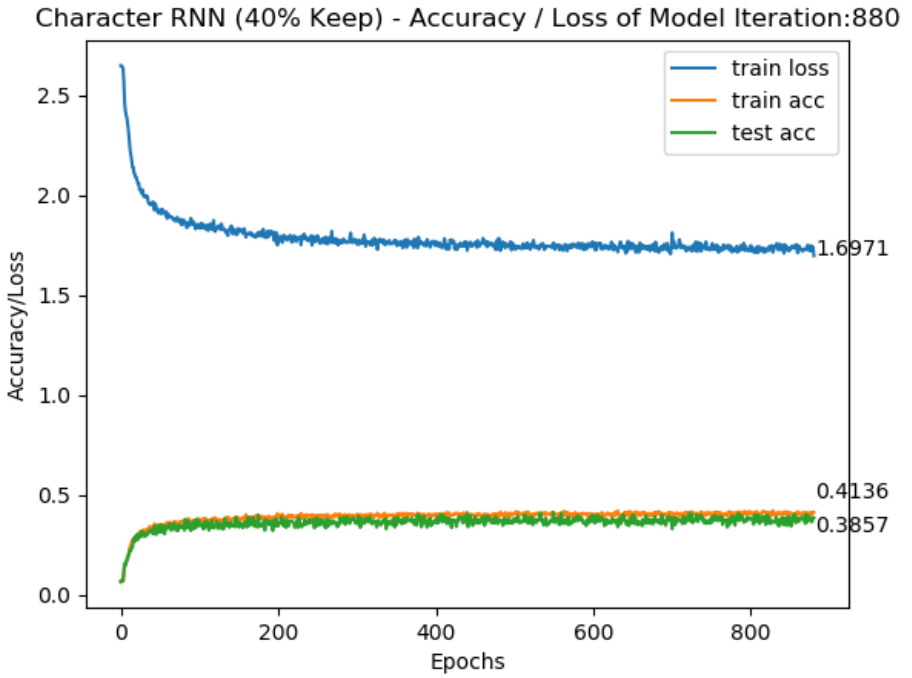

Figure 5.20 Character - Accuracy/Loss of Optimal Model with keep = 0.4 at 880 epochs

Dropout of 0.4

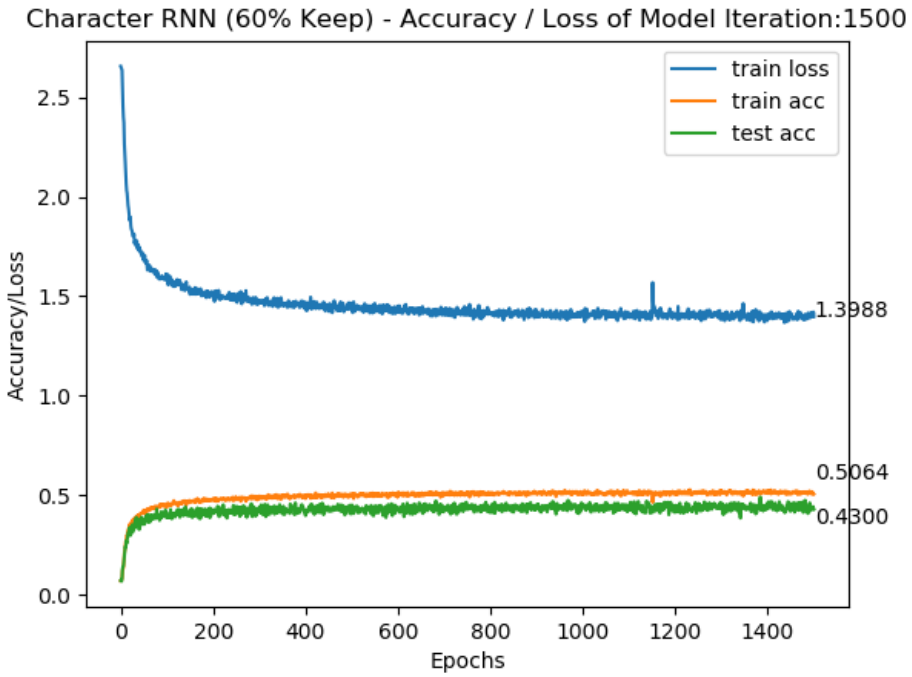

Figure 5.21 Character - Accuracy/Loss of Model with keep = 0.6 at 1500 epochs

At epochs 1436, it provides us with the lowest entropy cost of 1.3679 on the training data and the accuracy of 42.29% on the testing data.

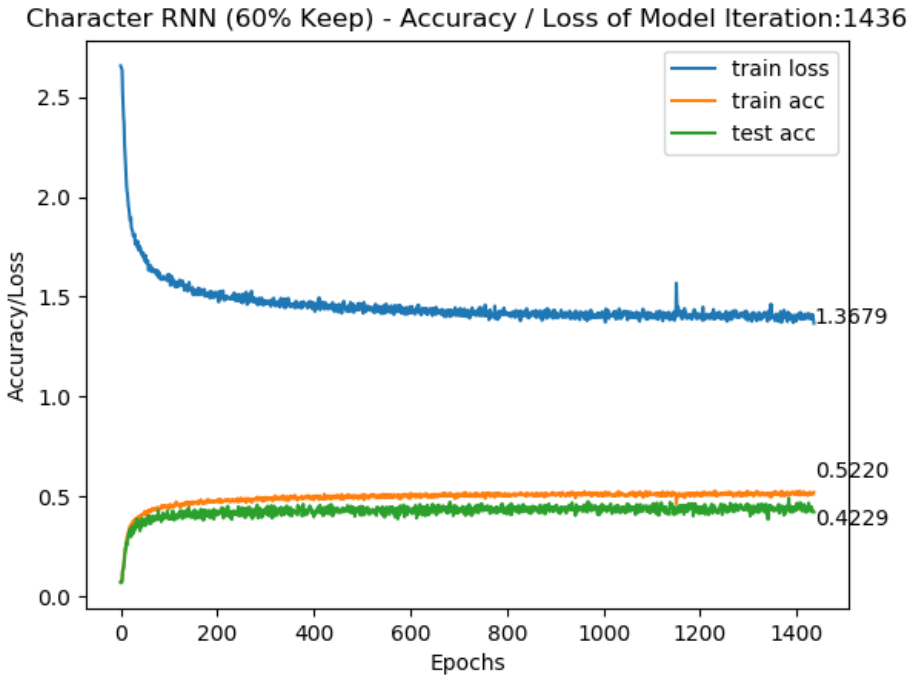

Figure 5.22 Character - Accuracy/Loss of Optimal Model with keep = 0.6 at 1436 epochs

Dropout of 0.2

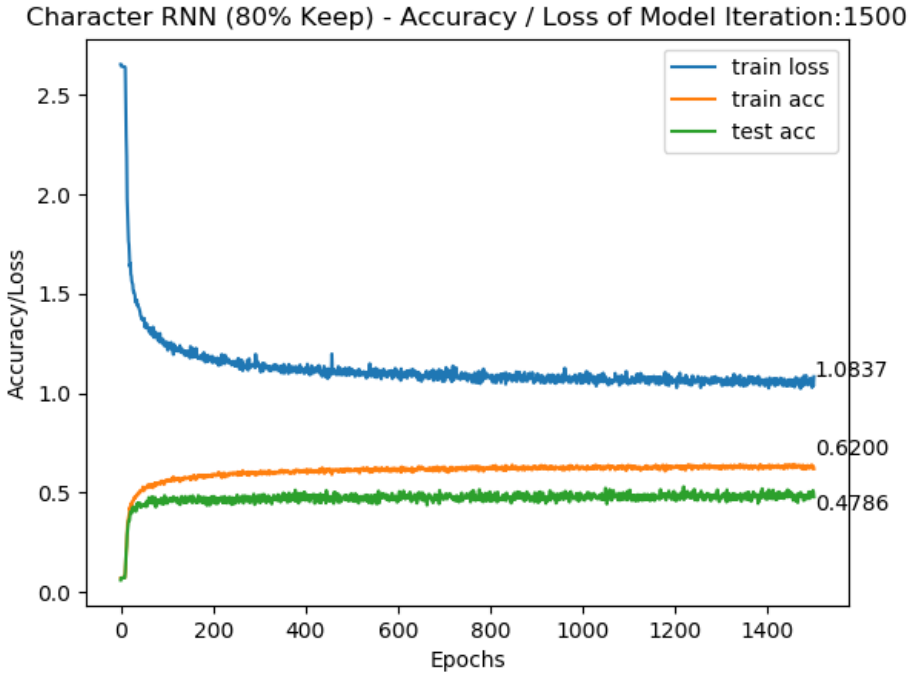

Figure 5.23 Character - Accuracy/Loss of Model with keep = 0.8 at 1500 epochs

At epochs 1471, it provides us with the lowest entropy cost of 1.0233 on the training data and the accuracy of 50.14% on the testing data.

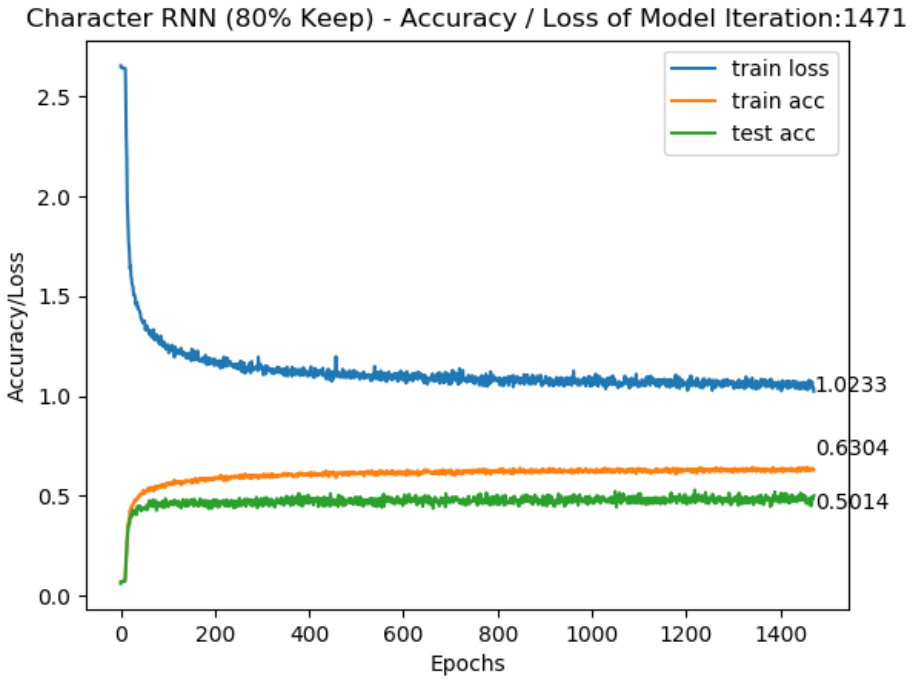

Figure 5.24 Character - Accuracy/Loss of Optimal Model with keep = 0.8 at 1471 epochs

Character RNN Dropout Percentage Comparisons (3 significant figures)					
	0	0.8	0.6	0.4	0.2
Epochs	500	1447	880	1436	1471
Train Loss	0.399	2.120	1.697	1.368	1.023
Test Accuracy	42.571%	27.714%	38.571%	42.285%	50.142%

From the table shown above, if 40% or more of each pooling layer is dropout, the overall performance of the model decreases as seen in the dropout of 80% with a test accuracy of 27.71% (~-15%), 60% with a test accuracy of 38.57% (~-14%) and 40% with a test accuracy of 42.29% (~-0.3%).

However, if less than 40% of each pooling layer is dropout, the overall performance of the model increases as seen in the dropout of 20% with a test accuracy of 50.14% (~+8%).

Dropouts are meant to encourage the network to learn a sparse representation however in this case the models are all underfitted hence not much changes will occur.

3.3.5.4 Word RNN with dropout

Dropout of 0.8

The results for the probability that each element is dropped by 20%, 40%, 60% and 80% for the word RNN is shown below.

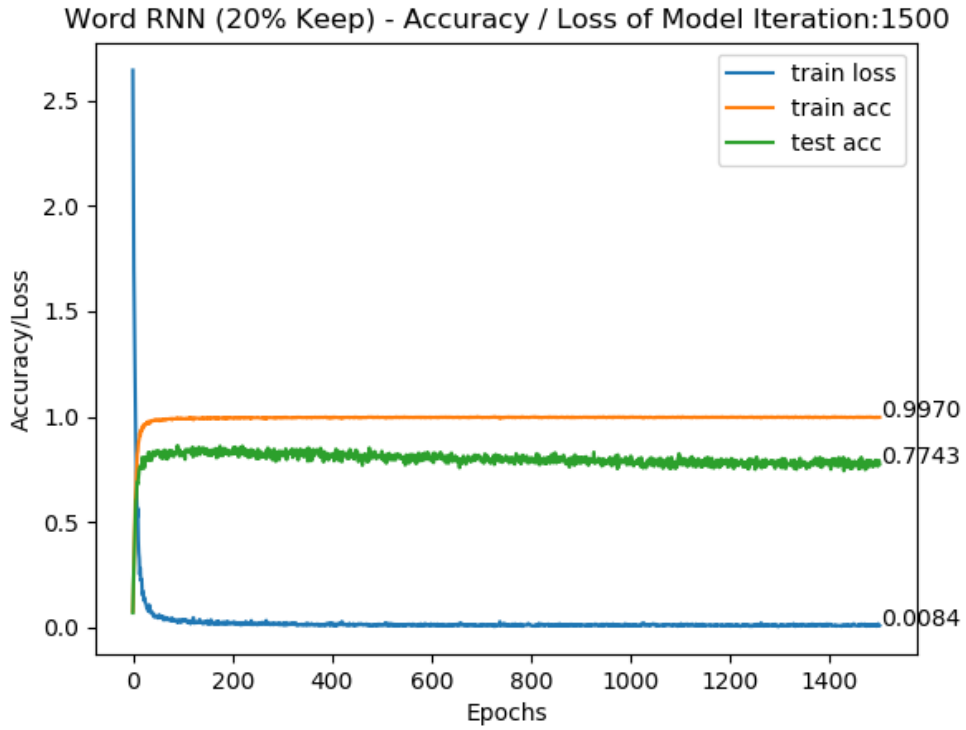

Figure 5.25 Word - Accuracy/Loss of Model with keep = 0.2 at 1500 epochs

At epochs 1296, it provides us with the lowest entropy cost of 0.00284 on the training data and the accuracy of 79.29% on the testing data.

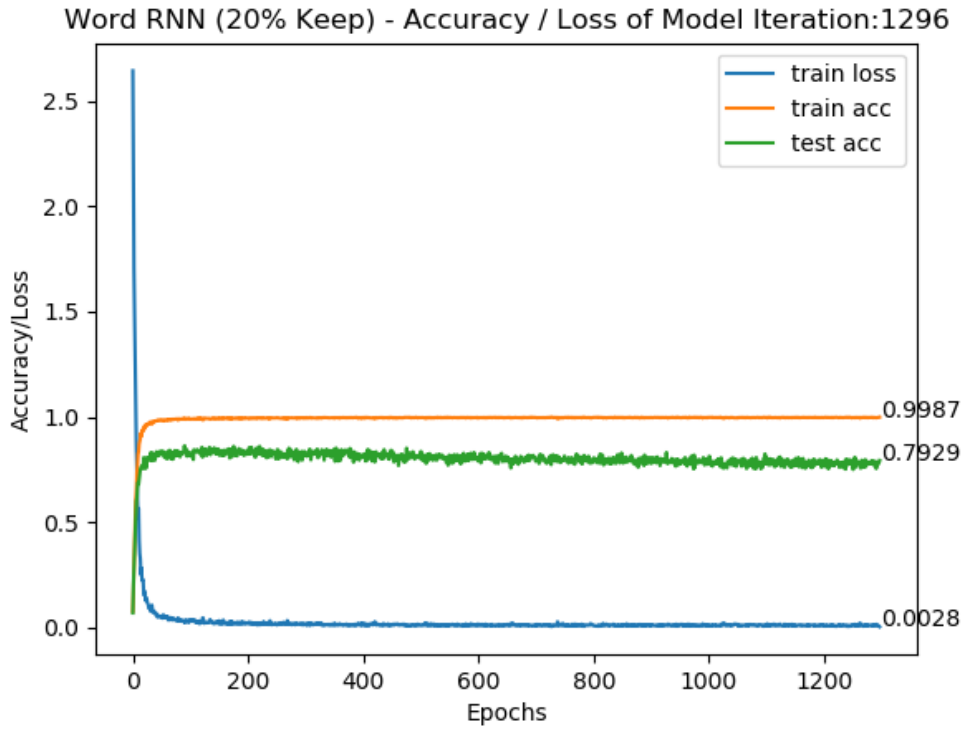

Figure 5.26 Word - Accuracy/Loss of Optimal Model with keep = 0.2 at 1296 epochs

Dropout of 0.6

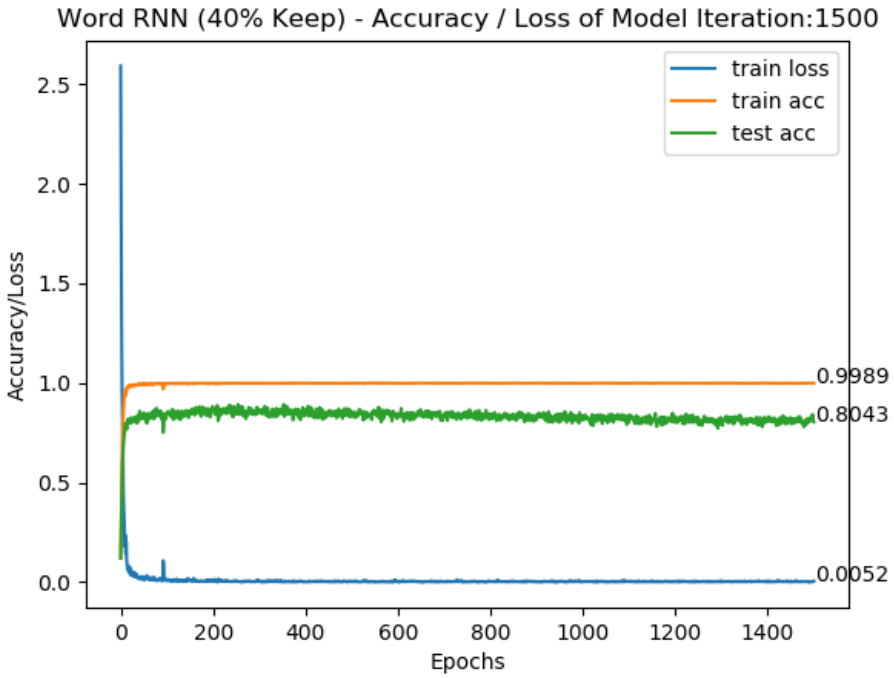

Figure 5.27 Word - Accuracy/Loss of Model with keep = 0.4 at 1500 epochs

At epochs 744, it provides us with the lowest entropy cost of 0.000404 on the training data and the accuracy of 85.14% on the testing data.

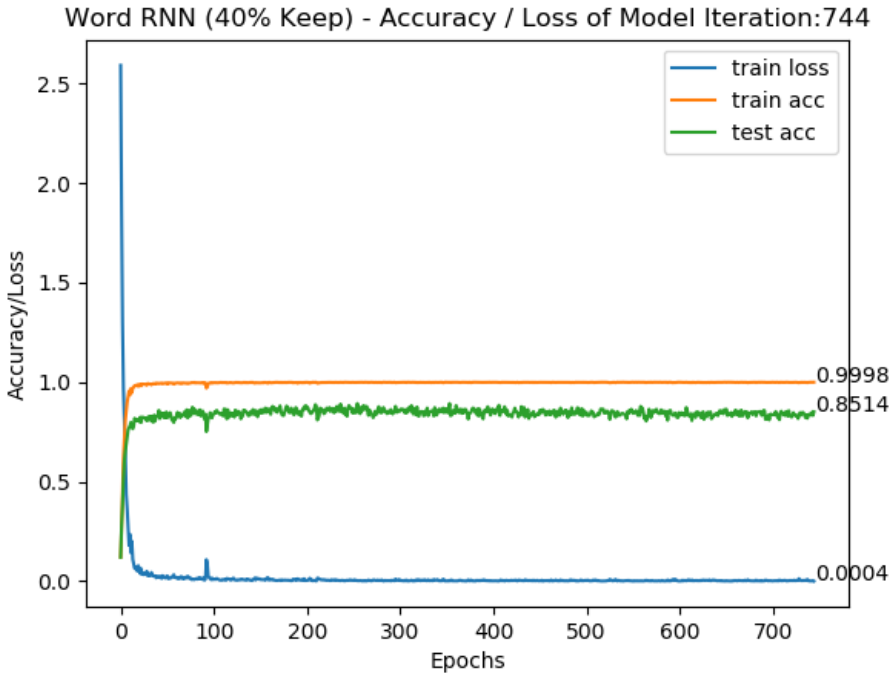

Figure 5.28 Word - Accuracy/Loss of Optimal Model with keep = 0.4 at 744 epochs

Dropout of 0.4

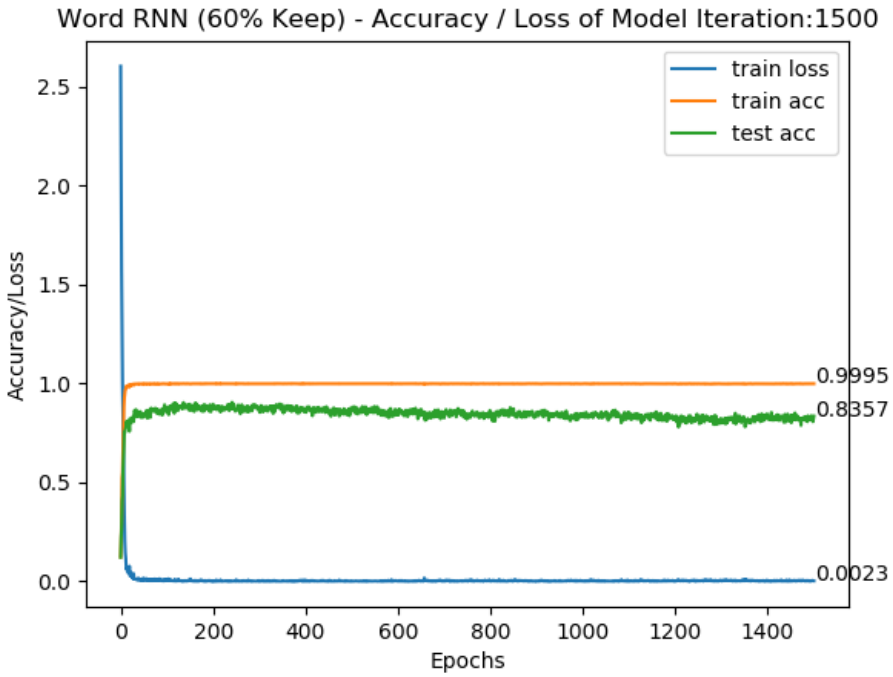

Figure 5.29 Word - Accuracy/Loss of Model with keep = 0.6 at 1500 epochs

At epochs 673, it provides us with the lowest entropy cost of 0.000131 on the training data and the accuracy of 85% on the testing data.

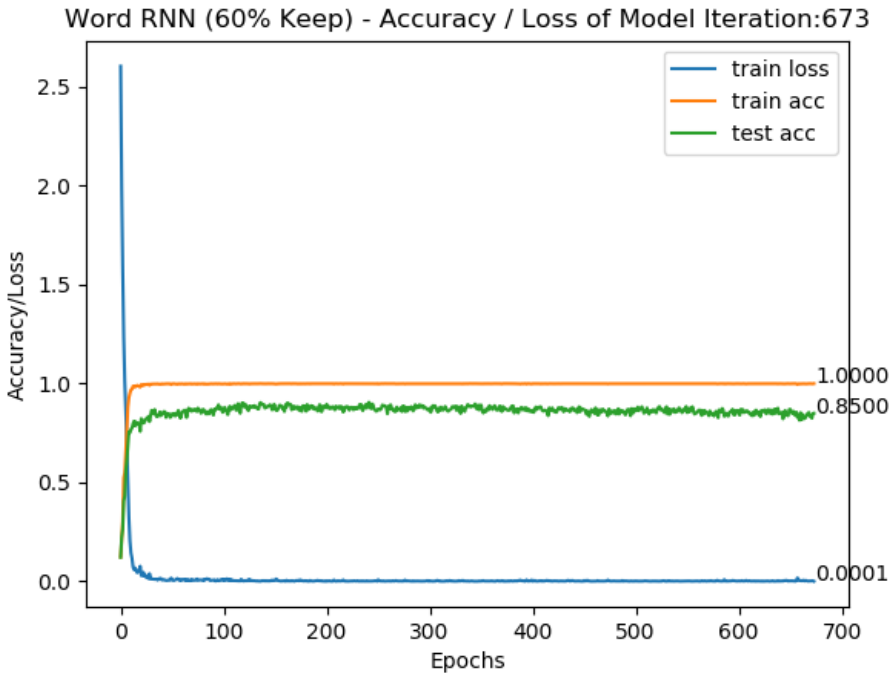

Figure 5.30 Word - Accuracy/Loss of Optimal Model with keep = 0.6 at 673 epochs

Dropout of 0.2

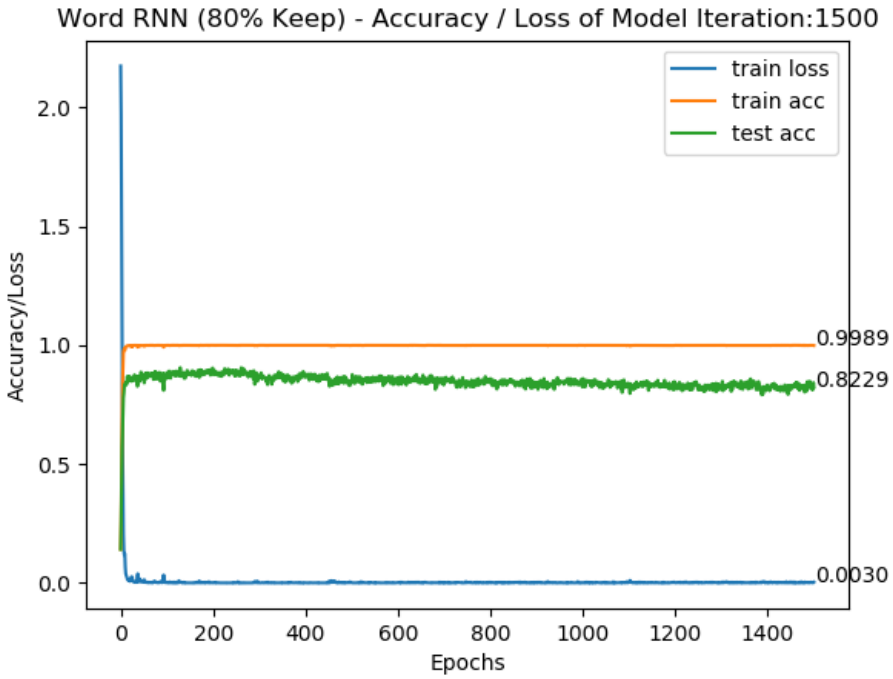

Figure 5.31 Word - Accuracy/Loss of Model with keep = 0.8 at 1500 epochs

At epochs 434, it provides us with the lowest entropy cost of 0.0000159 on the training data and the accuracy of 86.71% on the testing data.

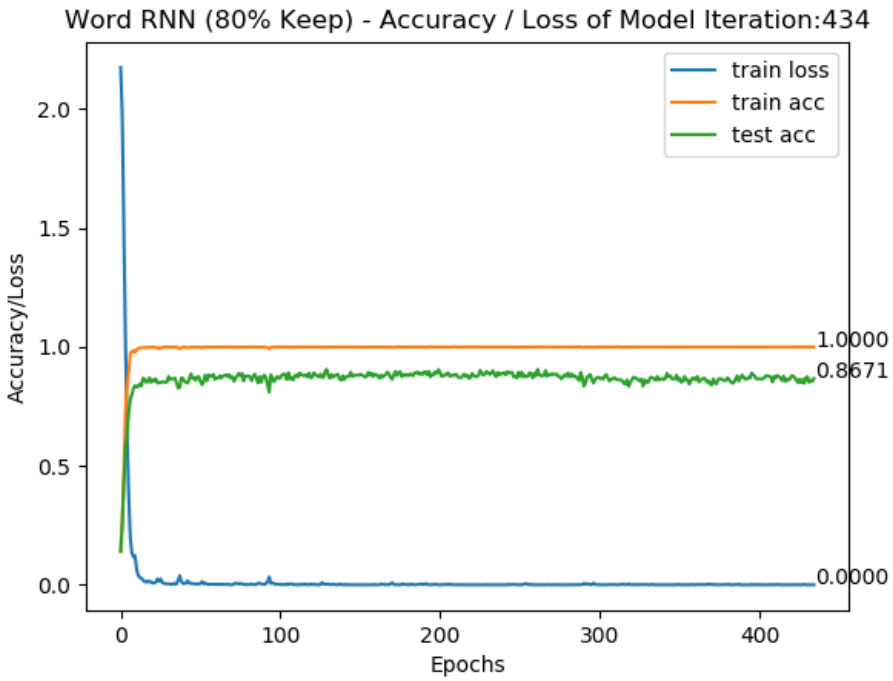

Figure 5.31 Word - Accuracy/Loss of Optimal Model with keep = 0.8 at 434 epochs

Word RNN Dropout Percentage Comparisons (3 significant figures)					
	0	0.8	0.6	0.4	0.2
Epochs	838	1296	744	673	434
Train Loss	3.576E-9	2.84E-3	4.04E-4	1.31E-4	1.59E-05
Test Accuracy	84%	79.286%	85.142%	85%	86.714%

From the table shown above, if 80% or more of each pooling layer is dropout, the overall performance of the model decreases as seen in the dropout of 80% with a test accuracy of 79.29% (~-5%).

However, if less than 80% of each pooling layer is dropout, the overall performance of the model increases as seen in the dropout of 60% with a test accuracy of 85.14% (~+1%), 40% with a test accuracy of 85% (~+1%) and 20% with a test accuracy of 86.71% (~+2%).

Dropouts are meant to encourage the network to learn a sparse representation and in this case by removing half of the nodes used for learning would allow the model to learn better by providing noisier environment to better generalize the model.

3.3.6 Question 6

3.3.6.1 Vanilla Layer Experiment

To determine the optimal model, a total of 1500 epochs of training have been tested with checkpoints of 100 epochs for both the vanilla and LSTM layer RNN.

The result shown below is for 1500 epochs for a **vanilla character RNN** layer:

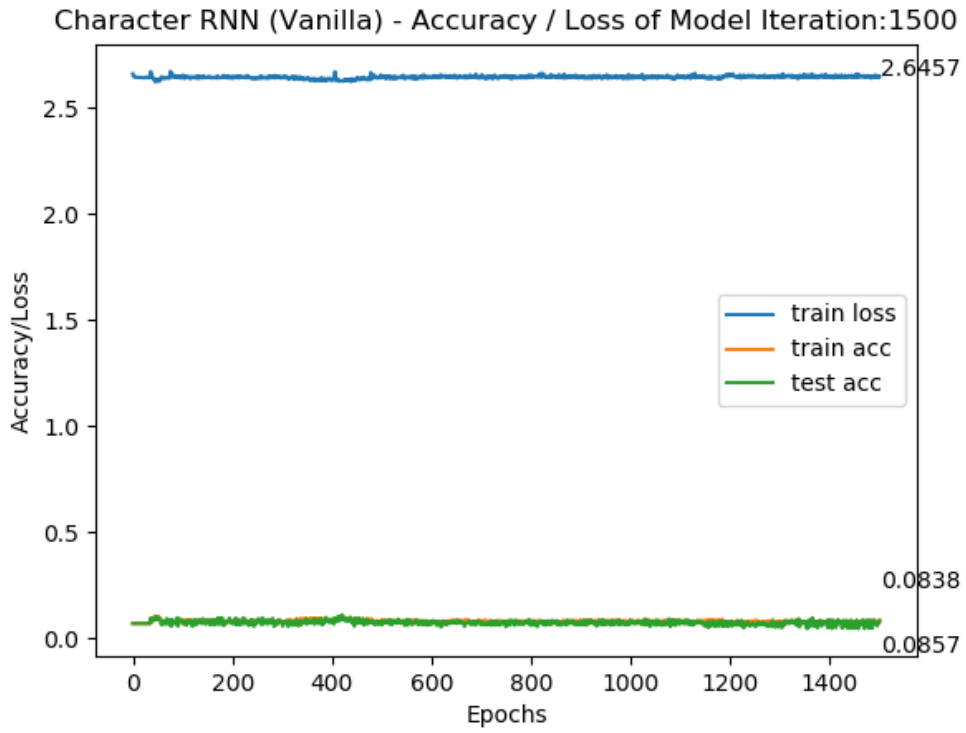

Figure 6.1 Character - Vanilla Accuracy/Loss of Model at 1500 epochs

The lowest training loss achieved is 2.620 with a test accuracy of 10.14% during epoch 45. The optimal model is shown below.

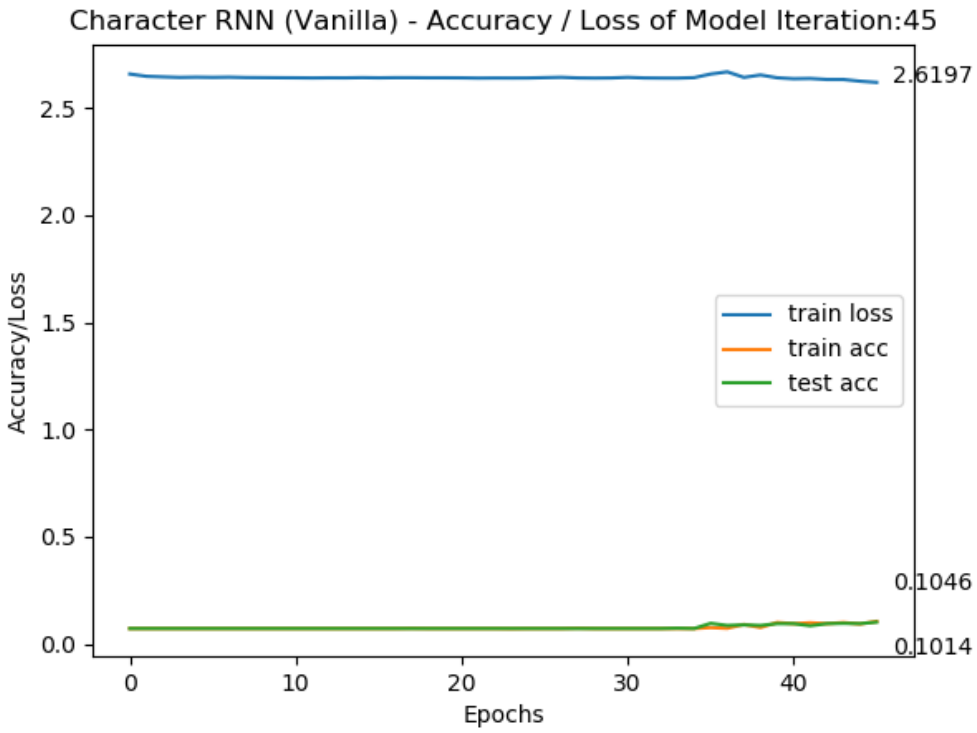

Figure 6.2 Character – Vanilla Accuracy/Loss of Optimal Model at 45 epochs

The result shown below is for 1500 epochs for a **vanilla word RNN** layer:

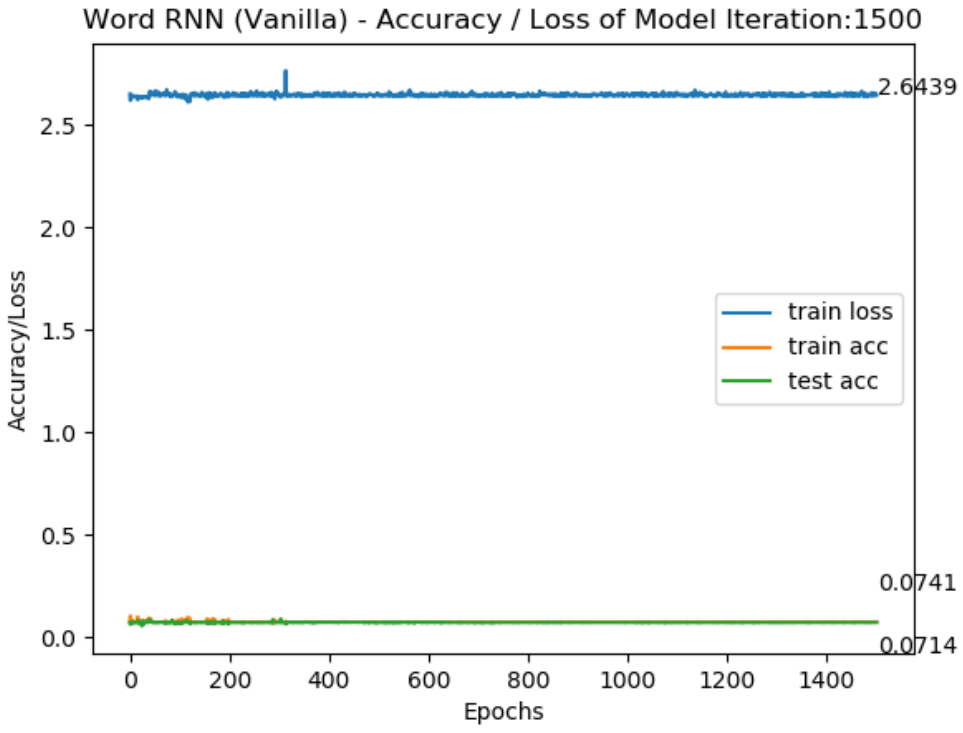

Figure 6.3 Word – Vanilla Accuracy/Loss of Model at 1500 epochs

The lowest training loss achieved is 2.610 with a test accuracy of 7.29% during epoch 117. The optimal model is shown below.

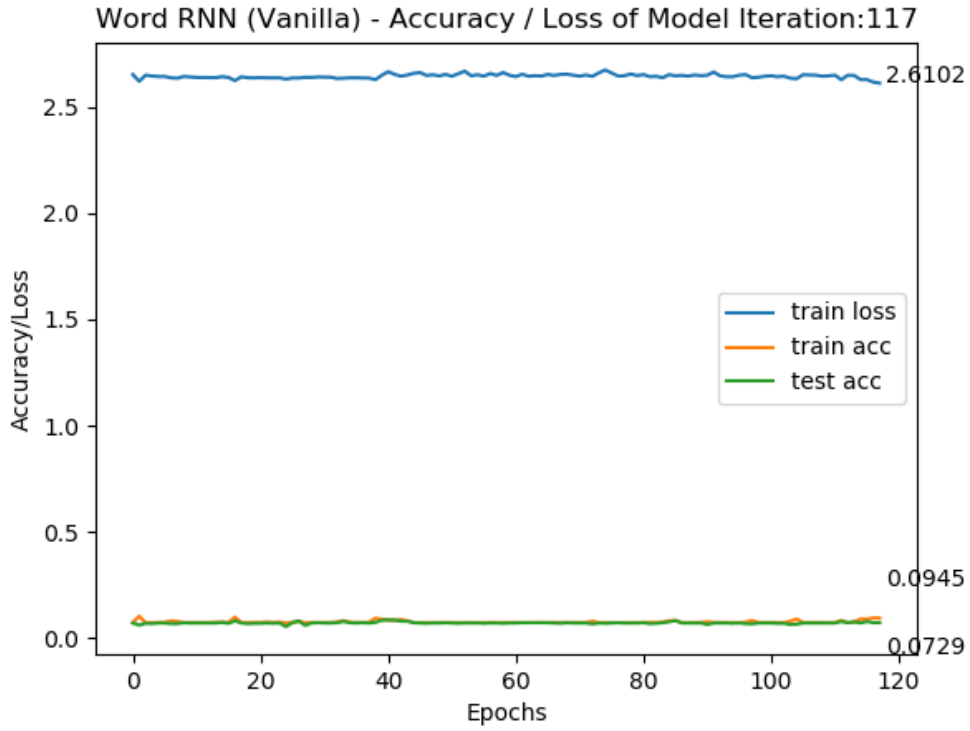

Figure 6.4 Word – Vanilla Accuracy/Loss of Optimal Model at 117 epochs

3.3.6.2 LSTM Layer Experiment

The result shown below is for 1500 epochs for a **LSTM character RNN** layer:

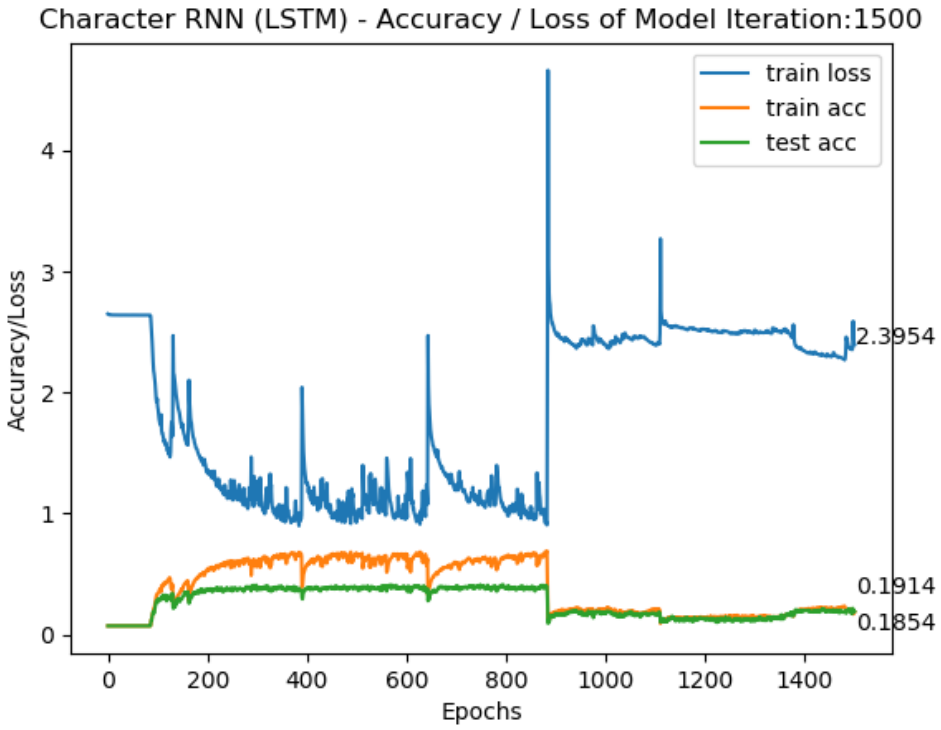

Figure 6.5 Character – LSTM Accuracy/Loss of Model at 1500 epochs

The lowest training loss achieved is 0.8985 with a test accuracy of 38.43% during epoch 384. The optimal model is shown below.

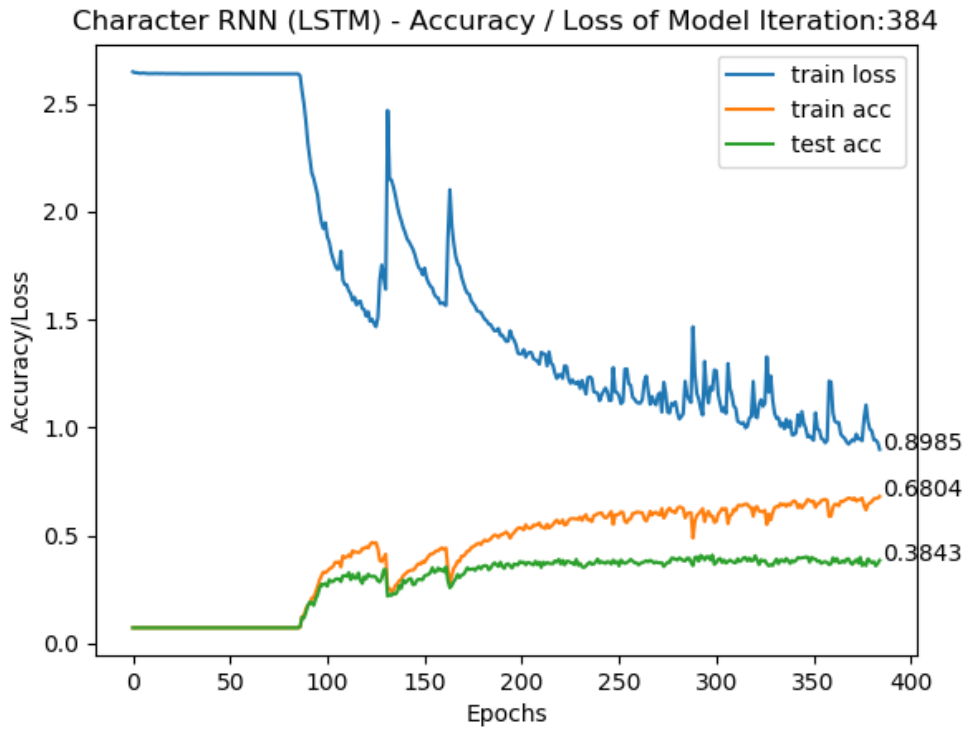

Figure 6.6 Character – LSTM Accuracy/Loss of Optimal Model at 384 epochs

The result shown below is for 1500 epochs for a **LSTM word RNN** layer:

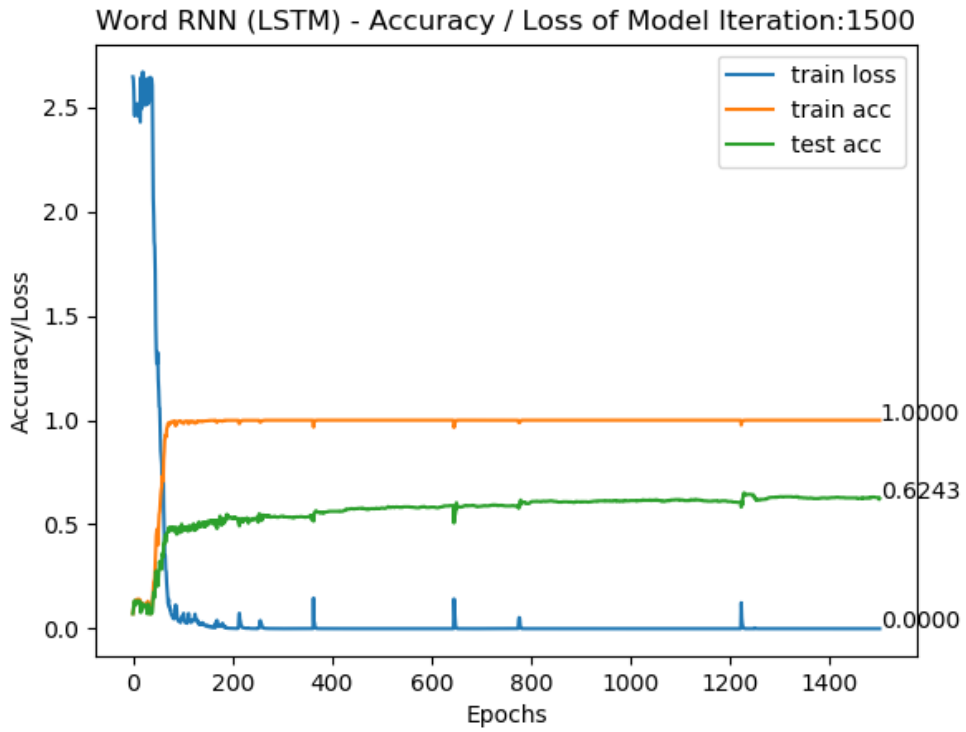

Figure 6.7 Word – LSTM Accuracy/Loss of Model at 1500 epochs

The lowest training loss achieved is 1.38E-08 with a test accuracy of 60.57% during epoch 1220. The optimal model is shown below.

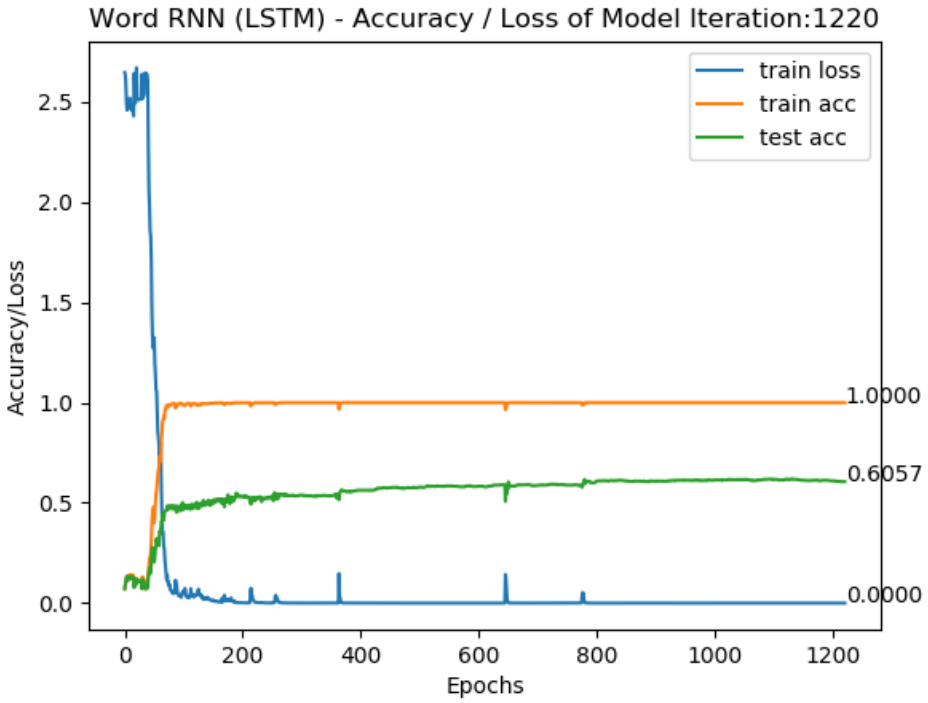

Figure 6.8 Word – LSTM Accuracy/Loss of Optimal Model at 1220 epochs

Character Based Comparisons Between Layers (3 significant figures)			
	GRU	Vanilla	LSTM
Epochs	500	45	384
Train Loss	0.399	2.620	0.899
Test Accuracy	42.571%	10.143%	38.428%

Word Based Comparisons Between Layers (3 significant figures)			
	GRU	Vanilla	LSTM
Epochs	838	117	1220
Train Loss	3.576E-9	2.610	1.38E-08
Test Accuracy	84%	7.286%	60.571%

For this experiment, different layers are used in both the character and word based RNN to determine which provides the best accuracy.

In the table for the comparison of character RNN layers, the ranking in terms of training loss and test accuracy is as follows:

1. GRU
2. LSTM
3. Vanilla

In the table for the comparison of word RNN layers, the ranking in terms of training loss and test accuracy is as follows:

1. GRU
2. LSTM
3. Vanilla

As expected, the vanilla layer would perform the worst among both classification as it uses a basic RNN cell which is unable to overwrite its content at each time-step. Whereas the GRU and LSTM can determine whether the existing memory should be kept.

Between the GRU and LSTM models, the GRU model performing better could be due to the fact that the training data contains primarily short distance dependencies where the LSTM model constantly tries to capture potential long distance dependency which eventually hurts its performance.

3.3.6.3 2-Layers Experiment

To determine the optimal model, a total of 1500 epochs of training have been tested with checkpoints of 100 epochs for both the character and word RNN.

The result shown below is for 1500 epochs for 2-layer Character RNN:

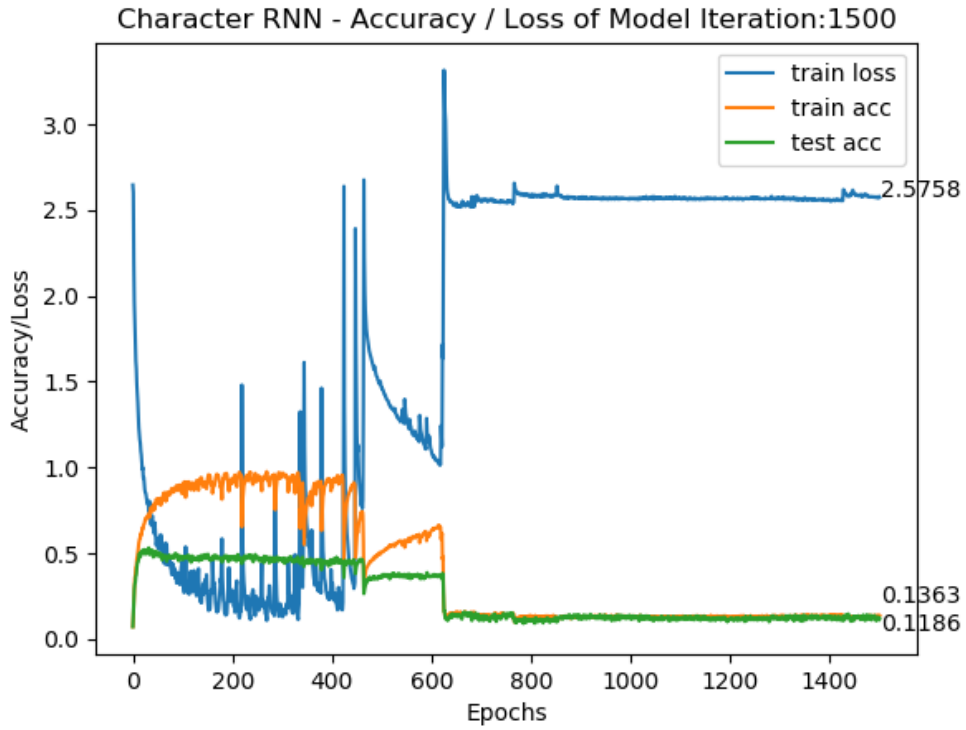

Figure 6.9 2-Layered Character Accuracy/Loss of Model at 1500 epochs

The lowest training loss achieved is 0.1077 with a test accuracy of 46.43% during epoch 269. The optimal model is shown below:

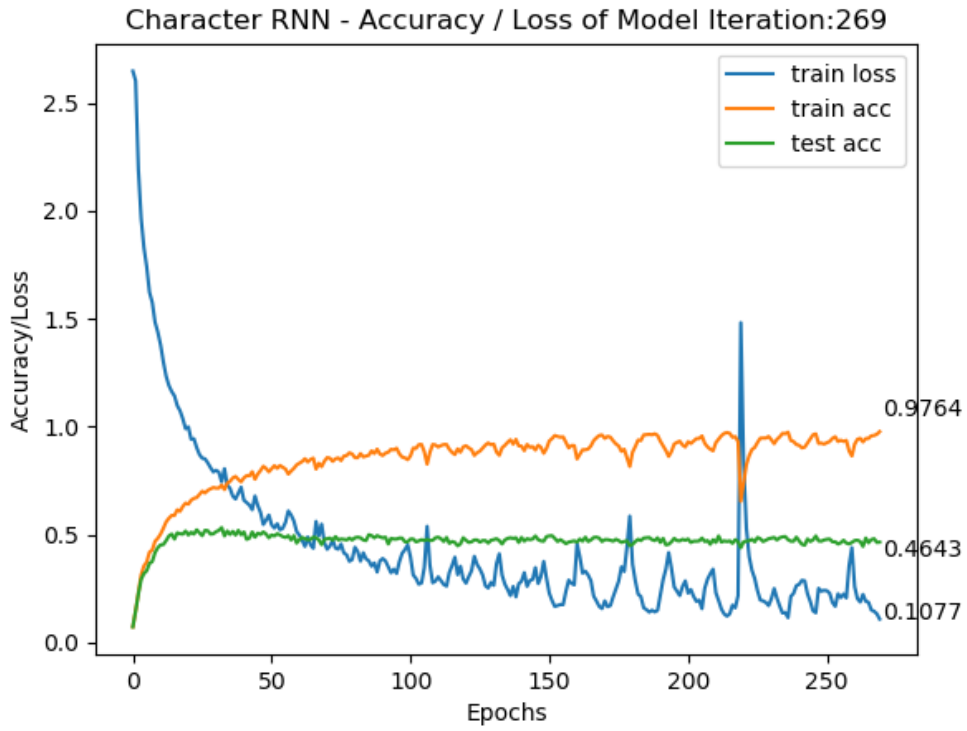

Figure 6.10 2-Layered Character Accuracy/Loss of Optimal Model at 269 epochs

Character Based Comparisons (3 significant figures)		
	1-Layer	2-Layers
Epochs	500	269
Train Loss	0.399	0.108
Test Accuracy	42.571%	46.429%

For this experiment, an additional hidden layer is added to test whether it affects the accuracy of the character RNN model.

As shown in the table, the RNN model with 2 hidden layers performs better than the 1 hidden layer model with about an increase of 4% in test accuracy and a decrease of 0.2 in loss.

However, both models are severely overfitted with the 2 hidden layer RNN having a training accuracy of around 98% whereas the test accuracy is only 46.43% resulting in a difference of around 50%.

The result shown below is for 1500 epochs for 2-layer Word RNN:

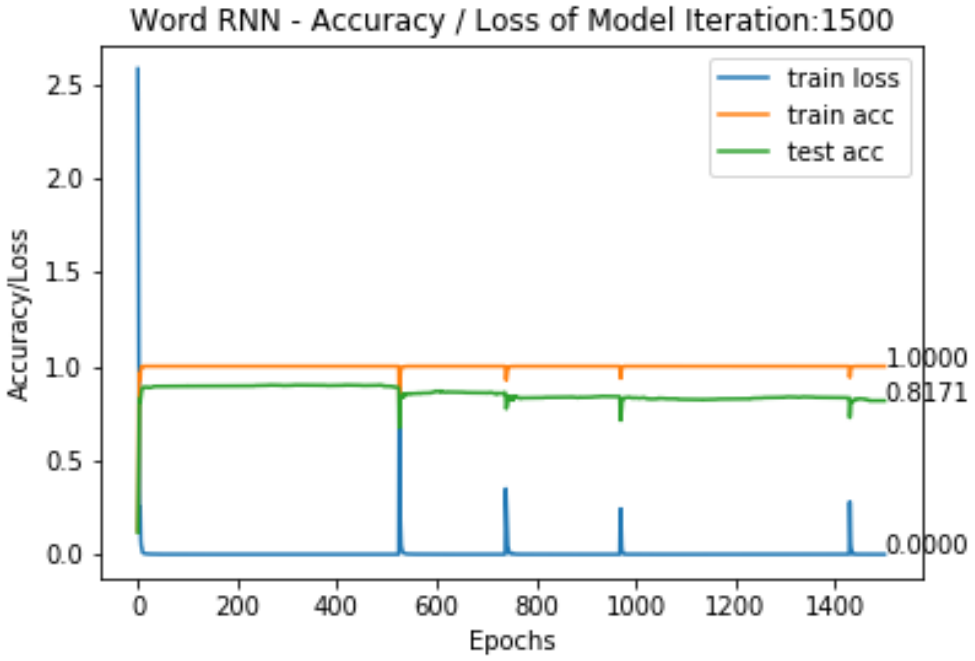

Figure 6.11 2-Layered Word Accuracy/Loss of Model at 1500 epochs

The lowest training loss achieved is 8.770E-09 with a test accuracy of 83.14% during epoch 1428. The optimal model is shown below.

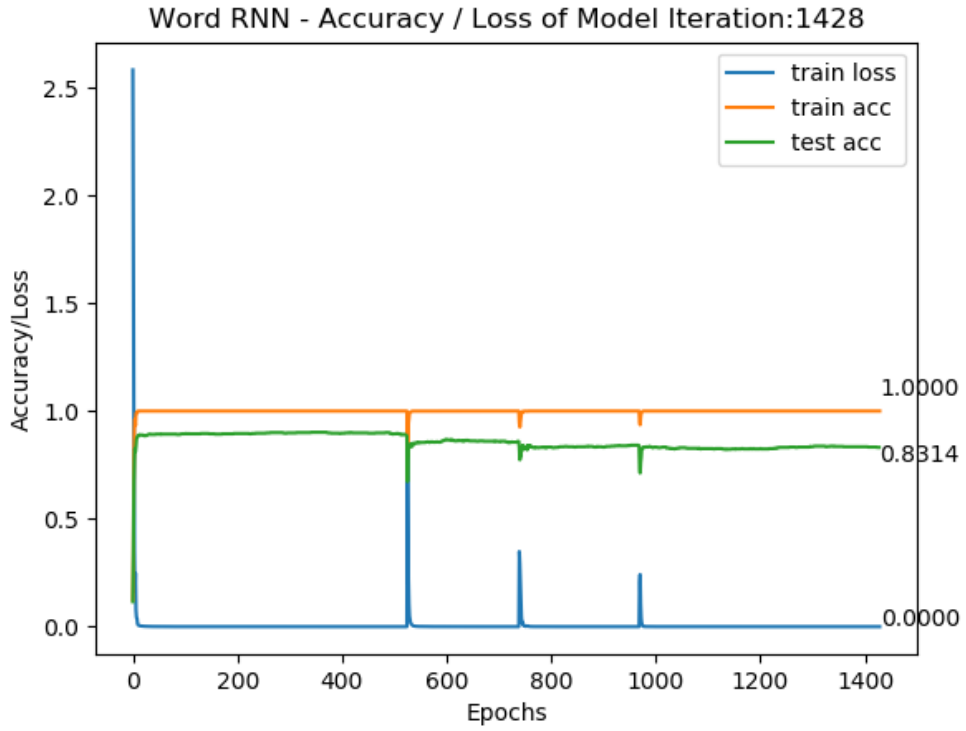

Figure 6.12 2-Layered Word Accuracy/Loss of Model at 1428 epochs

Word Based Comparisons (3 significant figures)		
	1-Layer	2-Layers
Epochs	838	1428
Train Loss	3.576E-9	8.770E-9
Test Accuracy	84%	83.142%

For this experiment, an additional hidden layer is added to test whether it affects the accuracy of the word RNN model.

As shown in the table, the RNN model with 1 hidden layer performs better than the 2 hidden layer model with about an increase of 1% in test accuracy and a decrease of 5E-9 in loss.

Despite the 1 layer model performing better, both models seem to have generalized well, and should be considered.

Normally, we would expect a deeper network to perform much better which in this case still holds true as the differences between the two models are very minor and if the RNN with 2 layers were allowed to be trained longer than 1500 epochs, it may outperform and be better than the RNN with only 1 layer.

3.3.6.4 Gradient Clipping Experiment

To determine the optimal model, a total of 1500 epochs of training have been tested with checkpoints of 100 epochs for both the character and word RNN.

The result shown below is for 1500 epochs for Character RNN:

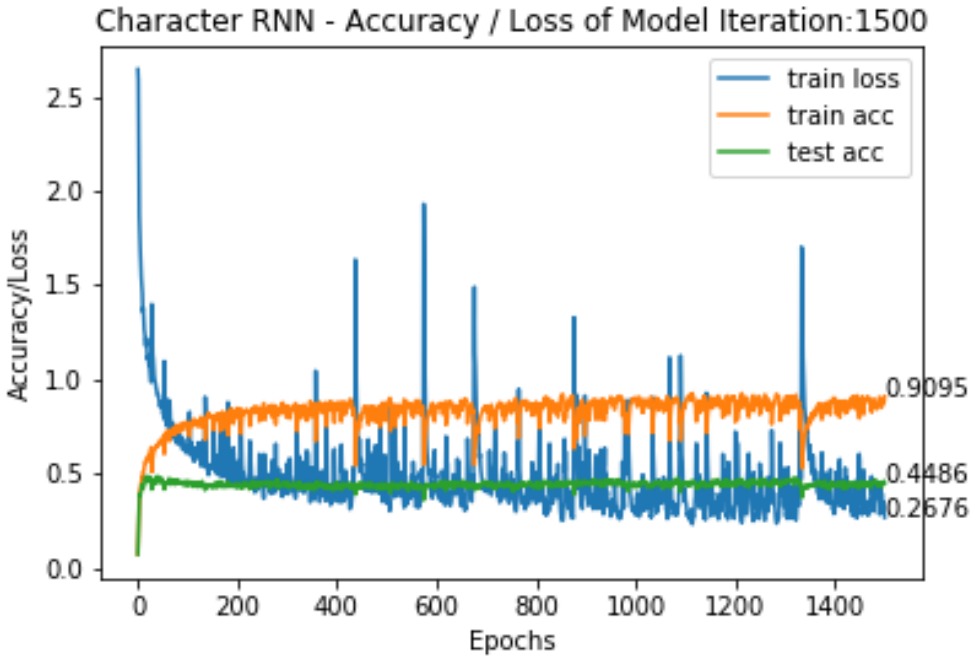

Figure 6.13 Character Accuracy/Loss of Model with clipping=2 at 1500 epochs

The lowest training loss achieved is 0.2316 with a test accuracy of 45.14% during epoch 1114. The optimal model is shown below.

Figure 6.14 Character Accuracy/Loss of Optimal Model with clipping=2 at 1114 epochs

Character Based Comparisons (3 significant figures)		
	W/o Clipping	Clipping = 2
Epochs	500	1114
Train Loss	0.399	0.232
Test Accuracy	42.571%	45.143%

For this experiment, gradient clipping with threshold of 2 have been added to the RNN training process of the character RNN model.

As shown in the table, the RNN model with clipping performs better than the model without clipping with a lower loss of around 0.1 and a higher test accuracy of around 3%.

As the optimal RNN model in question 3 can be seen with numerous exploding gradients, by applying gradient clipping, it should help the model to perform better which holds true in this case.

The result shown below is for 1500 epochs for Word RNN:

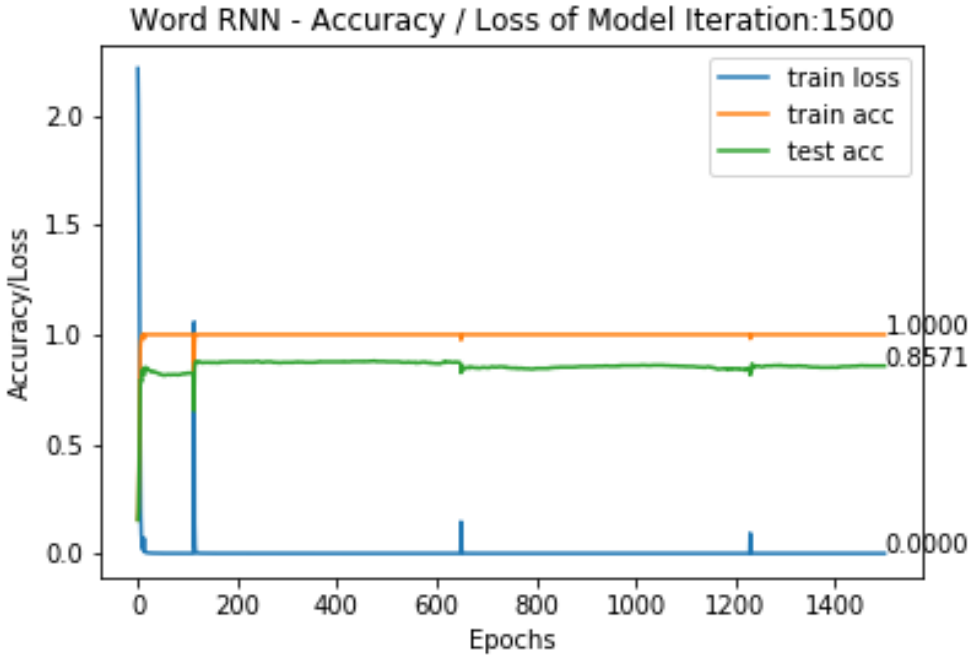

Figure 6.15 Word Accuracy/Loss of Model with clipping=2 at 1500 epochs

The lowest training loss achieved is 2.7034E-09 with a test accuracy of 84.43% during epoch 1179. The optimal model is shown below.

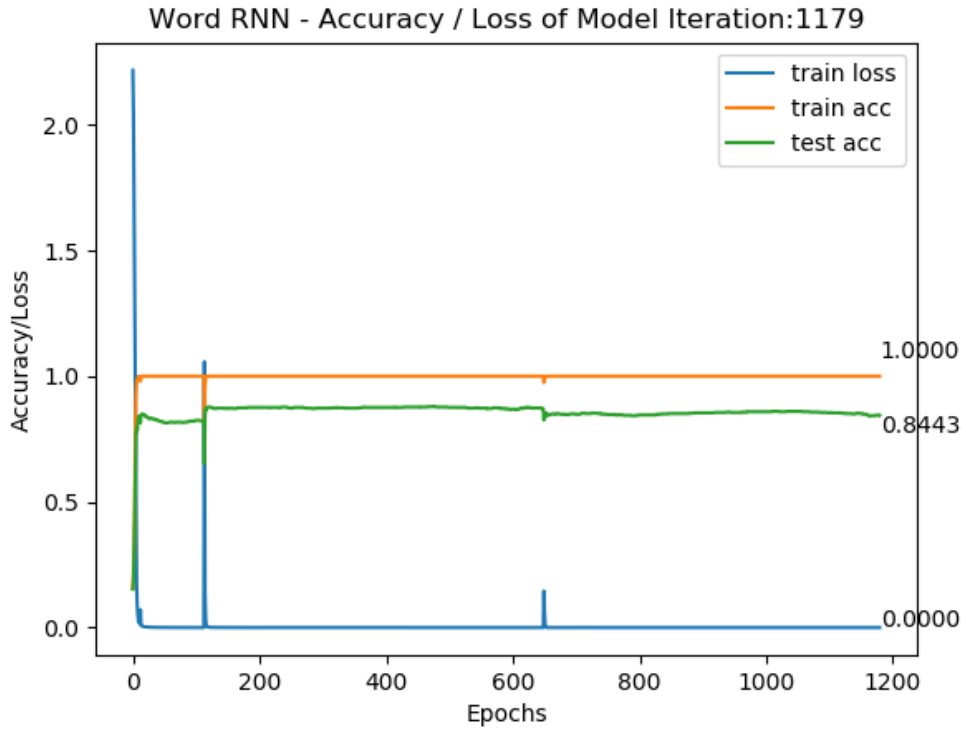

Figure 6.16 Word Accuracy/Loss of Optimal Model with clipping=2 at 1179 epochs

Word Based Comparisons (3 significant figures)		
	W/o Clipping	Clipping = 2
Epochs	838	1179
Train Loss	3.576E-09	2.703E-09
Test Accuracy	84%	84.428%

For this experiment, gradient clipping with threshold of 2 have been added to the RNN training process of the word RNN model.

As shown in the table, the RNN model with clipping performs better than the model without clipping with a lower loss of around 1E-9 and a higher test accuracy of around 0.4%.

The purpose of gradient clipping is to help RNN models with long term dependencies to manage gradients that may grow very large or very small which may cause models to diverge.

Hence, for this case, the above statement holds true by showing that a model with clipping performs better than one that does not have clipping.

4 Conclusions

In this report, we addressed two different types of problems, object recognition and text classification problems. We discussed about the methods and detailed the results of our experiments and is summarized below:

4.1 Part A: Object Recognition Problem

- Optimal number of feature maps – 214 for both C1 and C2.
- Optimal model from parts 1 to 3 – 50% Dropout with 214 filter at C1 and C2.

4.2 Part B: Text Classification Problem

- Character CNN without / with dropouts – With dropouts < 50%
- Character RNN without / with dropouts – With dropouts < 40%
- Word CNN without / with dropouts – With dropouts < 40%
- Word RNN without / with dropouts – With dropouts < 80%
- Character RNN with GRU / Vanilla / LSTM layer – GRU layer
- Word RNN with GRU / Vanilla / LSTM layer – GRU layer
- Character RNN single / double layer – Double layer
- Word RNN single / double layer – Double layer
- Character RNN without / with clipping – With clipping
- Word RNN without / with clipping – With clipping

Finally, we shall discuss on our takeaways and conclusions on findings below:

- The number of epochs used for grid search must be reduced in a manner that will not affect the results – too low and it will bias simpler models.
- Determine the endpoints of the grid search first before determining the step size and other parameters.
- Other optimizers other than gradient descent converge faster but not necessarily better
- Adam optimizer tends to have unstable results and vanishing gradients
- Models with dropouts will generally generalize better than a model without it.
- Character based models tend to have very high losses and low accuracies which may be due to underfitting.
- LSTM and GRU units have proven the need to forget previous hidden states and determine when to update hidden states given new information in training temporal dependent data.
- Adding clipping to RNN models significantly helps to deal with issues such as exploding gradients and vanishing gradients which may move parameters away from the true minimum.