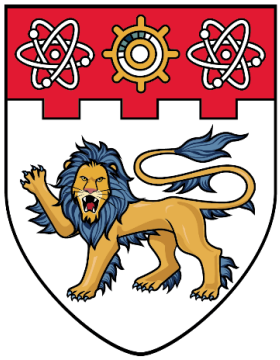

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE4042 Neural Networks and Deep Learning

Project Report 2

by

Lai Qing Hui, U1622339G
Ron Ng Jian Ying, U1622393B

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

2019

Contents

Part A: Object Recognition

Introduction

In this section, we will be building a convolutional neural network (CNN) to predict the label of a testing instance from the CIFAR-10 dataset. The dataset contains 10,000 training samples and 2,000 test samples. Each data sample is an RGB colour images of size 32 x 32 and each sample have a corresponding integer label from ranging from 0 to 9.

Experiments & Results

A base CNN was constructed with an architecture as seen in Figure 1 below.

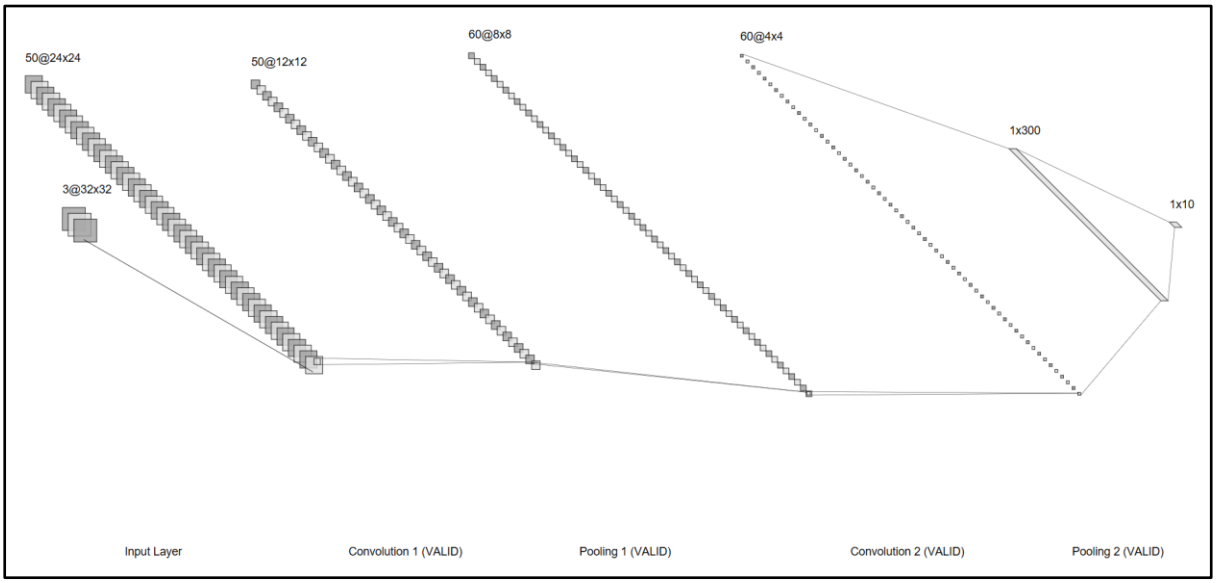

Figure 1: Base CNN architecture

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

Before we start the experiments, we first scaled the input images by using the min-max normalization:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Note that the x_{min} for the given dataset is equal to 0.

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

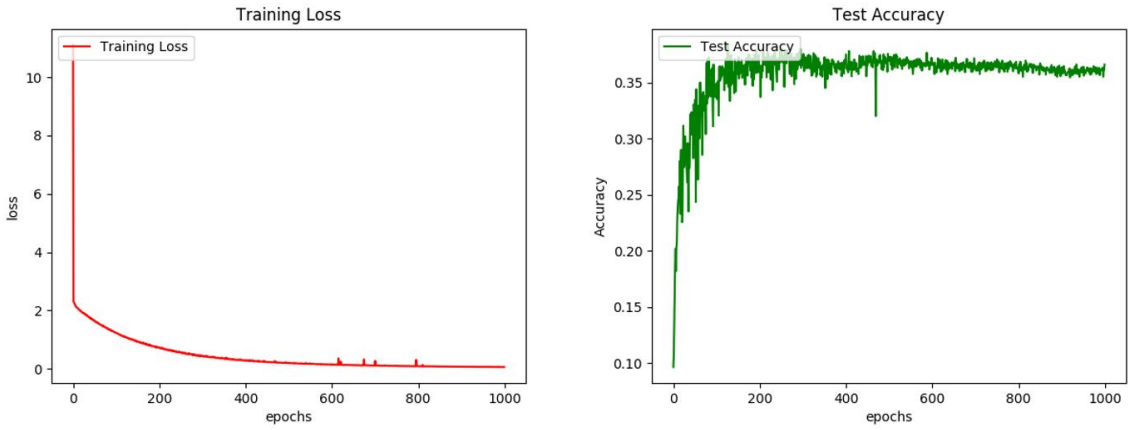

Figure 2: Training Loss vs Test Accuracy

From the results shown in Figure 2 it can be observed that the model converges at around 700 Epoch. However, it does not produce good result with accuracy peaking at only 38%. This can be due to the relative low complexity of the model which is not enough to properly memorize the CIFAR-10 dataset.

b. For any two test patterns, plot the feature maps at both convolution layers (C_1 and C_2) and pooling layers (S_1 and S_2) along with the test patterns.

10000th Test Pattern

Figure 3: 10000th test pattern

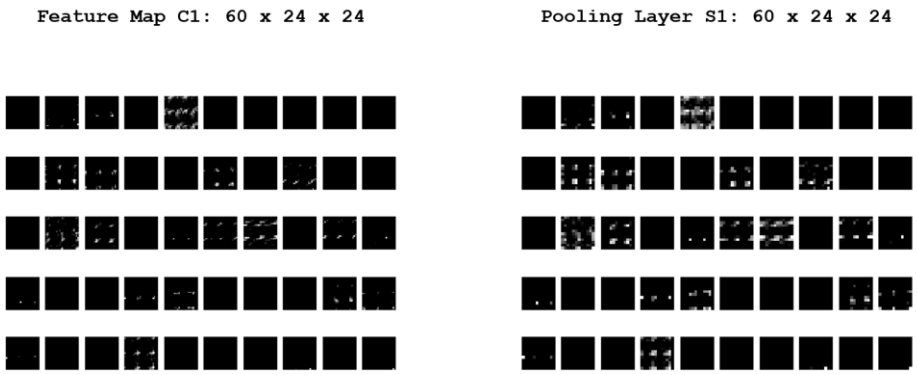

Figure 4: Feature map C1 and Pooling Layer S1 for the 10000th test pattern

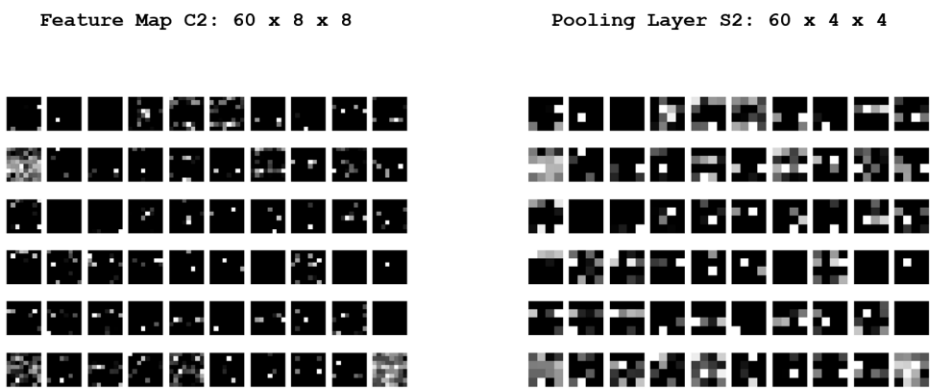

Figure 5: Feature map C2 and Pooling Layer S2 for the 10000th test pattern

9999th Test Pattern

Figure 6: 9999th test pattern

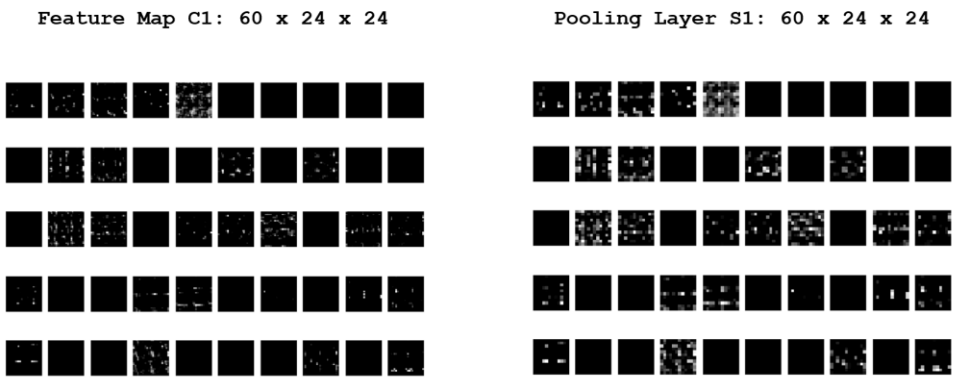

Figure 7: Feature map C1 and Pooling Layer S1 for the 9999th test pattern

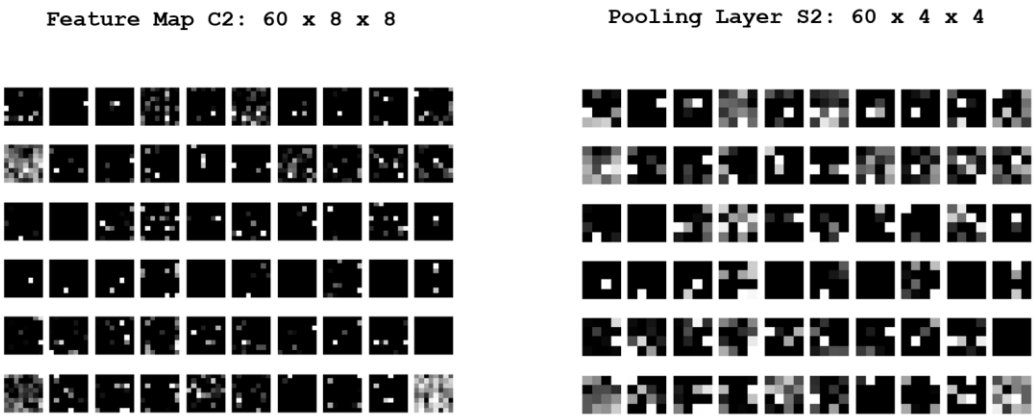

Figure 8: Feature map C2 and Pooling Layer S2 for the 9999th test pattern

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.

As there are two convolutional layers in the CNN architecture, there are many possible combinations for the filter size use in the layers. As a result, it will require a large amount of computational time and power in order to do an exhaustive search. Therefore, we chose to employ a strategy where we first experimented on a few sets of combinations, and from the best performed combination we will then perform a neighbourhood search to find the local optimal solution.

We started with the combinations of (20,20), (40, 40), (60,60), (80, 80), (100,100), (120, 120), and (140, 140). Each combination is trained for 1000 Epochs and the test accuracy are used to benchmark the combination performance.

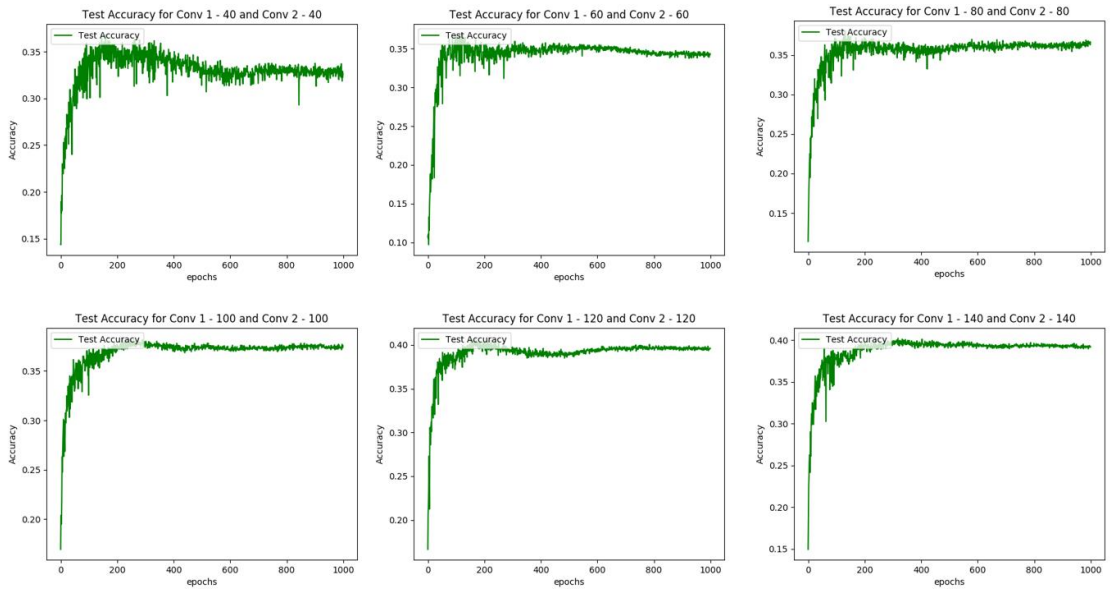

Figure 9: Experimental search results

From the results it can be observed that the results stay stagnant when after the size of 100. Therefore, we proceed to do a neighbourhood search around the size of 100. Some of the notable results from the neighbourhood search are shown in Figure 10 below.

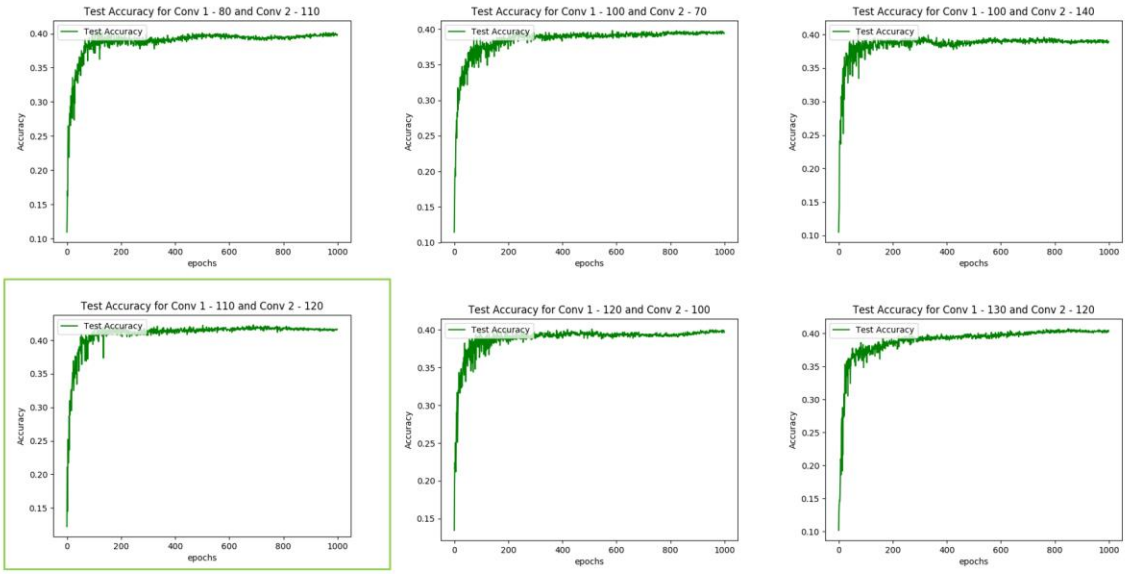

Figure 10: Selected results from neighbourhood search

From the neighbourhood search, the combination of (110, 120) produce the best results and is the most stable for the combinations that attained the maximum accuracy (42%). Therefore, it was chosen to be the filter values that we will be using for the rest of the experiments.

3. Using the optimal number of filters found in part (2), train the network in various parameters.

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

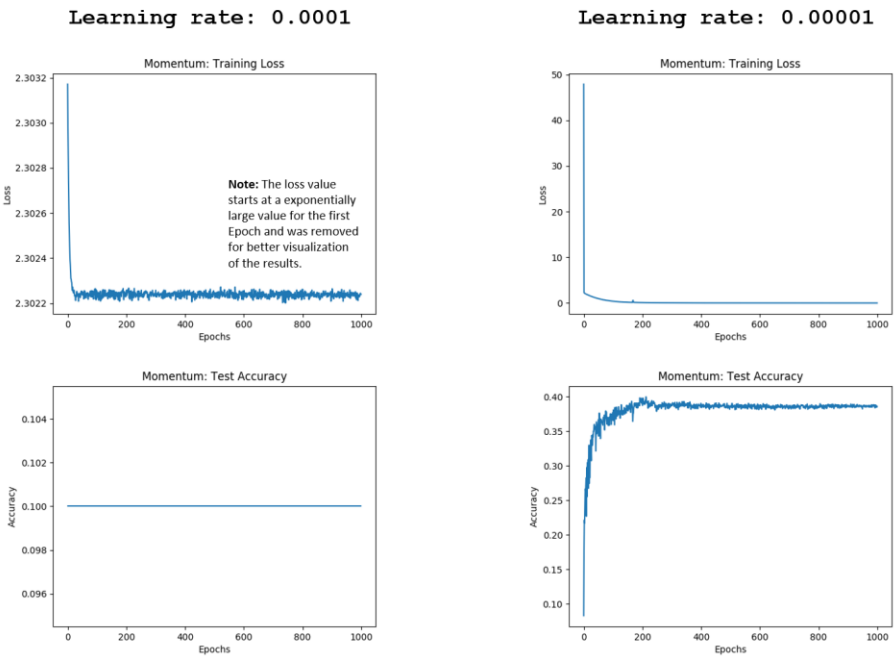

Figure 11: Momentum results

The model fails to converge for a learning rate of $1e^{-4}$ when momentum (γ = 0.1) is added to the gradient calculation. The momentum algorithm works by accumulating an exponentially decaying moving average of past gradients and continue to move in their direction. With this added push, the learning proves to be excessive when the learning rate is set to $1e^{-4}$. However, the model is able to converge when we reduced the learning rate by a factor of 10 to $1e^{-5}$.

b. Using RMSProp algorithm for learning

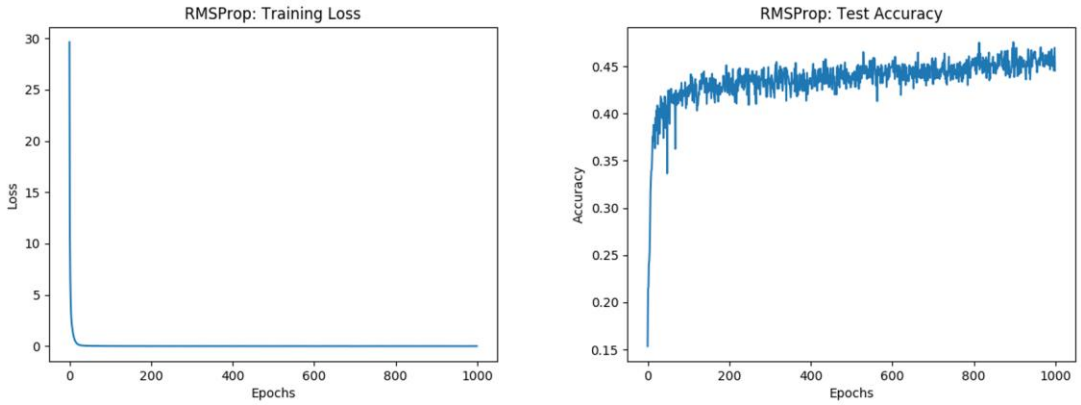

Figure 12: RMSProp results

It can be seen that the RMSProp is able to attain the previous maximum accuracy (42%) produced by the plain Gradient Descent method in far lesser Epoch. It was also able to further improve the accuracy to a new maximum of 45% after 1000 Epoch.

c. Using Adam optimizer for learning

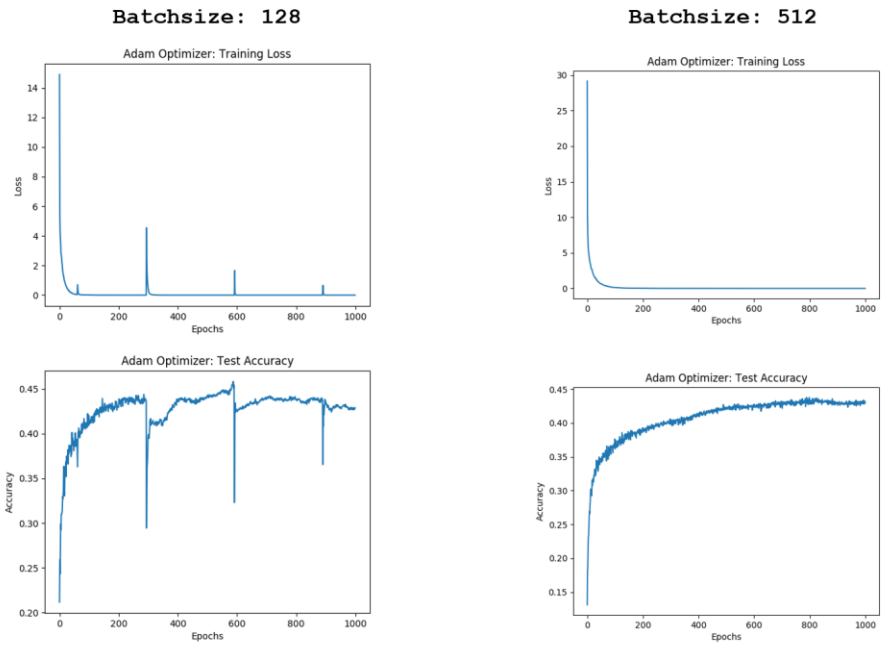

Figure 13: Adam Optimiser results

Similar to RMSProp, the Adam optimiser is able to achieve the previous maximum accuracy (40%) produced by the plain Gradient Descent method in far lesser Epoch. However, there exist spikes in the training loss at regular interval, which is caused by the small *Batchsize* parameter we used for the data sampling for mini batch Gradient Descent operation. This allows outlier data to exert high amount of influence on the calculation of the loss function. It can be observed that the spikes noises can be eliminated by increasing the training batch size to 512.

d. Adding dropout to the layers

Dropout were added to the Fully Connected Layer with rates decreasing in steps of 0.2, starting from 0.9 and ending at 0.1. Note that *keep prob* shown in Figure 14 below is equivalent to 1 - Dropout rate.

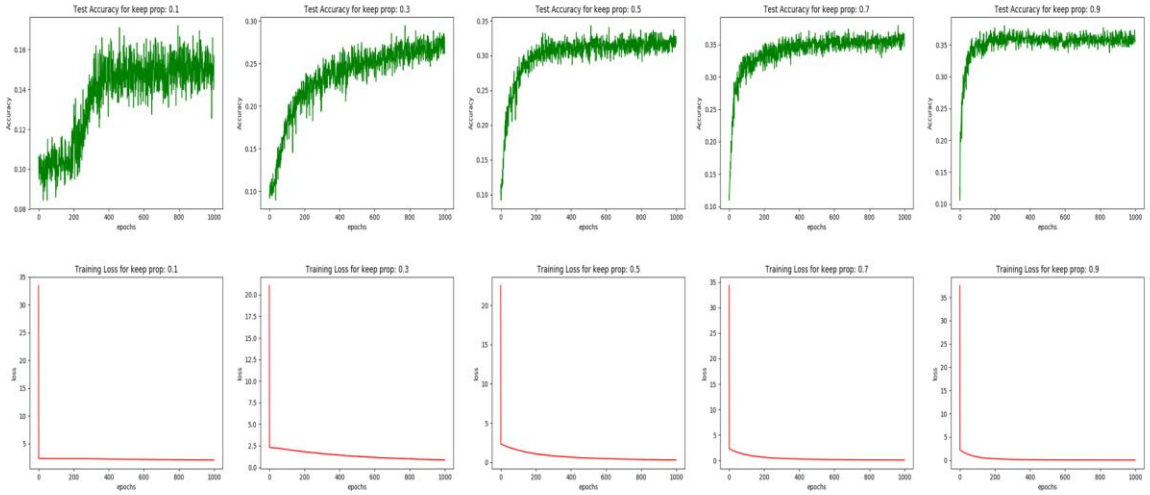

Figure 14: Comparisons result for various dropout values

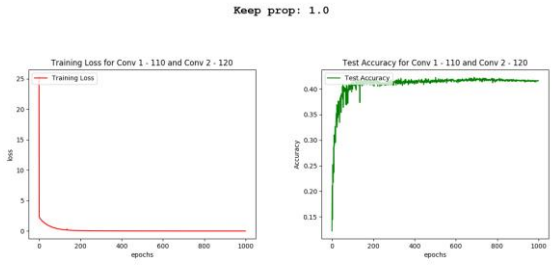

Figure 15: Reference result for zero dropout

The result shows that adding dropout to the network does not increase the performance of the model. One reason for this is because the size of the network is relatively small and there are no clear signs of overfitting as seen in the plots when no drop out was applied to the network. The test accuracy is seen to be consistently increasing inversely proportional to how the

training loss is decreasing. From the onset, this simple CNN model is already not able to learn the complex dataset accurately. By introducing dropout, we are regularizing the model which is essentially further reducing the capacity or thinning the network.

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

Model	Test Accuracy (%)	Converged Iteration (Epoch)	Remarks
Base	38	700	Random noises
Optimal number of feature map (C1: 110, C2: 120)	42	700	-
With Momentum (γ = 0.1, *Learning Rate* = $1e^{-4}$)	-	Fail to converge	-
With Momentum (γ = 0.1, *Learning Rate* = $1e^{-5}$)	38	500	-
RMSProp	45	900	Large amount of random noise
Adam Optimizer (*Batchsize = 128*)	-	Fail to converge	Periodical spikes of noises
Adam Optimizer (*Batchsize = 512*)	43	800	-
Dropout (*keep prop* = 0.1)	-	Fail to converge	Large amount of random noise
Dropout (*keep prop* = 0.3)	-	Fail to converge	Large amount of random noise
Dropout (*keep prop* = 0.5)	30	800	Large amount of random noise
Dropout (*keep prop* = 0.7)	35	800	Large amount of random noise
Dropout (*keep prop* = 0.9)	35	400	Large amount of random noise

Part B: Text Classification

Introduction

In this section, we will be experimenting with CNN and Recurrent Neural Network (RNN) for the task of text classification. The dataset used for this experiment are paragraphs from entries in Wikipedia with each paragraph having a corresponding label for its category. The two networks will be implemented in both word and character level and resultant performance will be compared.

Experiments & Results

1. Design a Character CNN Classifier that receives character ids and classifies the input.

The character classifier consists of two convolution layers. The table below shows the layers and training properties of the CNN.

	C_1	C_2
Filter Number	10	10
Window Size	20x256	20x1
Padding	VALID	VALID
Activation	ReLU	ReLU
Pooling	Max, 4x4	Max, 4x4
Stride	2	2
Stride Padding	SAME	SAME
Learning rate	0.01	
Batch size	128	

Table 1: Character CNN Layers and Settings

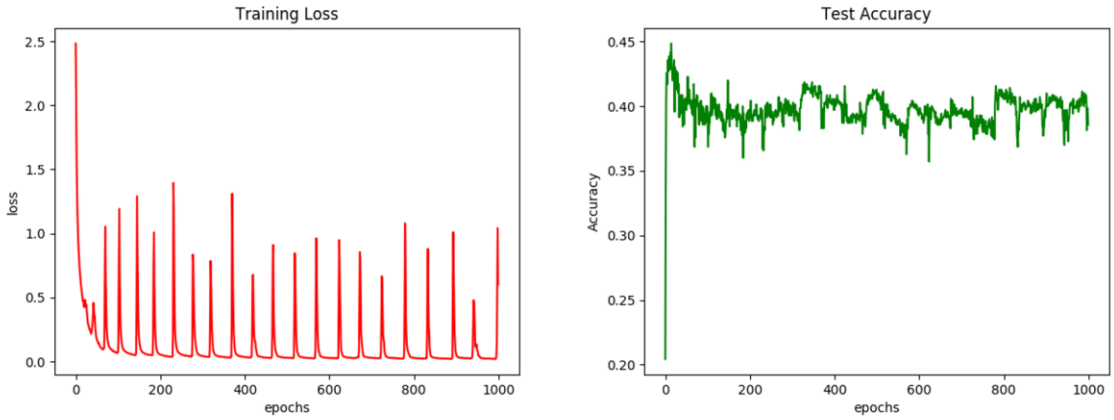

Figure 16: Training Loss and Test Accuracy of Character CNN

The two figures shown above are the training loss and test accuracy plots of the Character CNN classifier. The training loss starts to become noisy at around 70 epochs. The corresponding noise can be observed for the test accuracy as well. The spikes of noises are a consequence of the sampling of training data for Mini-Batch Gradient Descent training. Some mini batches may contain outlier data that causes surge in value in the cost function which explains the spiky trend as seen in the training loss plot. The effects cause by the outlier data can be reduced by increasing the batch size as this will reduce the influence of the outlier data on the calculation of the loss function.

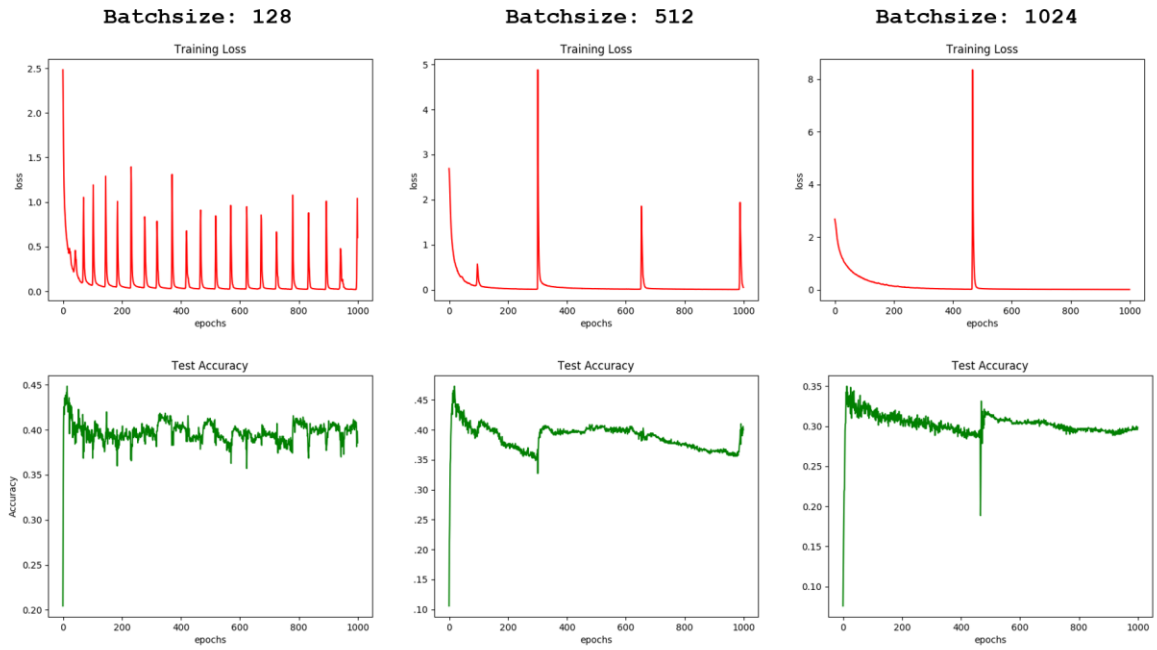

Figure 17: Performance comparisons for different batch size

From the plots as shown in Figure 17, it can be observed that the noise spikes are reduced as we increase the *Batchsize* parameter we used for sampling the data for Mini-Batch Gradient Descent training. **Note that as per the requirement of the assignment, the rest of the experiment will still be done with *Batchsize of 128.***

Lastly, because the classifier looks only at characters, it might not have enough information to classify the input thereby contributing to the noise as well.

2. Design a Word CNN Classifier that receives word ids and classifies the input. Pass the inputs through an embedding layer of size 20 before feeding to the CNN. The CNN has two convolution and pooling layers.

The table below shows the layers and training properties of the CNN.

	C_1	C_2
Filter Number	10	10
Window Size	20x20	20x1
Padding	VALID	VALID
Activation	ReLU	ReLU
Pooling	Max, 4x4	Max, 4x4
Stride	2	2
Stride Padding	SAME	SAME
Learning rate	0.01	
Batch size	128	

Table 2: Word CNN Layers and Settings

In addition, there is an **embedding layer** of size 20 before the input is fed into the CNN. The embedding layer is a learnable layer which converts one-hot vector of words representations (size of the vocab) to a vector of fixed length (embedding size) vectors.

Previously for the Character level classifier, One Hot Encoding was used. The reason was because we only had 256 possible characters. In this case, One Hot Encoding creates a vector of length 256 filled with ‘0’s and a ‘1’ at the index corresponding to the character. This creates a sparse 256x256 matrix in the end since each character requires one vector to represent. This is appropriate for the Character classifier for two reasons. Firstly, the matrix is small and would result in faster training of the model. Secondly, characters often do not have correlation with each other.

Embedding was used in the Word Classifier due to the bigger amount of data – we are taking whole words now instead of characters and there are much more than 256 unique words in our data set. The next important reason is that Embedding groups co-occurring items together since certain words always come after another. This is how word prediction works when you we are typing a message in our mobile phone.

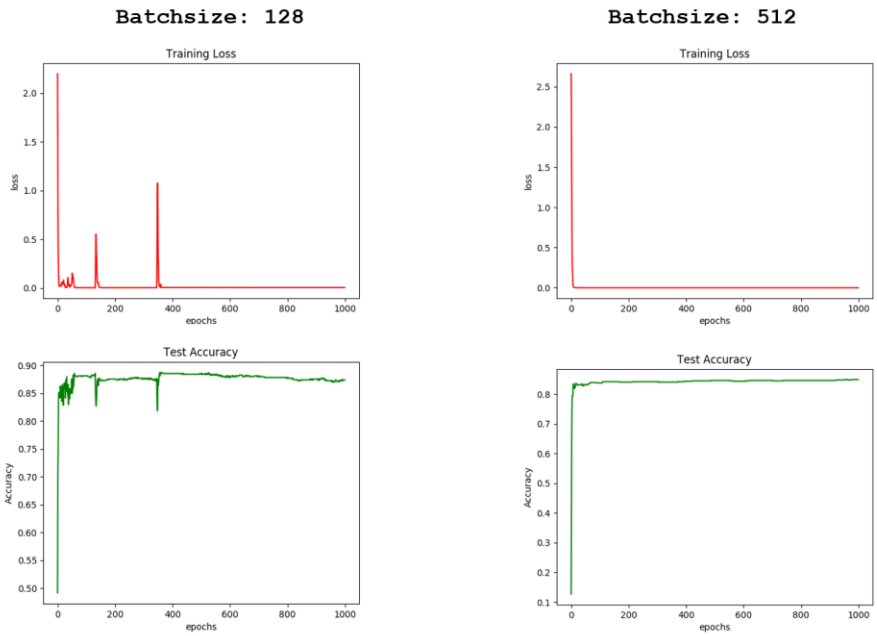

Figure 18: Performance comparison for different batch size

Figure 18 shows the performance of the word level classifier. The results proved that the word level classification is much more accurate than its character level counterpart for this dataset. Similar to the character level classifier, the word level classifier also suffers from the outlier problem as suffered by the word level classifier. However, for the word level classifier, the noise can be completely removed by increasing the *Batchsize* parameter to 512. Note that while the noise is removed when *Batchsize* is set to 512, the accuracy has reduced from 87% to 82%.

3. Design a Character RNN Classifier that receives character ids and classify the input. The RNN is GRU layer and has a hidden-layer size of 20.

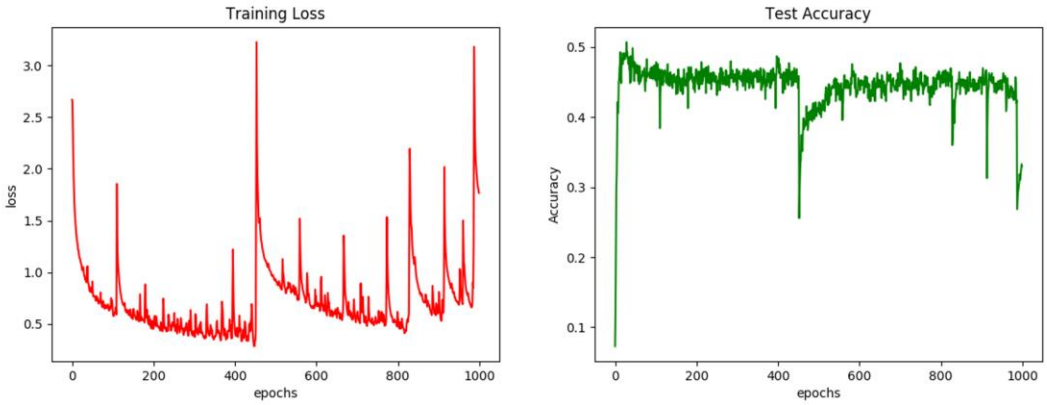

Figure 19: Training Loss and Test Accuracy for Character RNN

The character GRU based RNN classifier is highly unstable as seen in the Training Loss plot where the training loss value fluctuate a lot across the training Epoch. Early stopping is required at around 450 Epoch to prevent the model from diverging. The model also doesn't produce quality classification accuracy, with accuracy peaking at only 46%. The inefficiency of the model may be due to the limited information available at the character level to properly describe its corresponding category. This is also not aid by the relatively low complexity of the single layer structure of this GRU based RNN classifier.

4. Design a word RNN classifier that receives word ids and classify the input. The RNN is GRU layer and has a hidden-layer size of 20. Pass the inputs through an embedding layer of size 20 before feeding to the RNN.

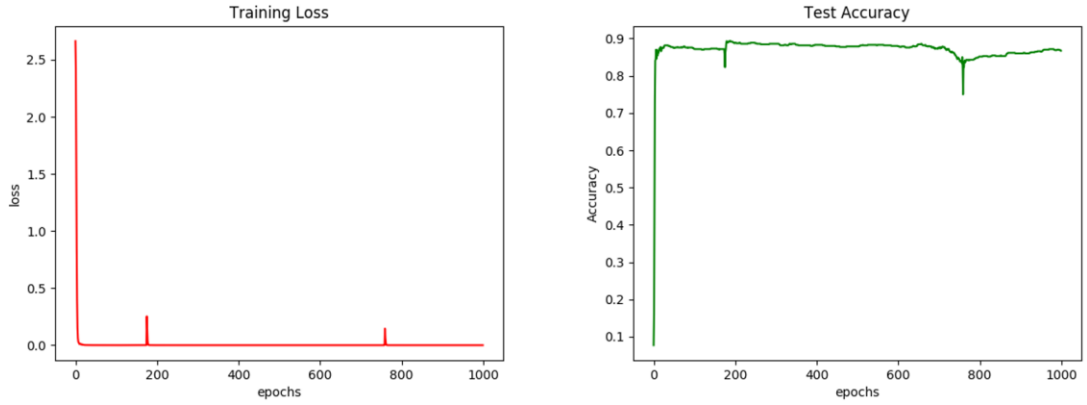

Figure 20: Training Loss and Test Accuracy for Word RNN

From Figure 20, it can be observed that the word GRU based RNN classifier produced a far more stable performance than its character level counterpart shown in Figure 19 previously. It is also almost twice as accurate compared to the character GRU based RNN classifier with test accuracy peaking at 88%.

5. Compare the test accuracies and the running times of the networks implemented in parts (1) – (4).

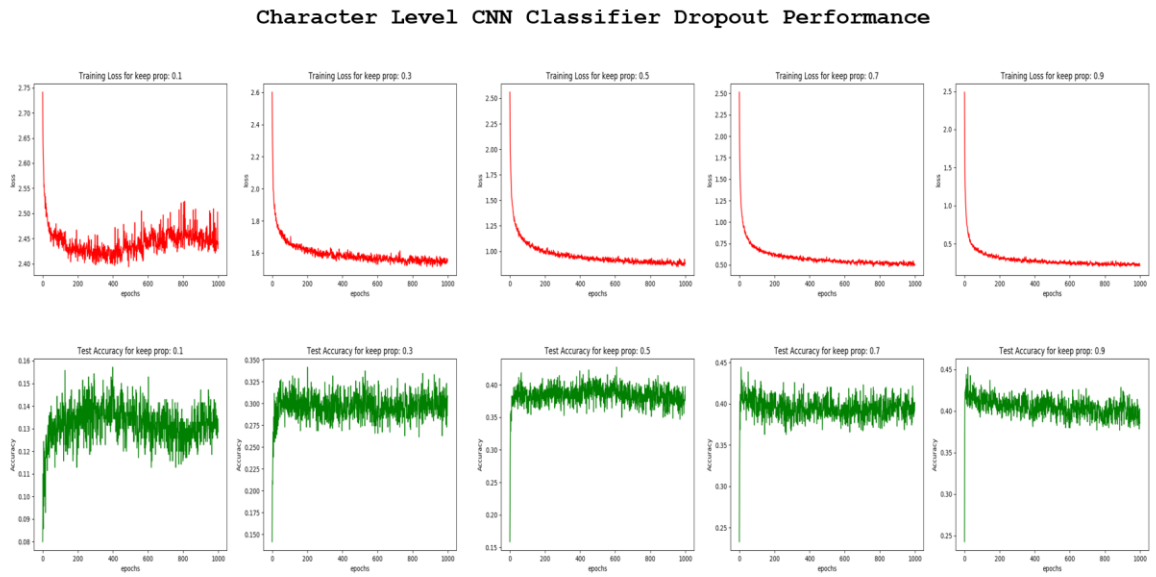

Figure 21: Performance of Character CNN with various dropout

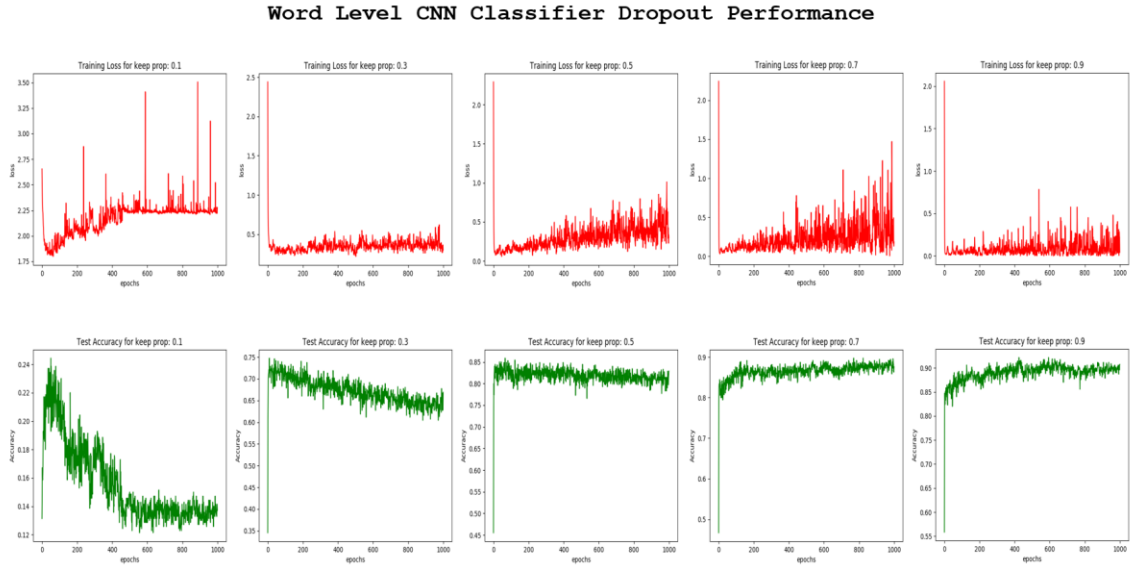

Figure 22: Performance of Word CNN with various dropout

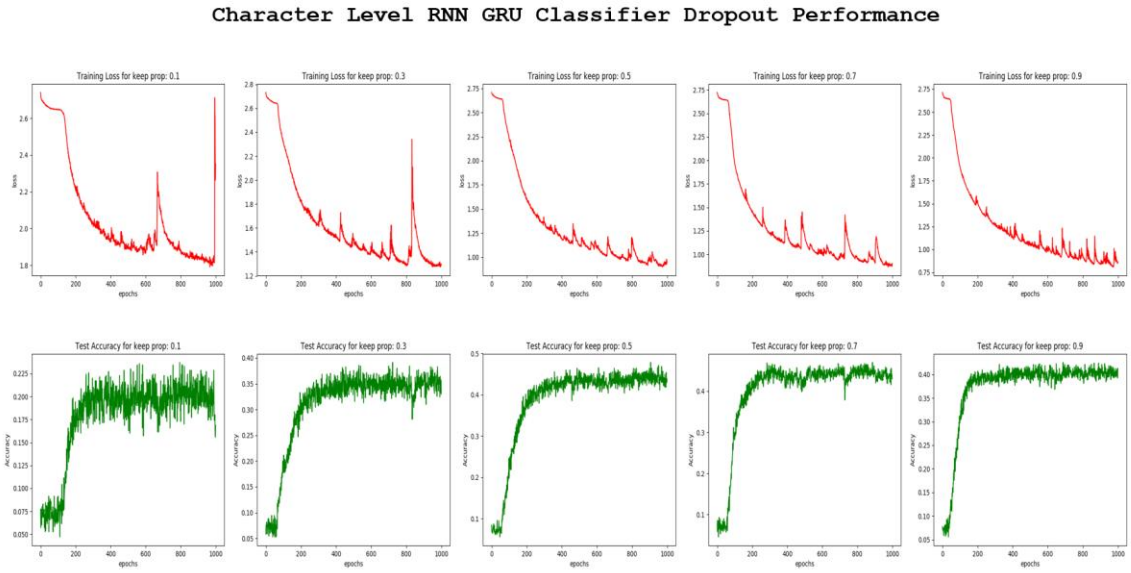

Figure 23: Performance of Character RNN with various dropout

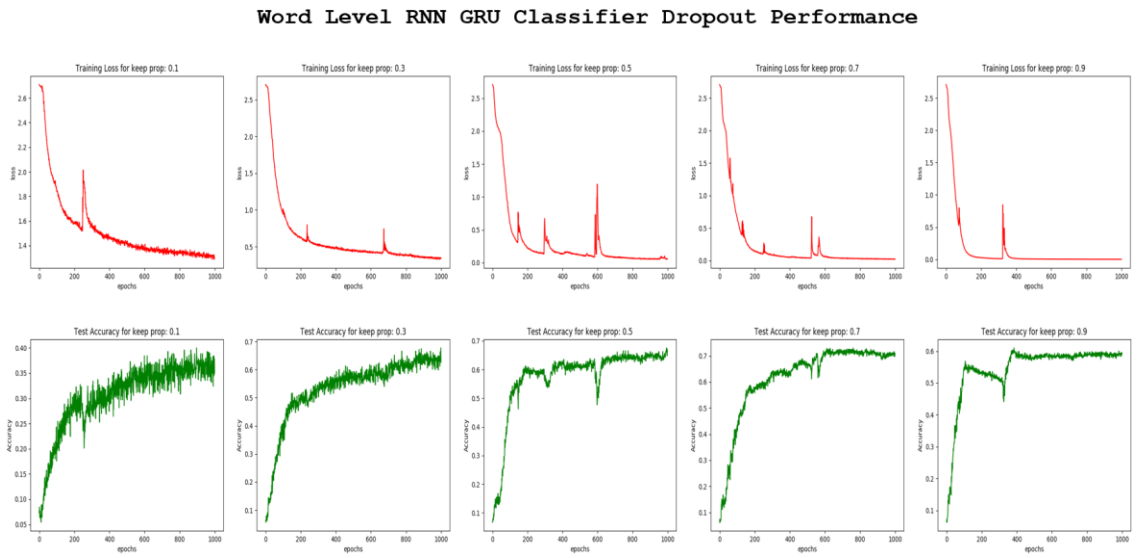

Figure 24: Performance of Word RNN with various dropout

Model	Test Accuracy (%)	Converged Iteration (Epoch)	Inference Runtime (ms)	Remarks
Character CNN				
- Keep Prob (0.1) , Batch size 128	-	Fail to converge	6.68	Noisy
- Keep Prob (0.3) , Batch size 128	30	800	6.73	Noisy
- Keep Prob (0.5) , Batch size 128	38	400	6.77	Noisy
- Keep Prob (0.7) , Batch size 128	40	700	6.65	Noisy
- Keep Prob (0.9) , Batch size 128	-	Fail to converge	6.67	Overfitting
- No Dropout, Batch size 128	-	Fail to converge	6.72	Fluctuates heavily
- No Dropout, Batch Size 1024	30	800	-	Less noisy
Word CNN				
- Keep Prob (0.1) , Batch size 128	-	Fail to converge	1.01	Noisy
- Keep Prob (0.3) , Batch size 128	-	Fail to converge	0.98	Increasing training loss
- Keep Prob (0.5) , Batch size 128	-	Fail to converge	0.97	Increasing training loss
- Keep Prob (0.7) , Batch size 128	-	Fail to converge	1.00	Increasing training loss
- Keep Prob (0.9) , Batch size 128	90%	900	1.01	Noisy
- No Dropout, Batch size 128	85%	400	1.01	Require early stop at 400 Epoch
- No Dropout, Batch Size 512	83%	100	-	Smooth

Character RNN					
-	Keep Prob (0.1)	-	Fail to converge	18.79	Noisy
-	Keep Prob (0.3)	-	Fail to converge	19.11	Multiple spikes
-	Keep Prob (0.5)	45%	970	19.15	Descending spikes
-	Keep Prob (0.7)	45%	990	19.13	Descending spikes
-	Keep Prob (0.9)	40%	960	19.14	Descending spikes.
-	No Dropout	45%	300	19.19	Noisy after epoch 440. Early stop at 250 gives best result
Word RNN					
-	Keep Prob (0.1)	38%	980	17.8	Single large spike at epoch 230
-	Keep Prob (0.3)	65%	1000	17.9	Gradual descend with two spikes
-	Keep Prob (0.5)	65%	900	18.2	Three spikes, gradual descend
-	Keep Prob (0.7)	60%	400	18.1	Small spikes, gradual descend.
-	Keep Prob (0.9)	60%	400	18.4	Small spike at epoch 380, converges at 400
-	No Dropout	90%	200	18.6	Early stop at 200 recommended

Table 3: Performance comparison for various CNN and RNN classifier

From the results it can be summarized that for both CNN and RNN the word level classifiers perform better than their character level counterparts in terms of test accuracy. Both of the network type manged to attained peak accuracy of 90%. The CNN Word level model performed the best with 0.1 dropout rate. While the RNN models performed the best without any dropout. It is worth mentioning that the best performing CNN model took far longer (900 Epoch) to converge compared to the best performing RNN model (200 Epoch).

Another point to consider for the choosing of the model is the inference runtime (the time taken to do a forward computation of the computational graph). All the models are tested on the Nvidia Tesla V100 (16GB) GPU. It can be observed that the Word CNN models ran the fastest with an average inference time of 1ms. It can also be seen that the RNN models are almost twenty times slower compared to the Word CNN models.

6. For RNN networks implemented in (3) and (4), perform the following experiments with the aim of improving performances, compare the accuracies and report your findings.

a. Replace the GRU layer with (i) a vanilla RNN layer and (ii) a LSTM layer

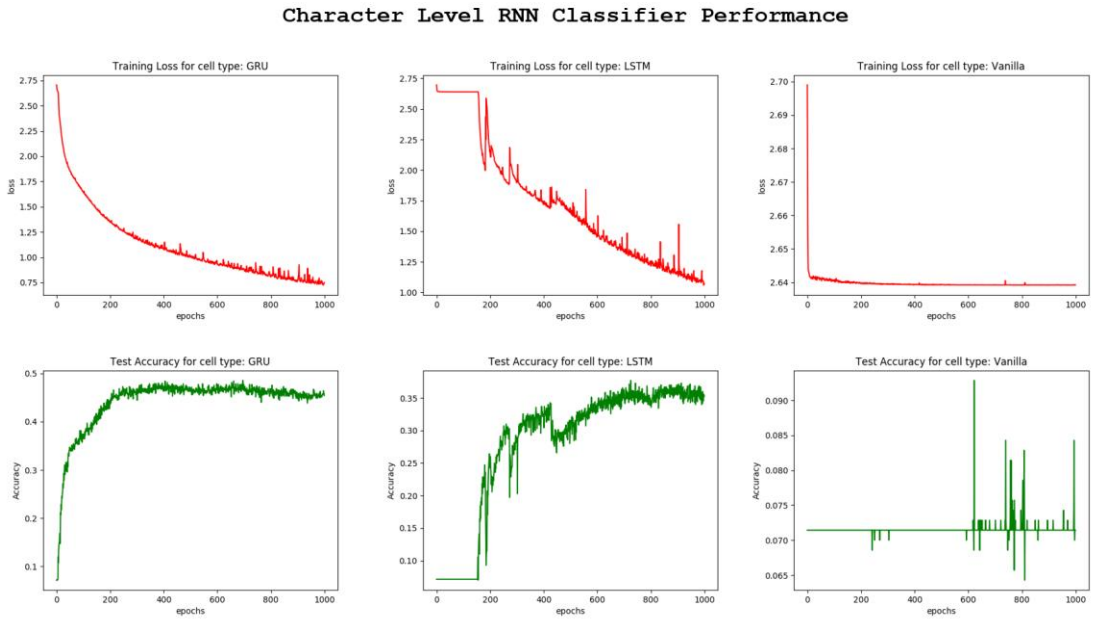

Figure 25: Performance comparisons of GRU vs LSTM vs Vanilla cell types for character level classifier

From the results as shown in Figure 25, it can be observed that the GRU variant of the character level classifier still produces the best performances out of the 3 cell types. The LSTM cell type only start to reduce in loss after 180 Epoch and the loss function is highly noised. The Vanilla cell type is the worst performing out of the three-cell type, the loss is not reduced, and it remained stagnant at a high of 2.64 even after training for 1000 Epoch.

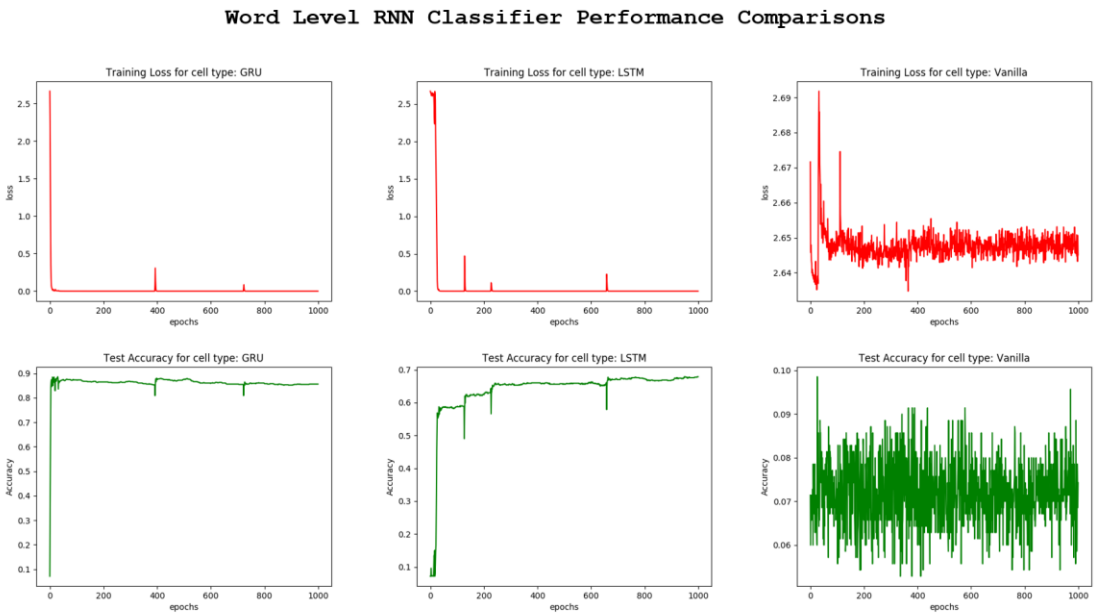

Figure 26: Performance comparisons of GRU vs LSTM vs Vanilla cell types for word level classifier

Similar to the character level classifier, the GRU cell type still produces the best performance out of the three cell types in terms of accuracy and stability. However, the LSTM cell type is seen to be more stable compared to its counterpart for the character level classifier. The Vanilla cell type still produces erratic performances and can be deduced to be not suitable for this dataset.

b. Increase the number of RNN layers to 2 layers

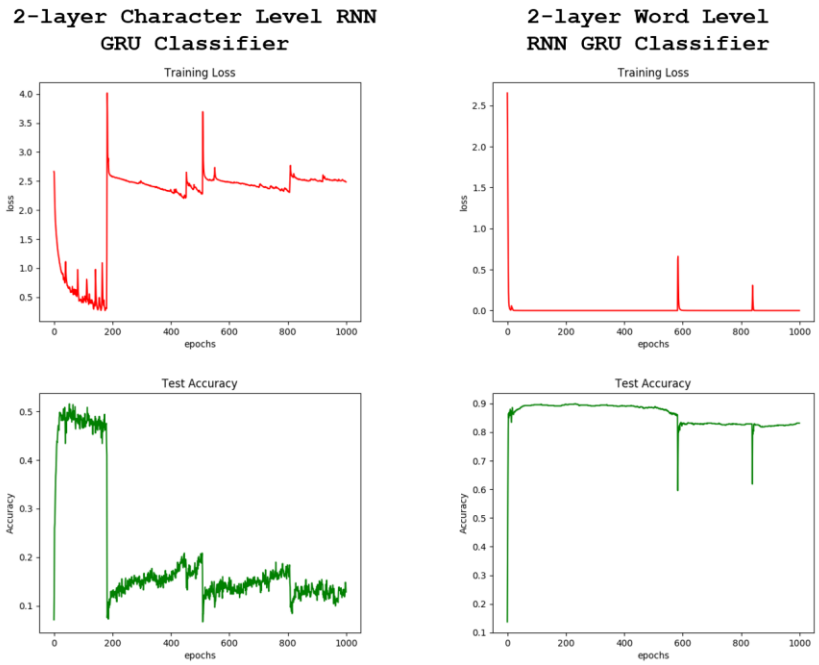

Figure 27: Performance of both RNN models when increasing number of layers to 2

From the results, it can be observed that the 2-layer character level classifier performed slightly better than its single layer counterpart (question 3) with accuracy increased from 47% to 50%. However, early stopping at approximately 100 Epochs is needed to prevent the model from diverging.

The 2-layer word level classifier out-perform its single layer counterpart (question 4) by a huge margin with accuracy increasing from 82% to 88%. Early stopping is also required at approximately at 100 Epochs to prevent the model from diverging.

c. Add gradient clipping to RNN training with clipping threshold = 2.

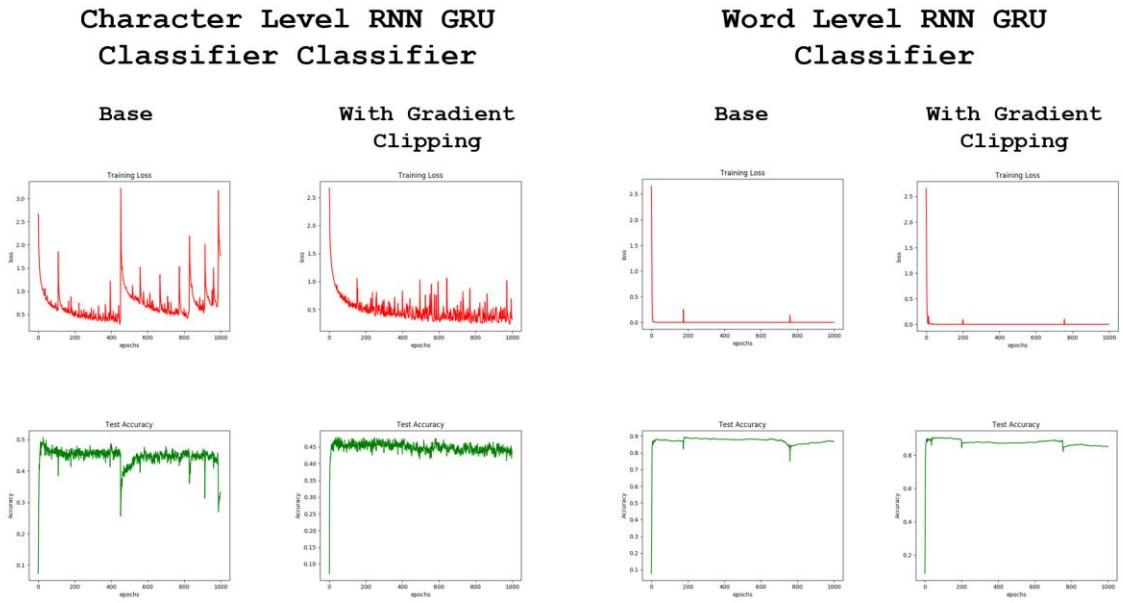

Figure 28: Gradient clipping results for Character RNN and Word RNN

Gradient clipping is used to prevent the over growing of the gradient calculated through the loss function as the training progresses. Loglikelihood-losses needs to be clipped, as it may evaluate near log(0) for bad predictions/outliers in dataset, thus causing exploding gradients. From Figure 28 shown above, it can be observed that for the character level RNN based GRU classifier, the sharp spike in the loss function after 430 Epoch can be eliminated with gradient clipping. The word level RNN based GRU classifier does not experienced exploding loss, therefore the gradient clipping does not make a significant difference to the performance.