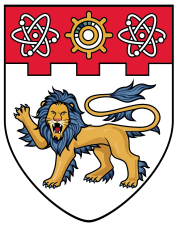


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042 Neural Network

Project 2

Completed by:
Kyle Huang Junyuan (U1721717G)
Nelson Ko Ming Wei (U1721410B)

Table of contents

Introduction

Project 2 is split into two sections - Part A is completed by Kyle Huang Junyuan while Part B is completed by Nelson Ko Ming Wei.

Part A: Object Recognition

Part A deals with the CIFAR-10 dataset which contains 10,000 RGB colour images of size 32x32 and their labels from 0 to 9. A Convolutional Neural Network (CNN) will be designed to accurately predict objects in these training samples. Subsequently, testing is performed on 20,000 test samples to evaluate model accuracy.

Part B: Text Classification

Part B deals with first paragraphs collected from Wikipage entries and their corresponding labels about their category. Both CNN and Recurrent Neural Network (RNN) will be designed to accurately classify texts at the word and character level of paragraphs in a training dataset. Subsequently, testing is performed on 700 test samples to evaluate model accuracy.

Methods

A Convolutional Neural Network (CNN) is a Deep Learning algorithm that can be trained to identify various aspects or objects of an input image. It consists of multiple convolution layers, pooling layers and a fully-connected layer.

A Recurrent Neural Network (RNN) is a class of artificial neural network that allows operation across a sequence of vectors. These include sequences in the input, output or in most cases, both.

Convolution Layer

The convolution layer makes use of multiple filters (kernels) that serves to detect patterns in sub-regions of an input image. Each filter is defined with weights and a window size which will then be used to 'sweep' across the input image with a pre-defined stride value. The objective of this layer is to extract high-level features from the input image.

Embedding layer

A representation of text where words that have the same meaning have a similar representation. Embedding layer which stores a lookup table to map the words represented by numeric indexes to their dense vector representation. It is applied to the input before feeding into the RNN.

Max/Average Pooling Layer

The pooling layer is responsible for reducing the dimensionality of the output from the convolution layer. Without the pooling layer, a large amount of computational power will be required to process the data. In addition, it can be useful for extracting dominant features which are unaffected by varying orientation or position of the feature.

In general, there exist two types of pooling - Max Pooling and Average Pooling. Max Pooling will return the maximum value of the portion of the image covered by the pooling kernel. In contrast, Average Pooling will return the average of all values of the portion of the image covered by the pooling kernel.

VALID/SAME Padding

With VALID padding, the window size of the filter is being applied without padding. That is, in the event the applied filter is exceeding the input image width, only the remaining areas of the input image covered by the filter will be taken into account.

With SAME padding, the window size of the filter will apply zero-padding where necessary. That is, in the event the applied filter exceeds the input image, zeros will be added together with the areas of the input image that are covered by the filter.

Fully-connected layer

A fully-connected layer helps to combine all the high-level features represented at the output of the convolution layers into a suitable form to be fed into a neural network. This is also known as flattening the image into a single column vector.

Gradient Clipping

Gradient Clipping is a regularization technique to prevent exploding gradients. Exploding gradient can occur when the gradient becomes too large and error gradients accumulate, resulting in an unstable network and unable to learn. By clipping the gradient between two numbers to prevent it from getting too large.

Dropouts

Dropout is a technique used to prevent a network from overfitting on training data. The dropout give a probability to deactivate neurons in a particular layer.

Experiments and Results

Part A: Object Recognition

Question 1

Part A of Project 2 requires us to design a network by using mini-batch gradient descent learning with batch size = 128 and the learning rate of 0.001. The images are scaled progressively in the following order as summarised:

1. Convolution Layer C1 (50x24x24):
    a. 50 filters
    b. Window size: 9x9
    c. Padding: VALID
    d. ReLU neurons
2. Pooling Layer S1 (50x12x12):
    a. Max pooling layer
    b. Window size: 2x2
    c. Stride: 2
    d. Padding: VALID
3. Convolution Layer C2 (60x8x8):
    a. 60 filters
    b. Window size: 5x5
    c. Padding: VALID
    d. ReLU neurons
4. Pooling Layer S2 (60x4x4):
    a. Max pooling layer
    b. Window size: 2x2
    c. Stride: 2
    d. Padding: VALID
5. Flatten (60x4x4 = 960)
6. A fully connected layer F3 (960 -> 300):
    a. Size: 300
7. Softmax layer (300 -> 10):
    a. Size: 300

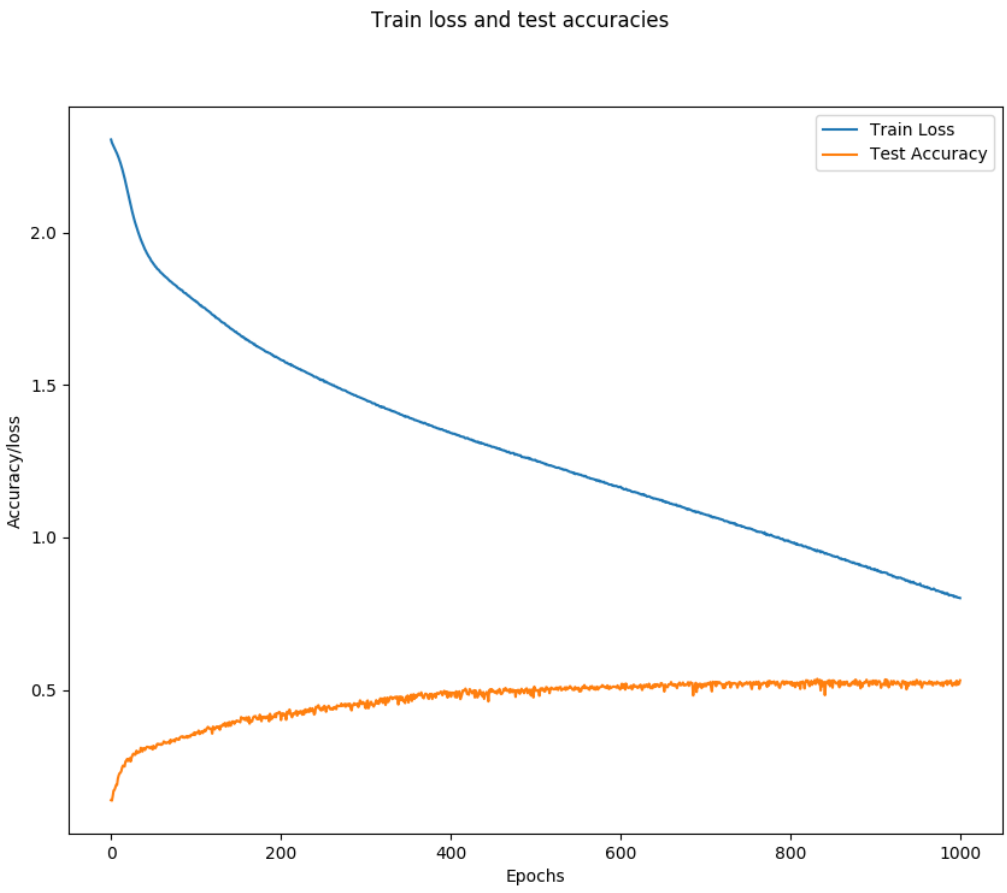


Figure 1: Gradient Descent Training Loss and Test Accuracy against Epochs

In the above figure 1, the training loss and test accuracy are illustrated against the number of epochs. As the test accuracy converges around the 800th epoch, it can be observed that the training loss continues to decrease at a steady rate. This suggests that any further training to the model might result in overfitting as the test accuracy has already converged.

The following figure 2 and figure 7 are two test patterns along with their respective convolution and pooling feature maps illustrated in figure 3 to figure 6 and figure 8 to figure 11.

Figure 2: Test Pattern 1

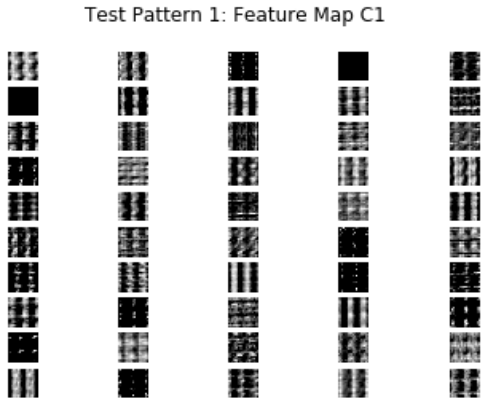


Figure 3: C1 of Test Pattern 1

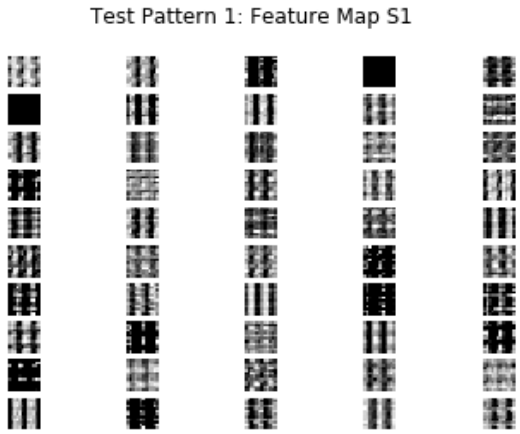


Figure 4: S1 of Test Pattern 1

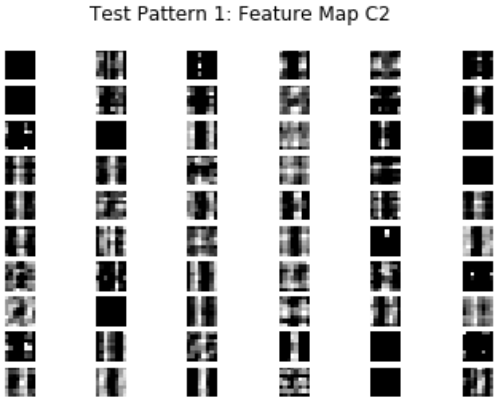


Figure 5: C2 of Test Pattern 1

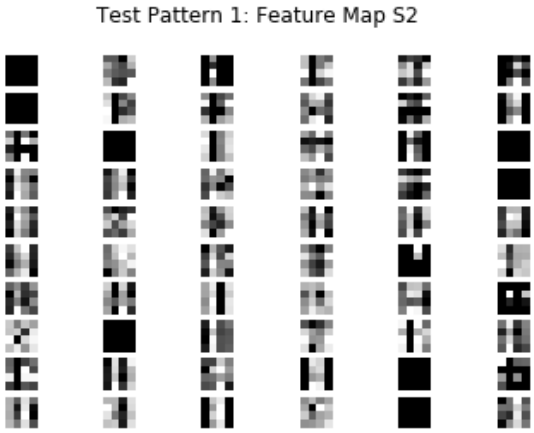


Figure 6: S2 of Test Pattern 1

Figure 7: Test Pattern 2

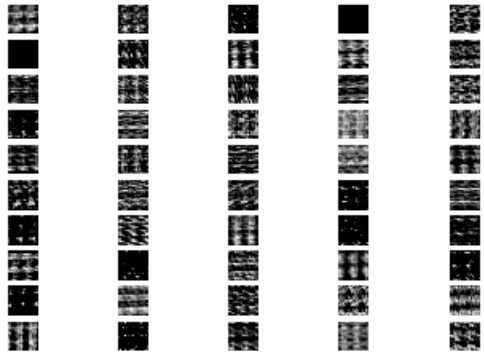


Figure 8: C1 of Test Pattern 2

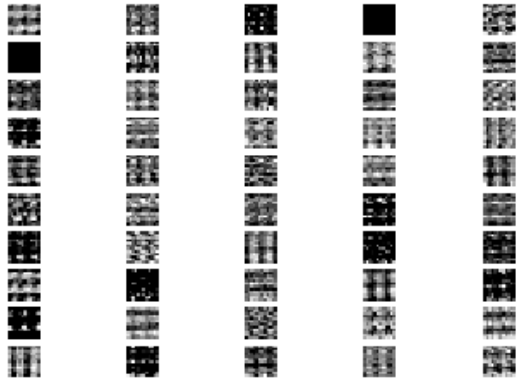


Figure 9: S1 of Test Pattern 2

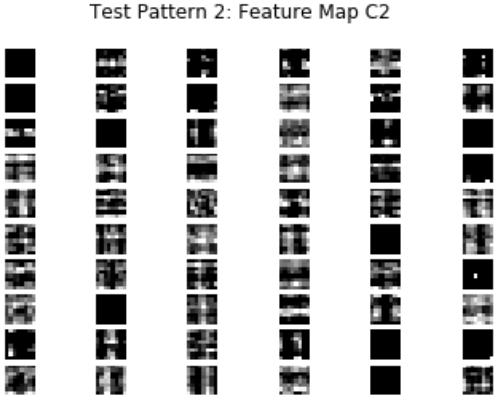


Figure 10: C2 of Test Pattern 2

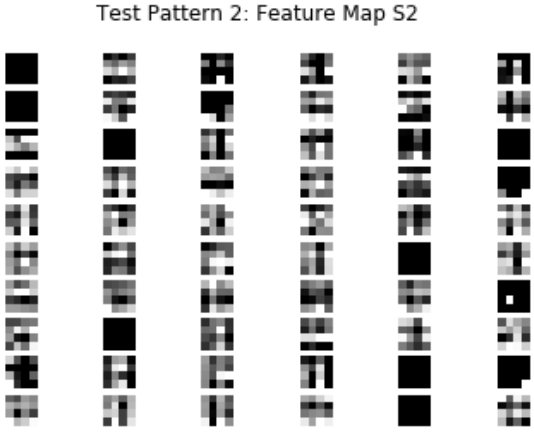


Figure 11: S2 of Test Pattern 2

Question 2

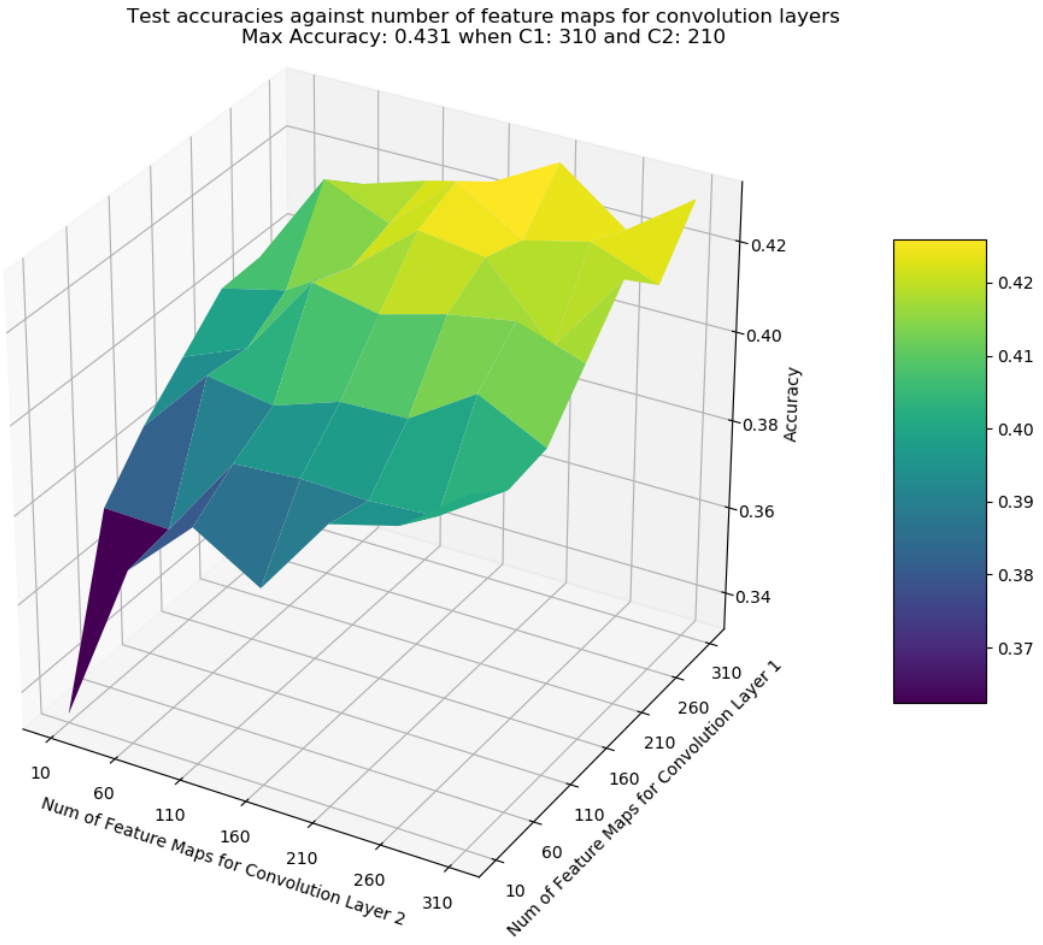


Figure 12: Number of Feature Maps for Conv 1 and Conv 2 against Test Accuracies

A grid search is performed to find the optimal number of feature maps for the two convolution layers, C1 and C2, based on the model obtained in the previous question. The conducted test range is 10, 60, 110, 160, 210, 260 and 310 feature maps. That is, for each test range applied to Convolution Layer 1, the same test range is conducted on Convolution Layer 2.

This results in $7^2 = 49$ test cases which greatly increased the running time for this experiment. As such, the number of epochs was reduced from 1000 to 100 just to demonstrate which configuration of feature maps would yield an increased accuracy.

With reference to the above figure 12 surface plot, the test accuracy obtained increased as the number of feature map increases. The optimal configuration is obtained when the Convolution Layer 1 has 310 feature maps and the Convolution Layer 2 has 210 feature maps. This configuration achieved a maximum accuracy of 0.431.

Question 3

The following models will now adapt to the optimal configuration of 310 feature maps for Convolution Layer 1 and 210 feature maps for Convolution Layer 2.

(a) Adding the momentum term with momentum γ=0.1

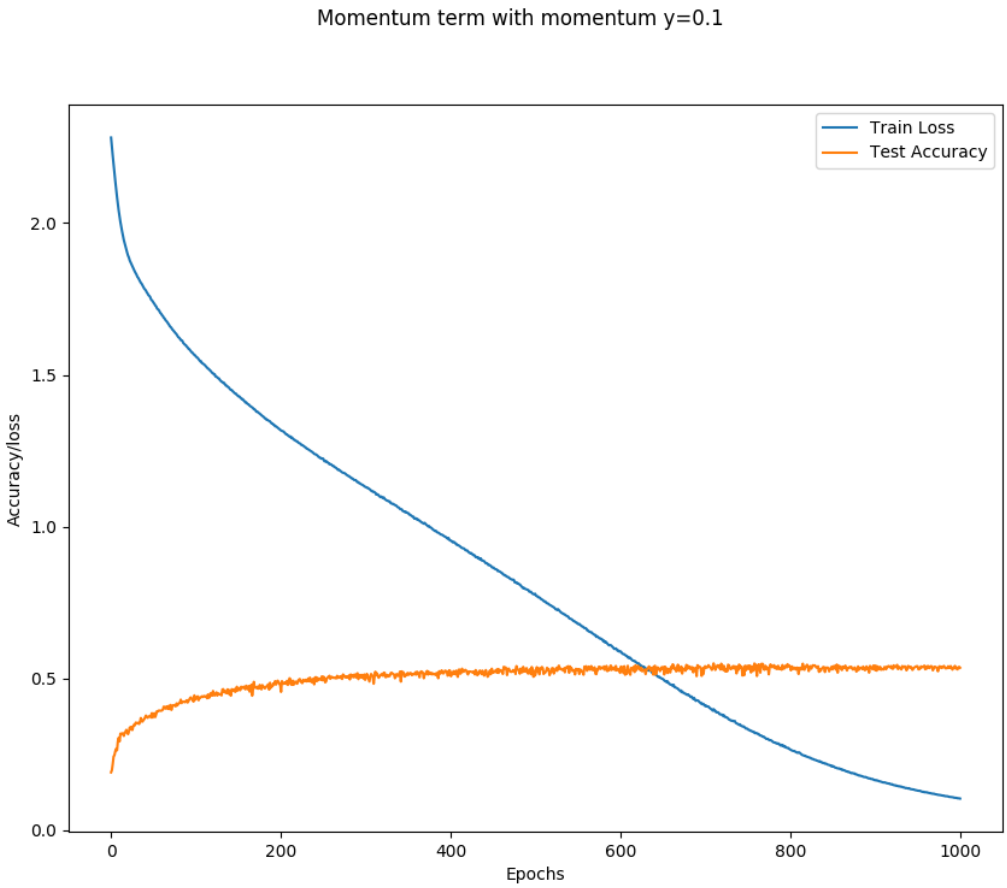


Figure 13: Added momentum term γ=0.1 Train Loss and Test Accuracy against Epochs

A Momentum Optimiser is now being used in the model with the term γ=0.1 as shown in the above figure 13. Instead of only using the gradient of the step to guide the search, momentum helps to accumulate the gradient of the past steps to determine the 'right direction to go'. As such, it helps the model to converge at a faster rate.

It can also be observed that this model yielded a lower training loss when compared to the original Gradient Descent model in figure 1. However, the test accuracy remains fairly stagnant at 0.54. This might be a result of overfitting where the model only performs well on the training dataset but not on the testing dataset.

An interesting find made online was the popular use of Stochastic Gradient Descent (SGD) with Momentum Optimiser. An attempt was made to switch from a Mini-batch Gradient Descent to SGD as seen in the following figure 14. The use of SGD with Momentum helped the model achieve convergence at a much earlier at around 30 epoch as compared to Mini-batch Gradient Descent with Momentum at around 790 epoch.

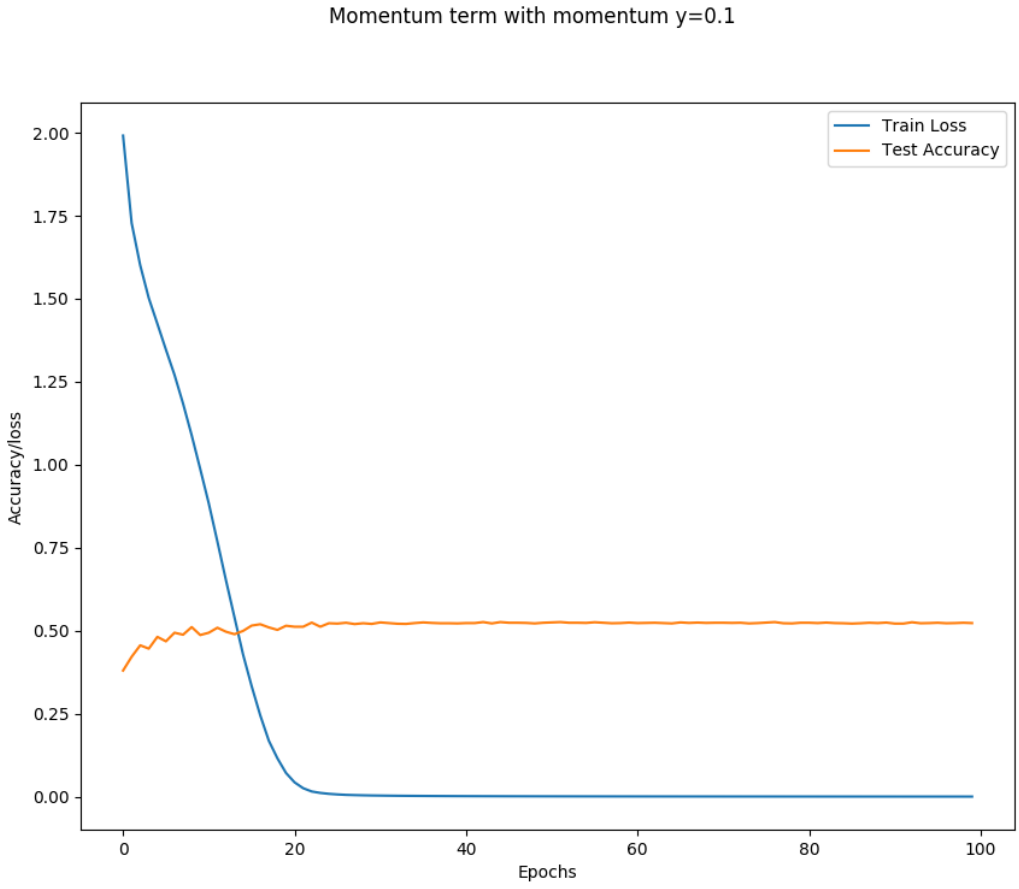


Figure 14: SGD with Added momentum term γ=0.1 Train Loss and Test Accuracy against Epochs

(b) Using RMSProp algorithm for learning

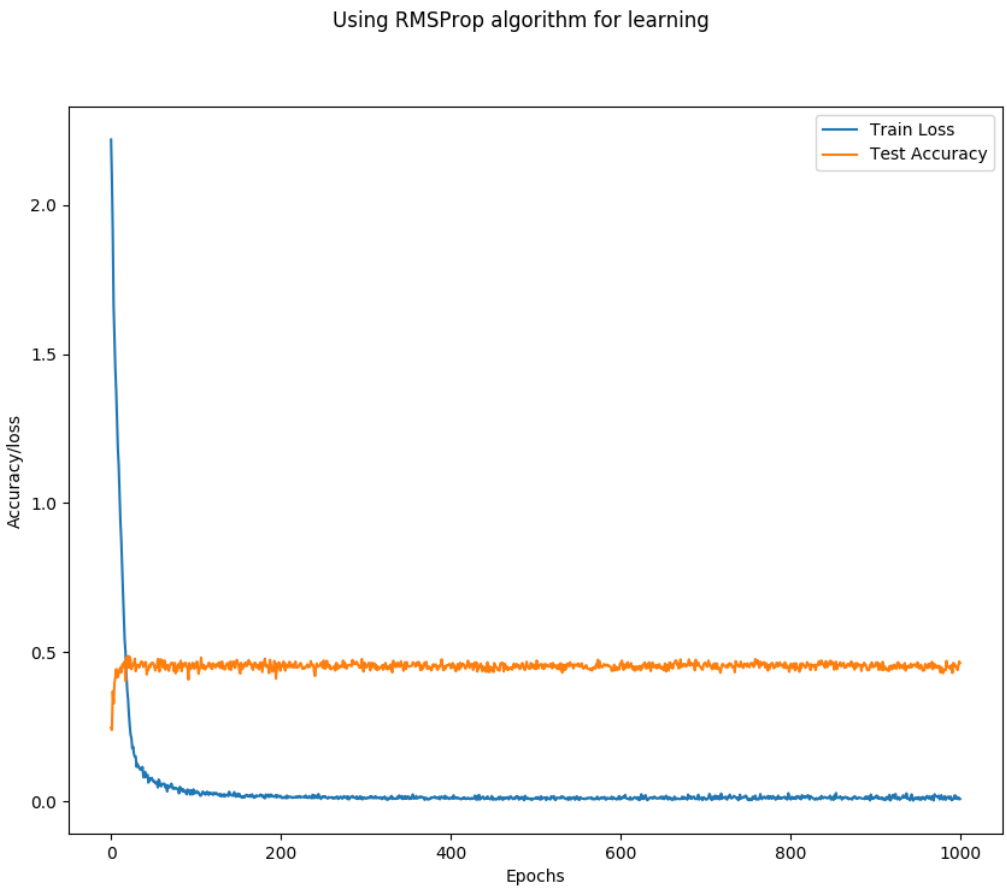


Figure 15: RMSProp algorithm Train Loss and Test Accuracy against Epochs

Root Mean Square Propagation (RMSProp) helps to reduce oscillations in the verticle direction. The above figure 15 illustrates the training loss and test accuracy achieved against epochs with the default parameters used for the RMSProp algorithm.

It is also observed that the training loss reduces and stabilises around 100 epoch which is much faster than the above model with Momentum Optimiser shown in figure 13.

(c) Using Adam optimizer for learning

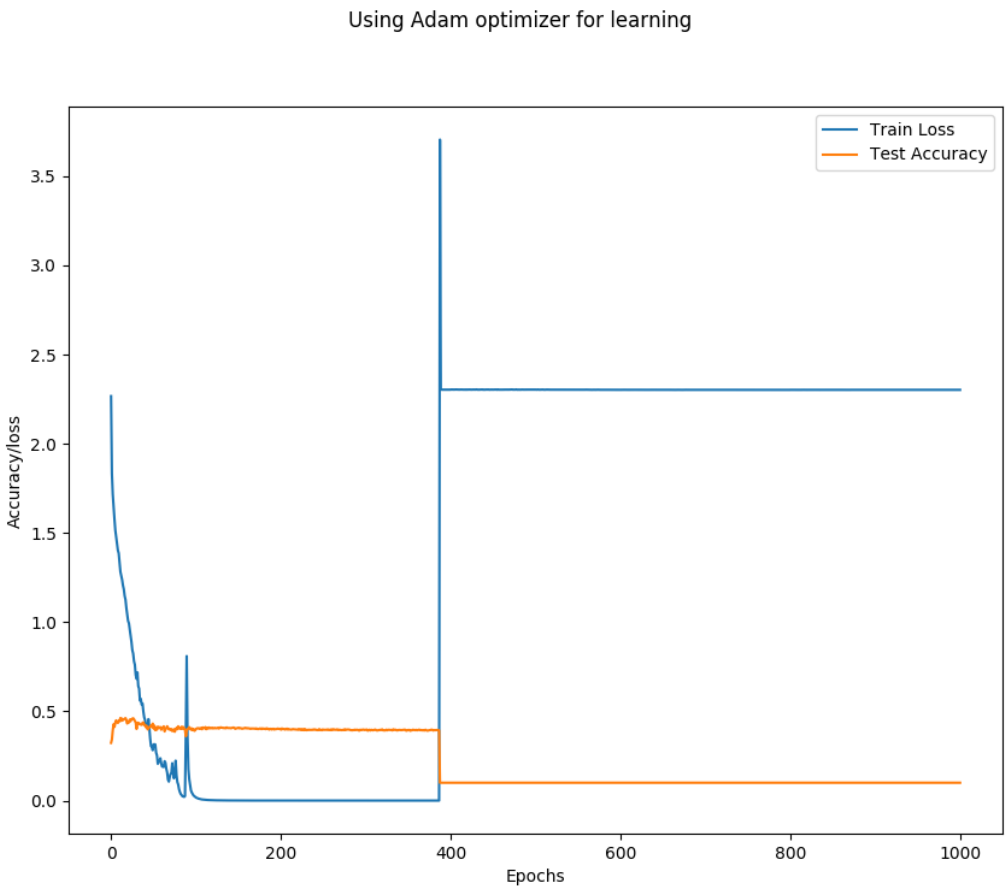


Figure 16: Adam Optimiser Train Loss and Test Accuracy against Epochs

The use of an Adam Optimiser or Adaptive Moment Optimisation takes into account both the heuristics of Momentum and RMSProp. The above figure 16 illustrates the training loss and test accuracy achieved against epochs with the default parameters used for the RMSProp algorithm.

The exploding graph as seen in figure 16 can be attributed to the fact that Adam maintains a rolling geometric mean of the recent gradients and the square of gradients. The square of gradients is then used to divide the current gradient in deciding the learning step. As such, instability can kick in when the squares of the gradient approach zero. This will cause the step size to 'jump around' before settling again.

In order to combat this instability from occurring, the learning rate can be decreased or by introducing gradient clipping.

(d) Adding dropout to the layers

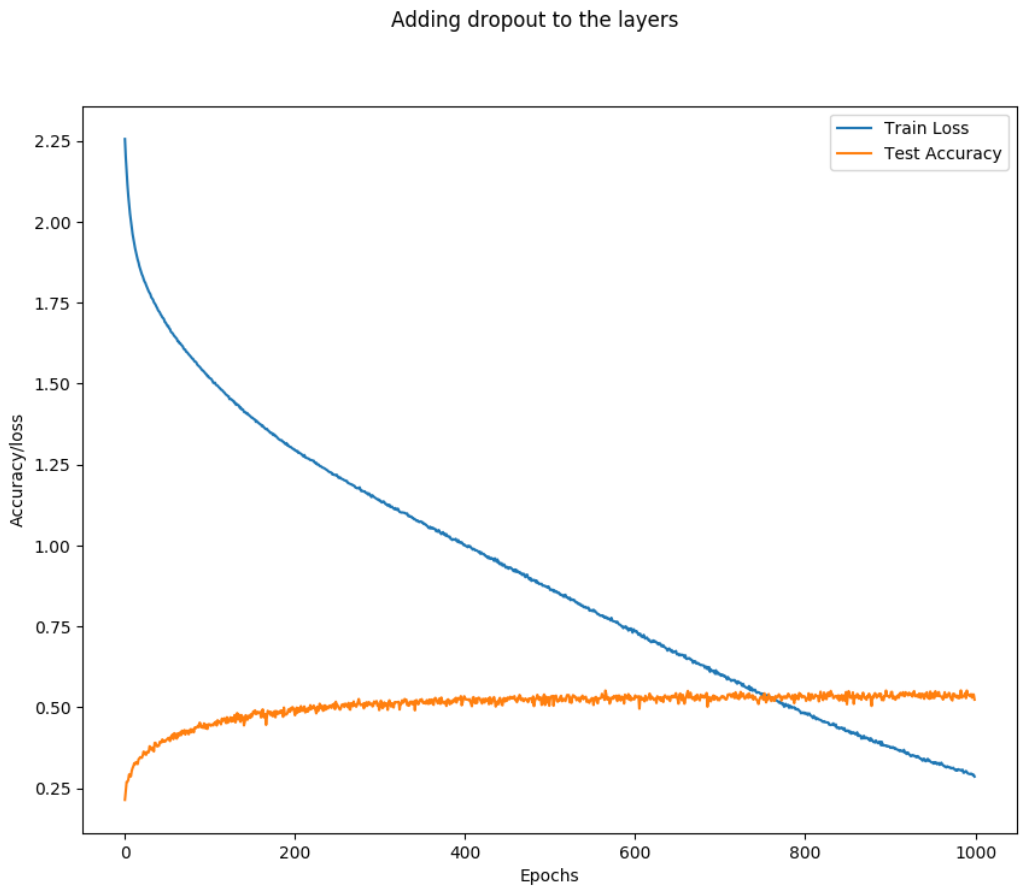


Figure 17: Adding dropout to the layers - Training Loss and Test Accuracy against Epochs

The above figure 17 illustrates the training loss and test accuracy when dropout with (keep rate = 0.8) is added to the layers.

Question 4

In all experiments, it can be observed that the test accuracies achieved for the various models were relatively similar, with the deviation being only around 10%. All models achieved an average of 50% accuracy which is not stellar but further improvements can be made by feeding the model a larger training and testing dataset.

Looking at accuracies alone, the Gradient Descent with Dropout (keep rate = 80%) model managed to yield the highest score of 55.1%. However, this particular model also took the longest to converge - at 990 epochs. The next highest scoring model with Momentum Optimiser achieved 54.4% accuracy with a slightly lesser 790 epochs.

The fastest model to converge would be the Adam Optimiser but it may not be stable. This may be solved by decreasing the learning rate or by introducing gradient clipping.

It is also worthy to note that the use of more feature maps does not necessarily correlate to gaining a higher accuracy. As seen in the Adam and RMSProp Optimiser model, both achieved lower accuracies when compared to the Gradient Descent model with 50 (for C1) and 60 (for C2) feature maps.

Summary of all Models

Model	Feature Maps	Convergence (at epoch)	Accuracy
Gradient Descent	C1: 50, C2: 60	820	0.531
Momentum Optimiser	C1: 310, C2: 210	790	0.544
RMSProp Optimiser	C1: 310, C2: 210	540	0.466
Adam Optimiser	C1: 310, C2: 210	10	0.445
Gradient Descent with Dropout (Keep rate = 80%)	C1: 310, C2: 210	990	0.551
		Average accuracy	**0.5074**

Part B: Text Classification

Question 1

In question 1, it requires to design a Character Convolution Neural Network (CNN) Classifier that receives character ids and classifies the input. The model consists of two convolution and pooling layers with all required parameters for the training. The training runs for 200 epochs and the following is the result:

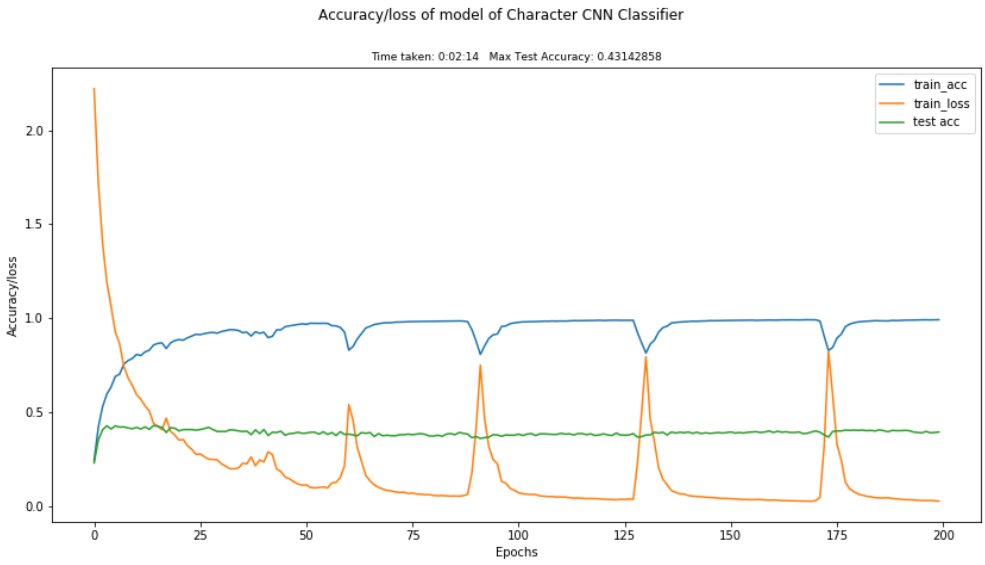


Figure 18: Training loss and Test Accuracy on Char CNN Classifier

As the training approach to 200 epochs, test accuracy is about 43%. However, as seen in figure 18, there were spikes occurred during training. Upon research, these spikes are due to the consequence of Mini-Batch Gradient Descent for Adam Optimizers. Also, spikes can happen if there is a chance in some mini-batches having bad data for the optimization. Some solution may suggest lowering the learning rate for Adam Optimizer.

Question 2

In question 2, it requires to design a Word Convolution Neural Network (CNN) Classifier that receives character ids and classifies the input. The model consists of two convolution and pooling layers with all required parameters for the training. The training runs for 200 epochs and the following is the result:

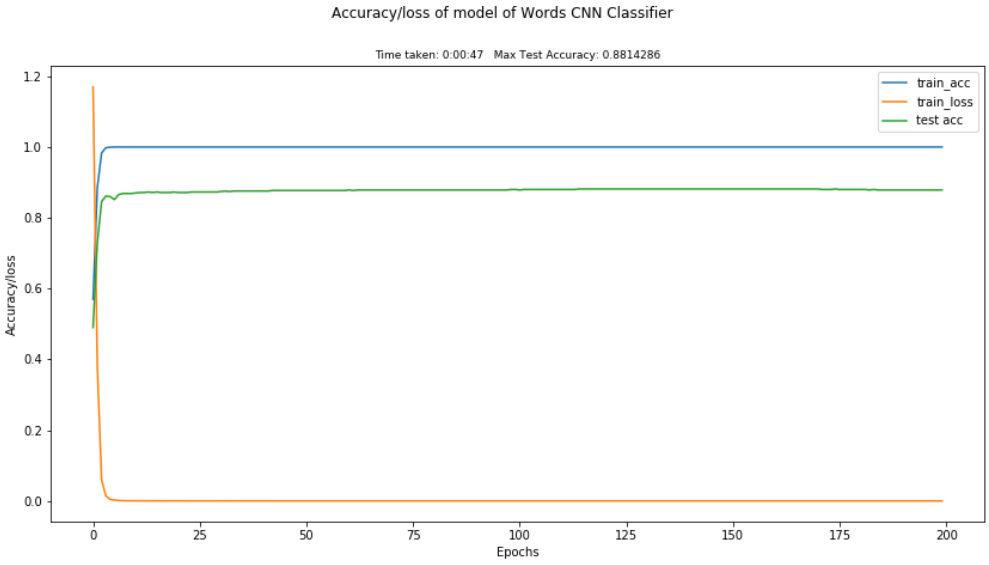


Figure 19: Training loss and Test Accuracy on Word CNN Classifier

From the above figure 19, as the training approach to 200 epochs, test accuracy is about 88%. This shows that the model using words ids produces higher accuracy than character ids.

Question 3

In question 3, it requires to design a Character Recurrent Neural Network (RNN) Classifier that receives character ids and classifies the input. The RNN model is GRU layer and has a hidden-layer size of 20. GRU stands for Gated Recurrent Unit which is an improved version of a standard RNN model. Standard RNN Model suffers from vanishing gradient problem and GRU is to aim to solve this. Also, GRU can be considered as a variation on the Long Short Term Memory (LSTM) model which both are similarly designed. (should add abit more? Seems incomplete)

The training runs for 200 epochs and the following is the graph result:

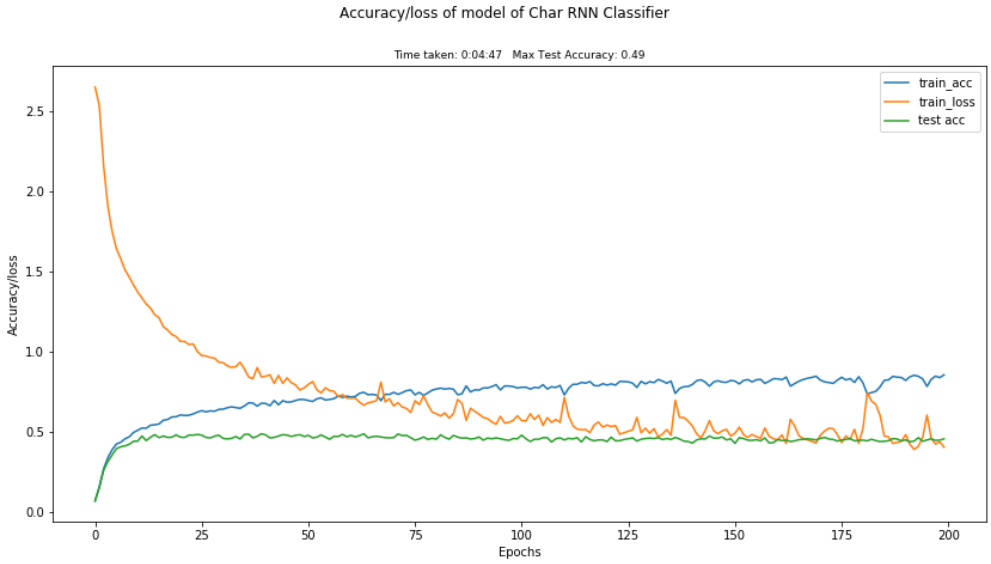


Figure 20: Training loss and Test Accuracy on Char RNN (GRU) Classifier

From the above figure 20, as the training approach to 200 epochs test accuracy is about 49%. There were noticeable spikes in the training loss, this is the same explanation given in **Question 1** due to the Mini-Batch Gradient Descent for Adam Optimizers. By observation, it shows that Character RNN classifier performed slightly better accuracy result than the Character CNN classifier.

Question 4

In question 3, it requires to design a Word Recurrent Neural Network (RNN) Classifier that receives word ids and classifies the input. The RNN model is GRU layer and has a hidden-layer size of 20. The inputs are passed through an embedding layer of size 20 before feeding to the RNN.

The training runs for 200 epochs and the following is the graph result:

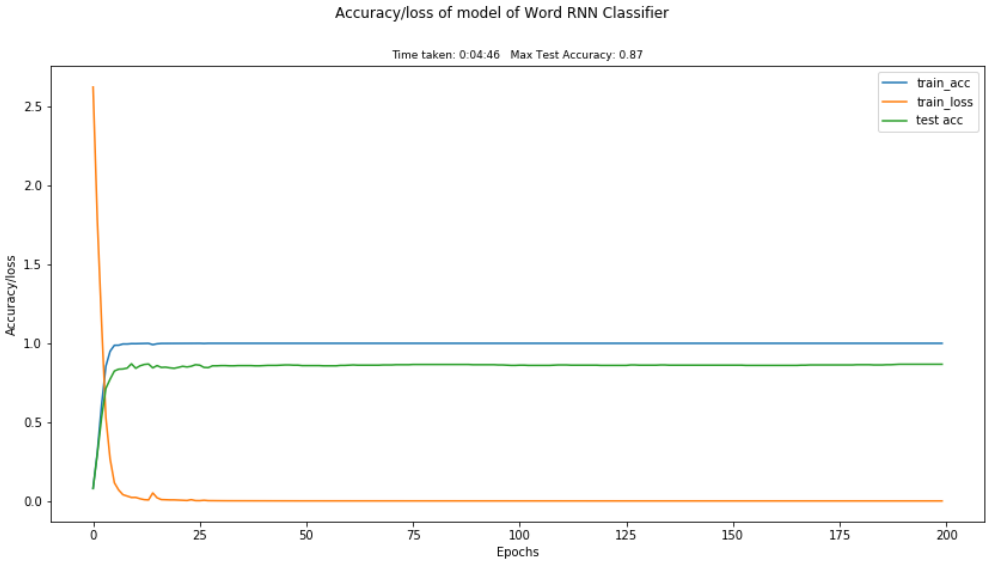


Figure 21: Training loss and Test Accuracy on Word RNN (GRU) Classifier

From the above figure 21, as the training to 200 epochs the test accuracy is about 87% having a relatively high accuracy compared to the Character RNN.

Question 5

Comparison of CNN and RNN

Upon comparison of the 2 models, there are noticeable difference on the test accuracy and run time. Comparison of Character CNN and RNN graph and Comparing of Word CNN and RNN graph are shown below.

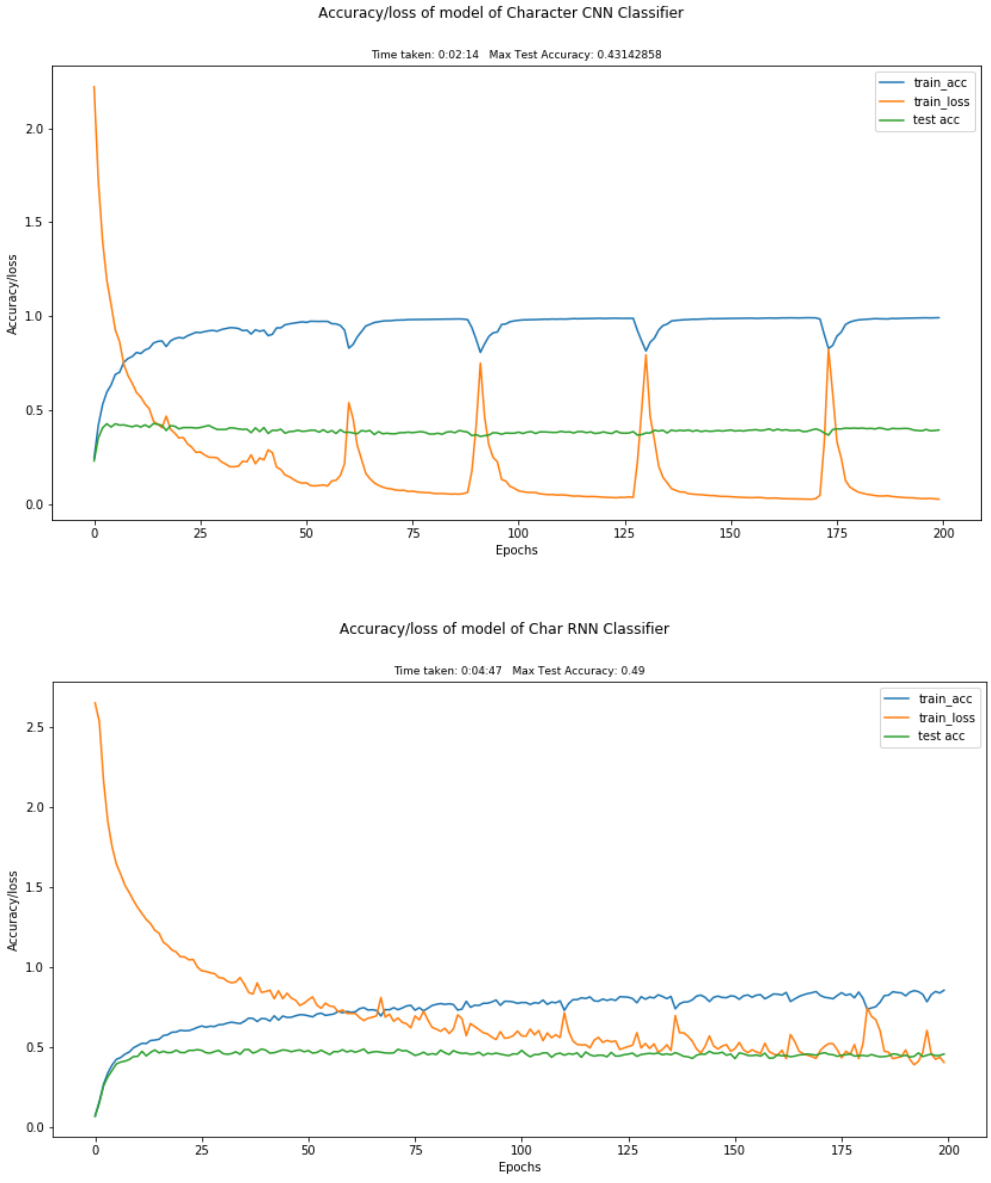


Figure 22: Comparison of CNN and RNN for Character Classifiers

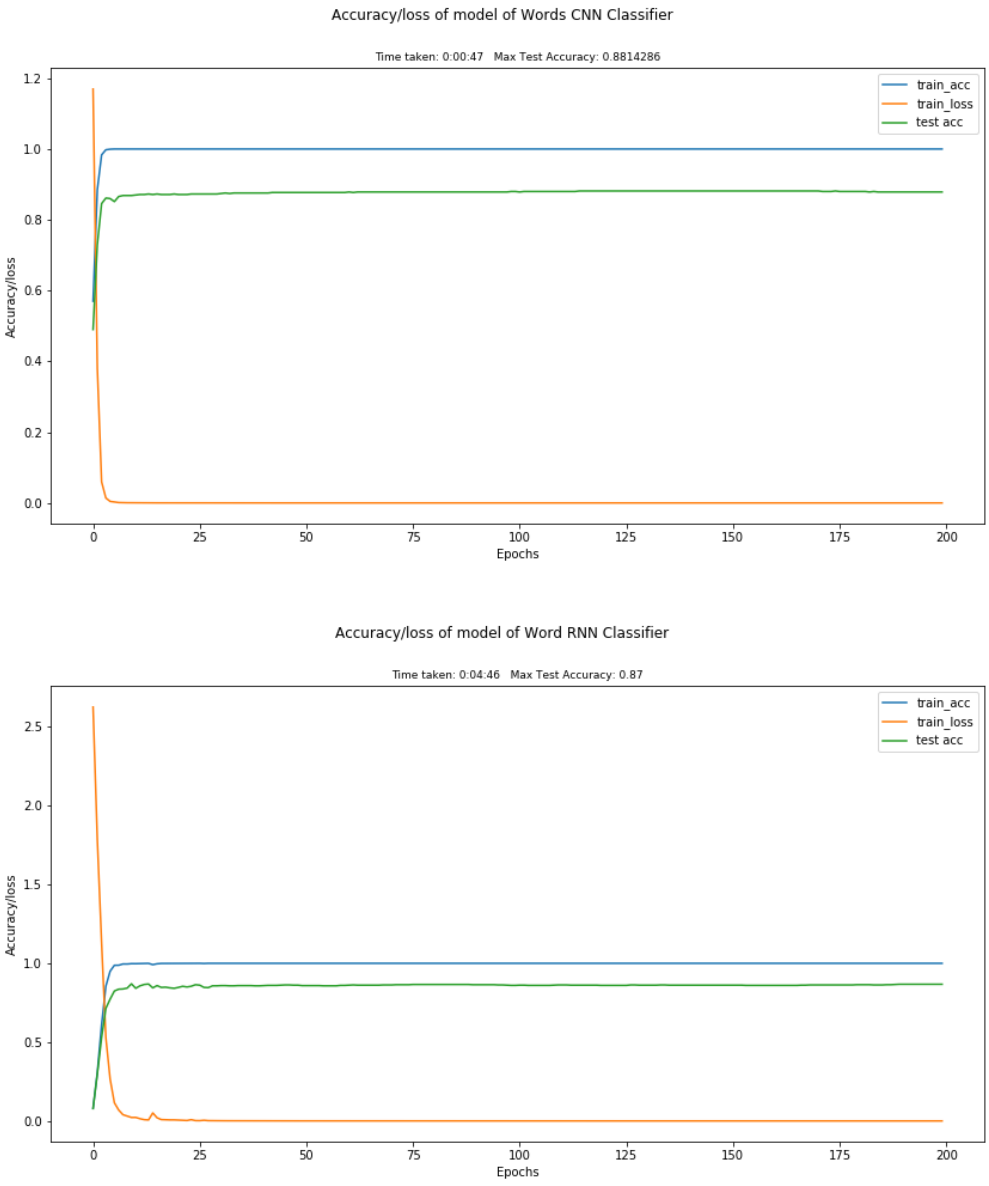


Figure 23: Comparison of CNN and RNN for Word Classifiers

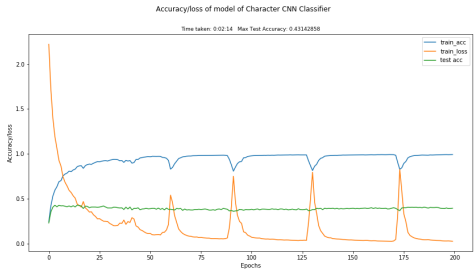
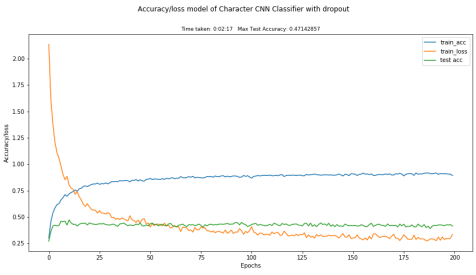
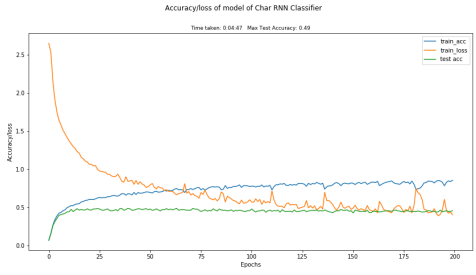
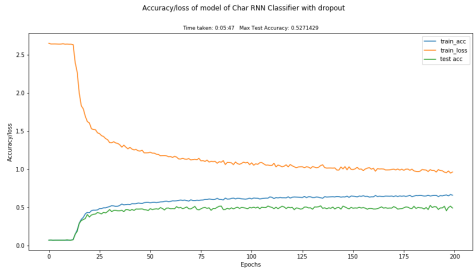
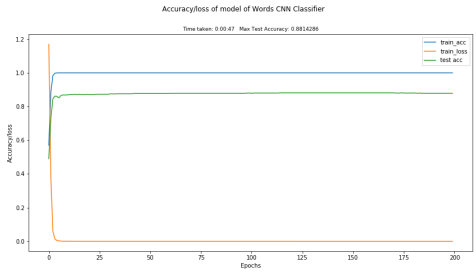
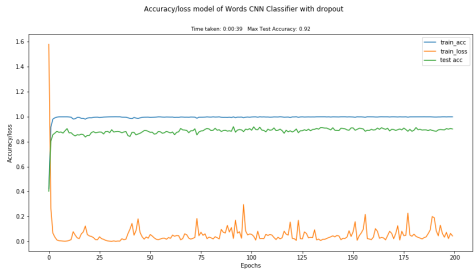
From the above graphs, higher test accuracy are achieved using words as inputs than using characters. RNN took a longer time to train as compared to CNN. RNN are more suitable for classifying text due to the fact that RNN's architecture allows to exhibit temporal behavior and capture sequential data which makes it a more 'natural' approach when dealing with textual data since text is naturally sequential. However, CNN also shown to produce good results when comes to performing the same task. Besides the difference in architecture of RNN and CNN, RNN is trained to recognize patterns across time, while CNN learns to recognize patterns across space.

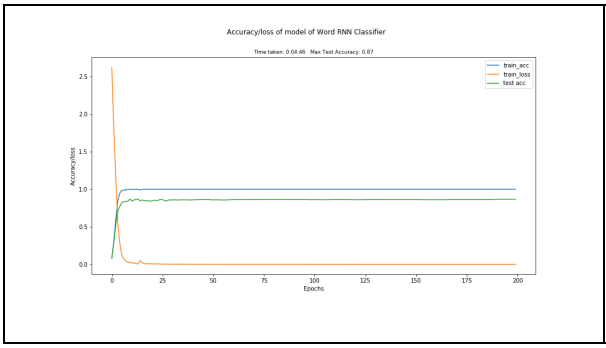
Summary Table of CNN and RNN:

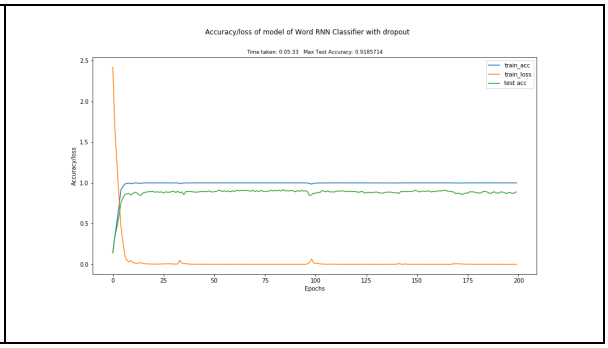
Model	Run Time	Test Accuracy
Character CNN Classifier	2:14 min	0.431
Character RNN Classifier	4:47 min	0.49
Word CNN Classifier	0:47 min	0.881
Word RNN Classifier	4:46 min	0.87

CNN did perform much faster time to train than RNN and having accuracy not too far off from that in RNN.

Comparison of CNN and RNN with and without dropouts:

Without dropout	With dropout
	
	
	





Test accuracy with and without dropouts:

Model	Without dropout (Accuracy)	With dropout (Accuracy)
Character CNN Classifier	0.431	0.471
Character RNN Classifier	0.49	0.527
Word CNN Classifier	0.881	0.92
Word RNN Classifier	0.87	0.919

Dropout is a regularization technique to help reduce overfitting network. From the comparison table above, it shows the result of having dropout and without. Since initially, the result produced from both models showed no sign of overfitting. If there are overfitting in the models, the graph will be able to show an anomaly where the training loss will increase at one point as training epoch increases. From Observation, models with dropout has made the training process noisy. This is because the nodes within a layer have been forced probabilistically take on more or less responsibility for the inputs. Dropouts also make the training loss decrease at a slower rate making a little worse. The reason is that it trade training performance for more generalization. It seems that the model that train with dropout show a slightly improved accuracy.

Question 6

Implementing Vanilla RNN Layer:

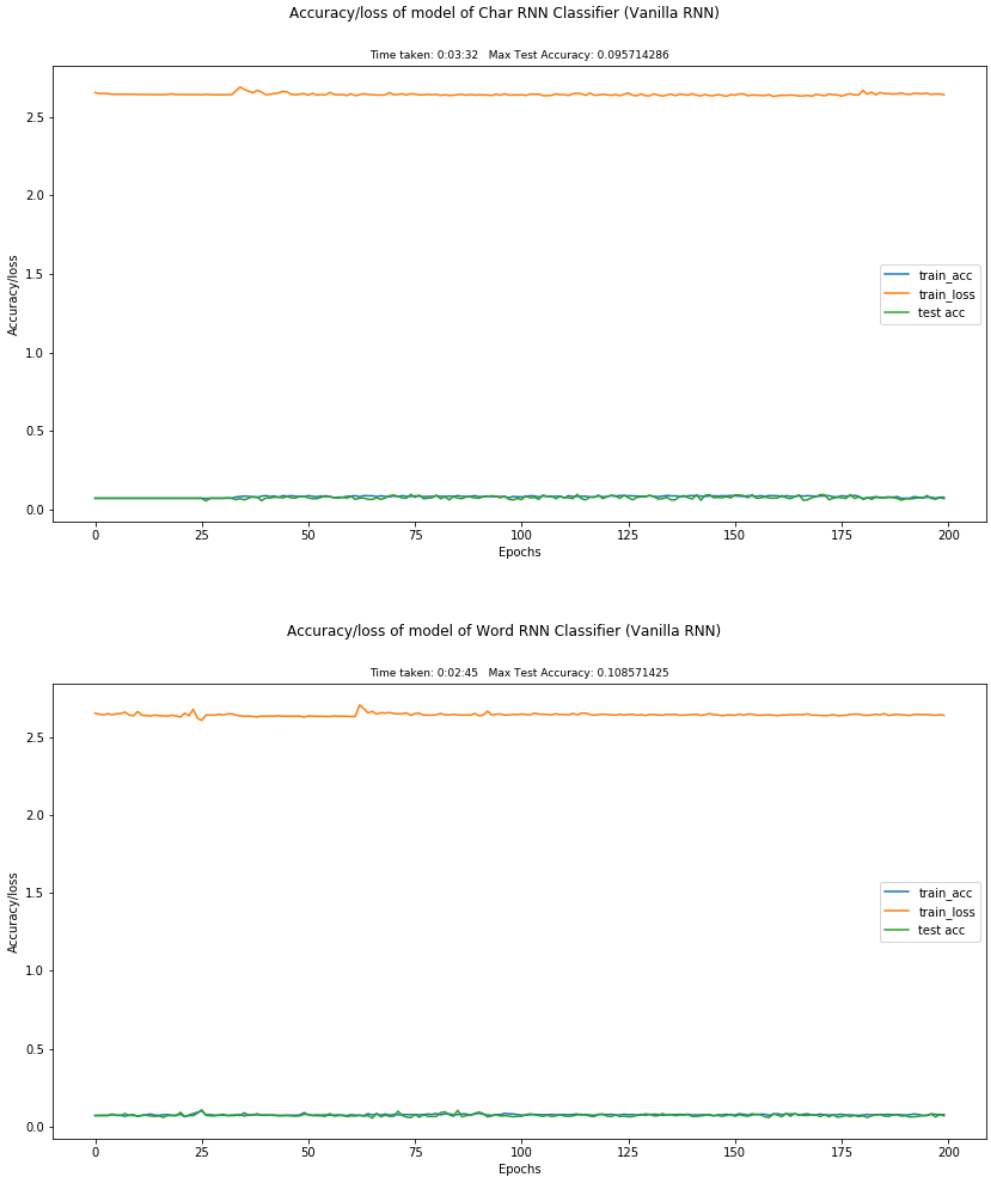


Figure 24: Character and Word RNN with Vanilla RNN

From the above graph, it seems that the Vanilla RNN barely learns anything through training. This in fact that Vanilla RNN is bad at handling backpropagation and it suffers from vanishing gradient problem. During back propagation, gradients value are used to update the weights. The vanishing gradient problem occurs when the gradient shrinks as it back propagates through time. It will not contribute much learning if a gradient value becomes extremely small. Therefore, in Vanilla RNN layers that get a small gradient update stops learning.

Implementing LSTM Layer:

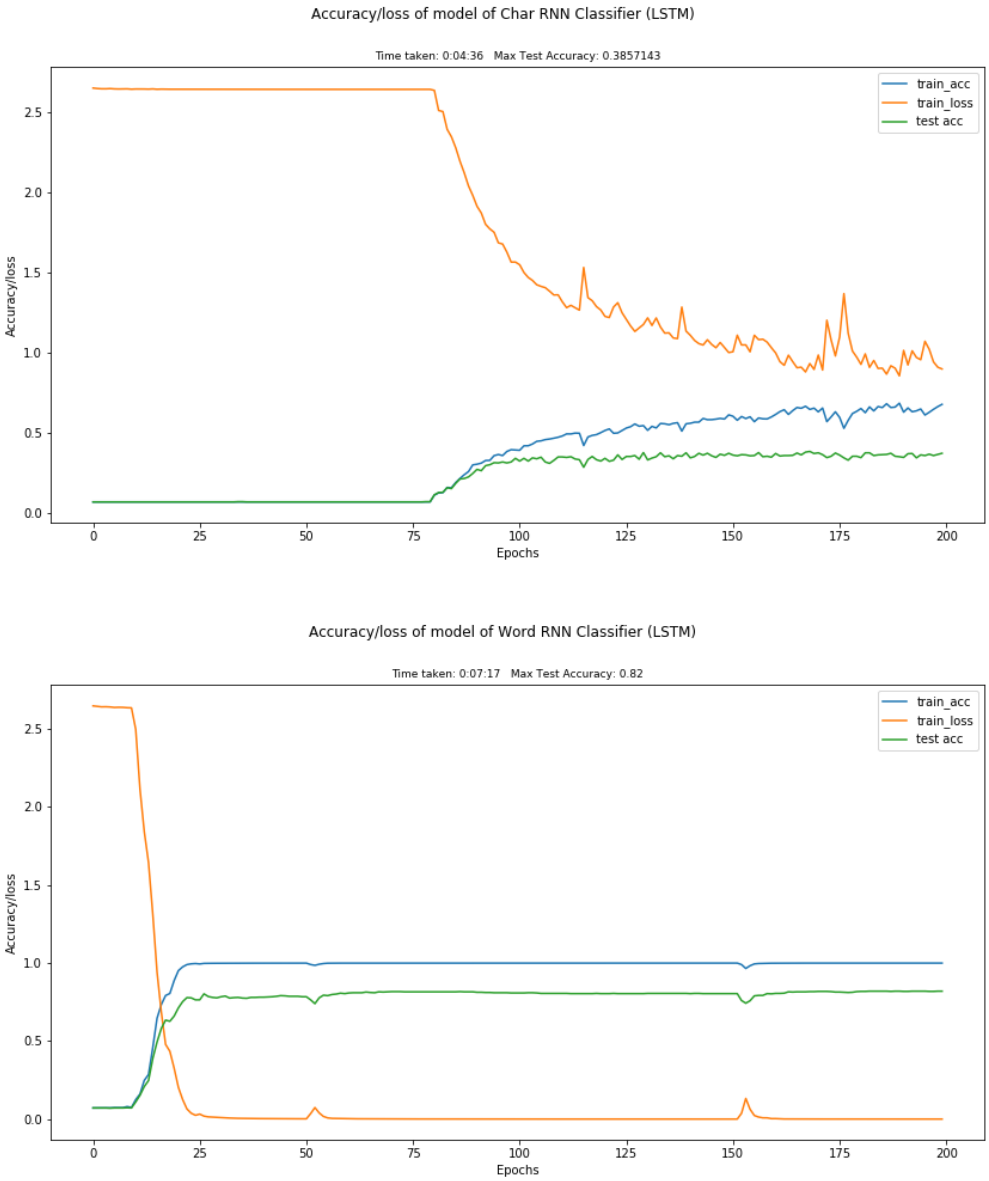


Figure 25: Character and Word RNN with LSTM RNN

From the graph above, LSTM RNN shows a better result as compared to a Vanilla RNN since LSTM was created to solve the problem have in Vanilla RNN of vanishing gradient. However, LSTM RNN took a longer time to training but it produce better test accuracy and training loss.

Implementing 2 RNN Layer:

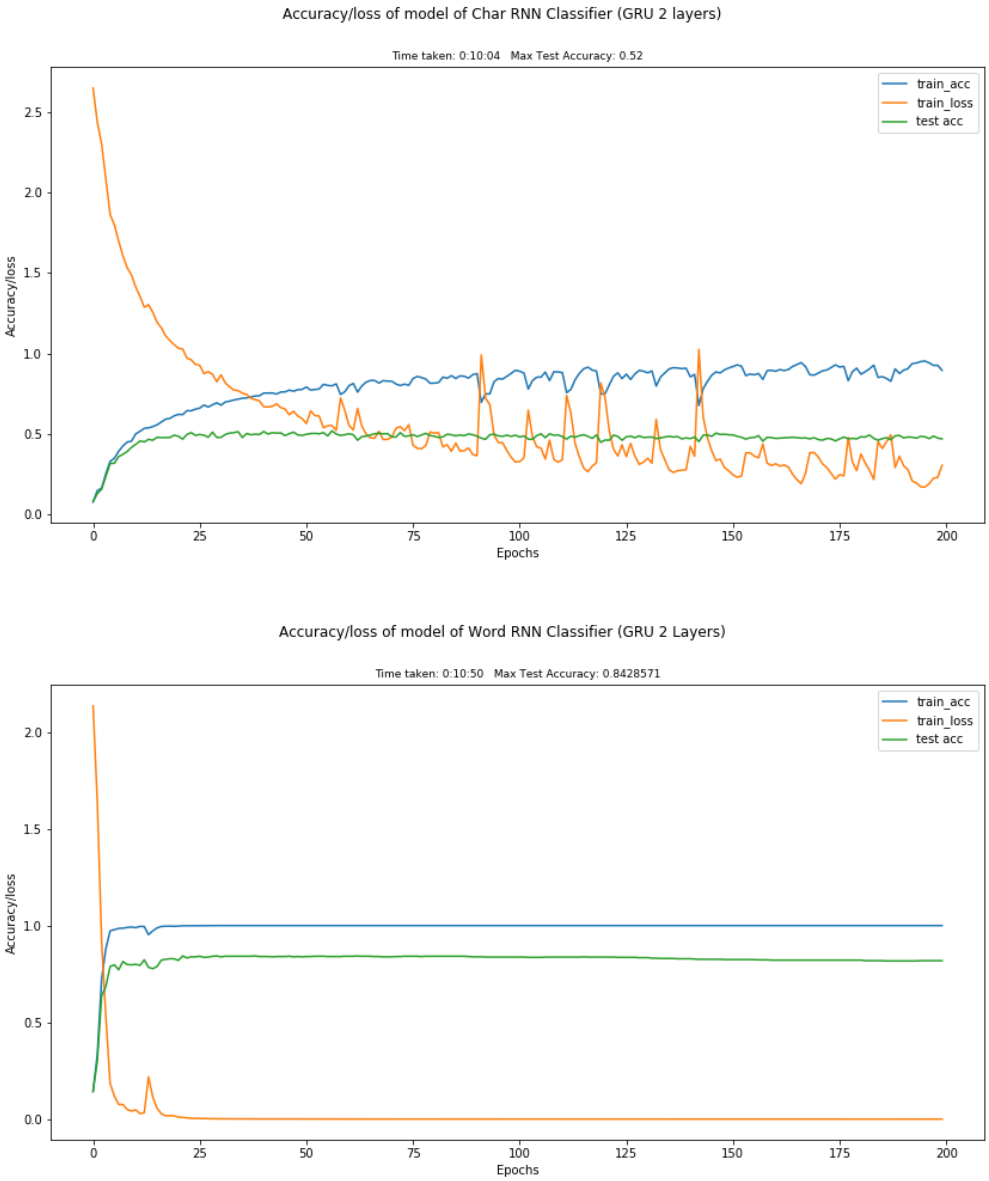


Figure 26: Character and Word RNN with 2 Layers

From the graph above, increase the number of layers to 2 shows an enhancement to the performance of the result. Increase the depth of the neural network enable problems to be better represented. Which each layer completing a task of the problem is passed to the next layer.

Implementing RNN with Gradient Clipping:

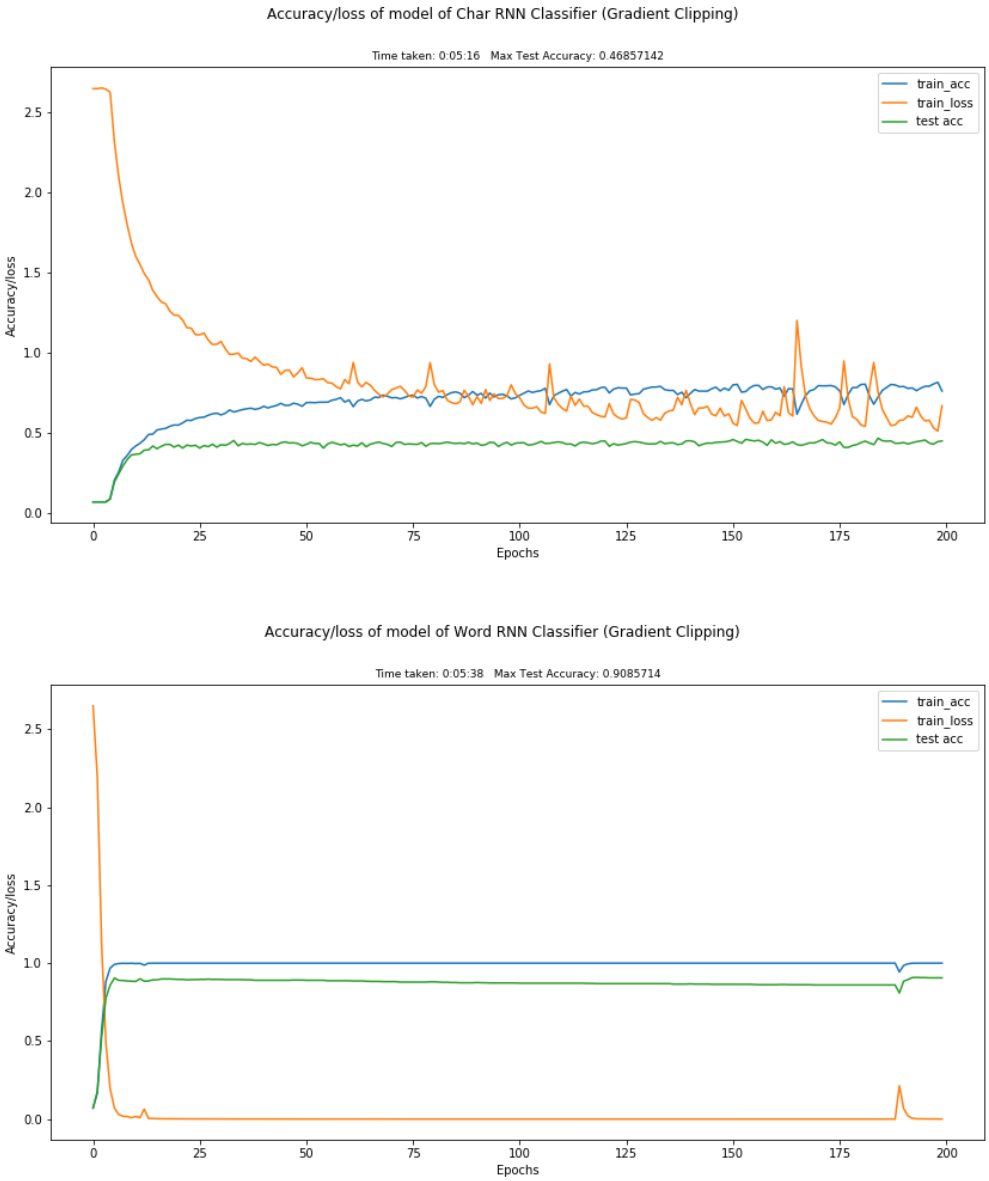


Figure 27: Character and Word RNN with Gradient Clipping

Gradient clipping is used to prevent exploding gradients which is a problem where large error gradients accumulated and result in very large update to the weights in the model during training. As this will effect the model being unstable and unable to learn from training data. Applying gradient clipping it helps to prevent these issues in the gradients that will mess up the parameters. Gradient clipping have a slight improvement for Word RNN Classifier than 2 layers RNN Word Classifier.

Summary of implementation model on RNN:

Model	Implementation	Run time	Test accuracy
Character RNN	Vanilla RNN	3:32 min	0.096
Word RNN	Vanilla RNN	2:45 min	0.109
Character RNN	LSTM RNN	4:36 min	0.386
Word RNN	LSTM RNN	7:17 min	0.82
Character RNN	2 Layer RNN	10:04 min	0.52
Word RNN	2 Layer RNN	10:50 min	0.842
Character RNN	Gradient Clipping	5:16 min	0.469
Word RNN	Gradient Clipping	5:38 min	0.909

Conclusions

Part A: Object Recognition

In conclusion, the objective of Part A was to perform object recognition on the CIFAR-10 dataset which contains 10,000 RGB colour images of size 32x32 and their labels from 0 to 9.

To achieve this, a multi-layer Convolutional Neural Network (CNN) was designed. The multi-layer consists of 2 Convolution Layers, 2 Pooling Layers, a fully connected layer and a softmax layer. Subsequently, the optimal number of feature maps needed to be found to obtain the highest accuracies for the 2 Convolution Layers. Subsequently, testing is performed on 20,000 test samples to evaluate model accuracy.

After obtaining the base performance of a Gradient Descent model with 50 filters in the C1 layer and 60 filters in the C2 layer, a grid search was performed to find the optimal configuration of filters. It was then concluded that Convolution Layer 1 should have has 310 feature maps and the Convolution Layer 2 should have 210 feature maps as seen in figure 12.

In the later part of this section, multiple optimisers such as Momentum, RMSProp and Adam were investigated to evaluate the best optimisers with the said C1 and C2 configuration filter.

After all the experiments, it can be concluded that the Gradient Optimiser with Dropout (keep rate = 0.8) model achieved the best accuracy at 55.1% while the next best performing is the Momentum Optimiser at 54.4% accuracy.

Part B: Text Classification

Part B experimented with a different kind of dataset for text classification, either character or words for classification of text in paragraphs. Experimenting with both CNN and RNN for the classification problem with different implementation.

Comparison between CNN and RNN on character and word classification shows some interesting results. CNN able to train faster and still produce and fair amount of accuracy compared to RNN. Furthermore, word classification on CNN gave a slightly higher accuracy than RNN. This could be mean that actually CNN can apply in certain types of Natural Language Processing application.

Further experiment on the models with different implementation such as dropouts and changing the RNN layer to different variant like Vanilla RNN, GRU and LSTM. If a model tends to overfit, dropout is introduced into the network to prevent overfitting. However, dropout function will have a trade off in performance for generalization. Having the most basic RNN, the Vanilla RNN could not learn during training because the model suffer from ‘short-term memory’ which caused by gradient vanishing problem during back propagation while updating the weights. Improved version of the Vanilla RNN to solve the gradient vanishing issue are the GRU and LSTM. Definitely GRU and LSTM performed much better and having good accuracy.

Finally, implementing gradient clipping helps to prevent gradient explosion and also produce a slightly better result in the text classification problem.