NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

CE4042: Neural Network & Deep Learning

Individual Assignment 2

Name: Do Anh Tu

Matriculation No.: U1721947F

Date: 9th November 2020

Contents

1. Introduction

Multi-layer perceptron is a classical example of neural networks, simple and provide us great understanding of the concepts in this field of machine learning. While MLPs work well for simple datasets and tasks, they become insufficient for more complex tasks such as computer vision and natural language processing. The former often required learning features of image, which can explode the number of parameters needed to train MLPs, while the later requires sequence processing that classical MLPs do not possess.

Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are better choice for these two high level tasks. CNNs are the go-to method for predictive modelling of images as they can extract patterns, location invariantly, in images and understand the spatial relations in data well (pixels in images). CNNs might be used for text-based tasks and this will be shown in this project. RNNs on the other hand, are geared with persistence unit that allow good modeling of temporal relations, which gives it good performance in sequence predicting like NLP applications.

In this assignment, the construction, and effects of various hyperparameters on the performance of these 2 types of network will be investigated.

2. Main Objectives

The assignment has two parts A and B. Part A deals with Image Recognition while B is about Text Classification. Given the two tasks, the goal for our experiments are *(i) to investigate the various hyperparameters which affect the model performance* and *(ii) produce optimal models given our knowledge on the optimal hyperparameters*.

a. **Image Recognition**

The dataset for this part is the popular CIFAR-10 dataset used for benchmarking CNNs and image recognition (classification). The original dataset has 60000 RGB images with 32x32 dimension and divided into 10 classes for supervise learning. In our experiment, for convenience of time in training, we will only be training on 10000 samples from this dataset and testing on 2000 other samples.

Our goal for this part A is to build CNNs to predict the label within 10 classes above, given the input is an image of size 32x32x3. Once the baseline model is obtained, we will be investigating the effect of *(i) number of feature map filters of convolutional layers, (ii) different optimizers* and *(iii) dropout regularization for fully connected layers* on the overall performance of the model in this task.

Our main metric for comparison will be **test accuracy of the model**s, where the **maximum** value amongst models being evaluated is to be selected.

b. **Text Classification**

The dataset for text classification is a collection of first paragraphs from Wikipedia entries and the 15 different labels for the categories of these entries. The training set has 5600 instance of such entries and the test set has 700 instances.

Our main goal for this part B is to build at character-level and word-level CNNs and RNNs that can predict the category of the Wikipedia entry given the sequence of texts in the dataset. Then, we will be *investigating the effect of dropout regularization on these two* *types of neural networks*. Lastly for RNNs, we will be investigating the difference in performances with *(i) different type of RNN cells*, *(ii) stacked architecture of RNNs* and *(iii) gradient clipping of optimizer*.

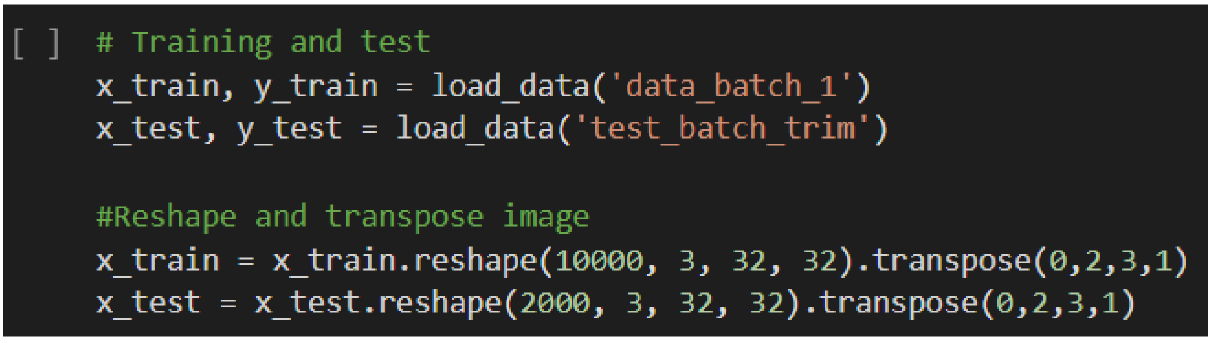
Like part A, our main metric for comparison will be **test accuracy** for selecting the optimal model.

3. Methodology

3.1. Part A

3.1.1. Preprocessing

The only processing of image from the given dataset is to obtain the original images in train and test set before feeding into the model (we removed reshape layer in the model construction, and do that after loading the datasets). The unflatten step is shown in the code below (Figure 1).



```
# Training and test
x_train, y_train = load_data('data_batch_1')
x_test, y_test = load_data('test_batch_trim')

#Reshape and transpose image
x_train = x_train.reshape(10000, 3, 32, 32).transpose(0,2,3,1)
x_test = x_test.reshape(2000, 3, 32, 32).transpose(0,2,3,1)
```

Figure 1. Code for unflatten the input images after loading dataset.

3.1.2. Model

Baseline model (question 1A)

The first convolution layer has 50 filters, size 9x9, padding 'VALID' and 'Relu' activation. It is followed by a max pooling layer of size 2x2, stride step 2 and 'VALID' padding. The second convolution layer has 60 filters, size 5x5, padding 'VALID' and 'Relu' activation. The next layer is a max pooling layer of similar hyperparameter as the first one max pooling layer. The output of the second max pooling layer is flattened out and parsed into a fully connected layer of width 300 neurons and no activation. The output is fed to the last fully connected layer to produce the logit for prediction (so size = 10 neurons).

Other hyperparameters of this model are epochs = 1000, batch_size = 128 and learning rate alpha = 0.001. The base optimizer is SGD and loss function is Sparse Categorical Cross Entropy.

Dropout (question 3A)

Whenever dropout is required, 2 additional dropout layers are added to the baseline model. The position of addition of these layers are *1) After flattening of Max Pooling 2 layer and before the first FC* and *2) after the first FC and before the output layer*. This is implemented as a Boolean flag guarded segment in definition of the model.

Grid search for optimal number of filters per convolution layer (question 2A)

The 10 values of number of filters is put in 2 arrays (1 for convolution 1 and 1 for convolution 2). Then a for loop going through these 2 arrays such that 25 combinations are searched and implement models with the corresponding number of filters in the respective layers.

Grid search for optimal optimizer and regularization strategy (question 3A)

The optimal number of filters from the previous step is kept for this part. The construction of model is same as that in baseline and dropout model. Other parameters are constant, except for the optimizer used. Here we are trying out 4 optimizers: SGD with momentum = 0.1, RMSProp, Adam and SGD with dropout = 0.5. The model with the highest test accuracy is the optimal model for this grid search.

3.3.3 Feature map

We build an activation model from the built and trained model in part 1A and specified the output of the layers we want to peek at. The layers are Conv1, MaxPool1, Conv2 and MaxPool2.

The code for this segment is taken from here.

3.3.4. Criteria for evaluation

We will be evaluating the model based on their test accuracy and only the model with highest test accuracy shall be deemed as selectable for next step (training or just usage).

The assumption is whatever results we get from the model after 1 run as it is (no repetition for bagging etc.) due to time constraint and workload. However, as training is conducted on **GPU** (which does not guarantee seeding of random state in its library implementation), **the result is expected to vary from runs to runs**.

3.2. Part B

3.2.1. Preprocessing

The preprocessing steps for character-level models started with remapping all character possible in the sentences to character ids, which are a number per unique character. Then the sentence in the dataset is limited to the same amount of characters called maximum document length. Sentence with smaller number of characters is zero padded and truncated if larger. The word ids are then one-hot encoded before feeding to the model. For CNNs, we will need to add in one more dimension to represent the channel dimension (=1) for the input before feeding them to the first convolution layer

The preprocessing steps for word-level models are as following. First all words in the text are found and remap to word ids, which are a number per each unique word. Then the sentence in the dataset needs to be limited to the same maximum document length constant. If the sentence is shorter than this, they are padded with zero and truncated otherwise. The next step is to pass these arrays of word ids through an embedding layer which is learnable. This layer converts the word representation to a vector of fixed length (the embedding size hyper parameter) that better represent the input (smaller than one-hot vector, manageable and may represent the relationship between words better). For CNNs, we will need to add in one more dimension to represent the channel dimension (=1) for the input before feeding them to the first convolution layer

3.2.2. Models

Common Hyperparameters

Some fixed hyperparameter in these experiments are training epochs limited to 250, character/word input with max size to be 100, Adam Optimize, batch_size of 128 and learning rate of 0.01. Embedding size is 20 and dropout rate is 0.

Character CNN

The main architecture of the character CNN (exclude the layers for preprocessing above) starts with the first convolution layer with 10 filters of 20x256 kernel, VALID padding and Relu activation. The first max pooling layer follows with a 4x4 window, 2 stride step and SAME padding. The next convolution layer has 10 filters of 20x1 kernel, VALID padding and Relu activation. The second max pooling layer has the same specifications as the first one. The output of this last layer is flattened and passed to an FC to produce the softmax activated output for class prediction.

Word CNN

The main architecture of the word CNN (exclude the embedding layer for preprocessing above) starts with the first convolution layer with 10 filters of 20x20 kernel, VALID padding and Relu activation. The first max pooling layer follows with a 4x4 window, 2 stride step and SAME padding. The next convolution layer has 10 filters of 20x1 kernel, VALID padding and Relu activation. The second max pooling layer has the same specifications as the first one. The output of this last layer is flattened and passed to an FC to produce the softmax activated output for class prediction.

Character RNN

The character-level RNN has similar preprocessing steps like the character CNN, without the adding of 1 more dimension for the channel. We will be using an RNN keras wrapper that has a specific GRU cell at it heart to implement the baseline GRU model. The hidden size for this layer is 20 and vocab size is assumed to be 256. This GRU layer is then followed by a fully connected layer with softmax layer to produce the multilabel predicted output.

Word RNN

The word-level RNN has similar preprocessing steps like the word CNN, without the adding of 1 more dimension for the channel. We will be using an RNN keras wrapper that has a specific GRU cell at it heart to implement the baseline GRU model. The hidden size for this layer is 20 while vocab size is to be determined from preprocessing step. This GRU layer is then followed by a fully connected layer with softmax layer to produce the multilabel predicted output.

With Dropouts

Whenever dropout regularization is required, we add in a dropout layer with dropout rate = 0.5 between the flattening of max pool 2 output and the fully connected layer producing softmax activation.

Vanilla RNN and LSTM

The implementation of Simple RNN and LSTM can be achieved by replacing the GRU cells in Character and Word RNN above with the corresponding cells as specified in the Tensorflow and Keras API.

Stacked RNNS

The stacking of RNN cell can be achieved with an array storing successive RNN layers to be stacked and passing it through the StackedRNNCells API constructor of Keras to create an RNN layer with stacked cell inside. Other hyper parameters should be the same as the baseline model.

Gradient Clipping

Gradient clipping of the Adam optimizer is done by specifying the clip value in the construction for the optimizer (clipvalue = 2).

3.2.3. Criteria for evaluation

Similar to 3.3.4

4. Experiments and Results

4.1. Part A

4.1.1. Question 1

A baseline CNN model of the specifications in 3.1.2 is trained with SGD, 1000 epochs, batch size 128 and learning rate alpha = 0.001. The results and feature maps required are shown below.

a. Plot training and test costs, accuracies against training epoch.

The training and test costs for the baseline model is shown below in *Figure 2*.

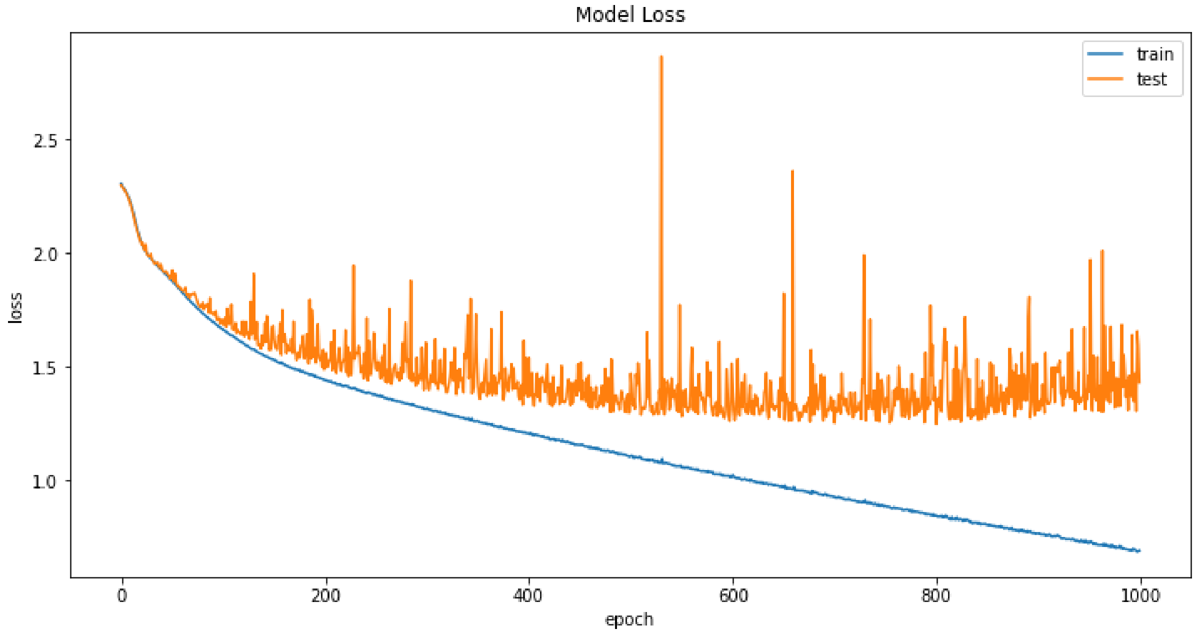


Figure 2. Train and Test Loss for baseline model (Q1)

The training and test accuracies for the baseline model is shown in the *Figure 3*.

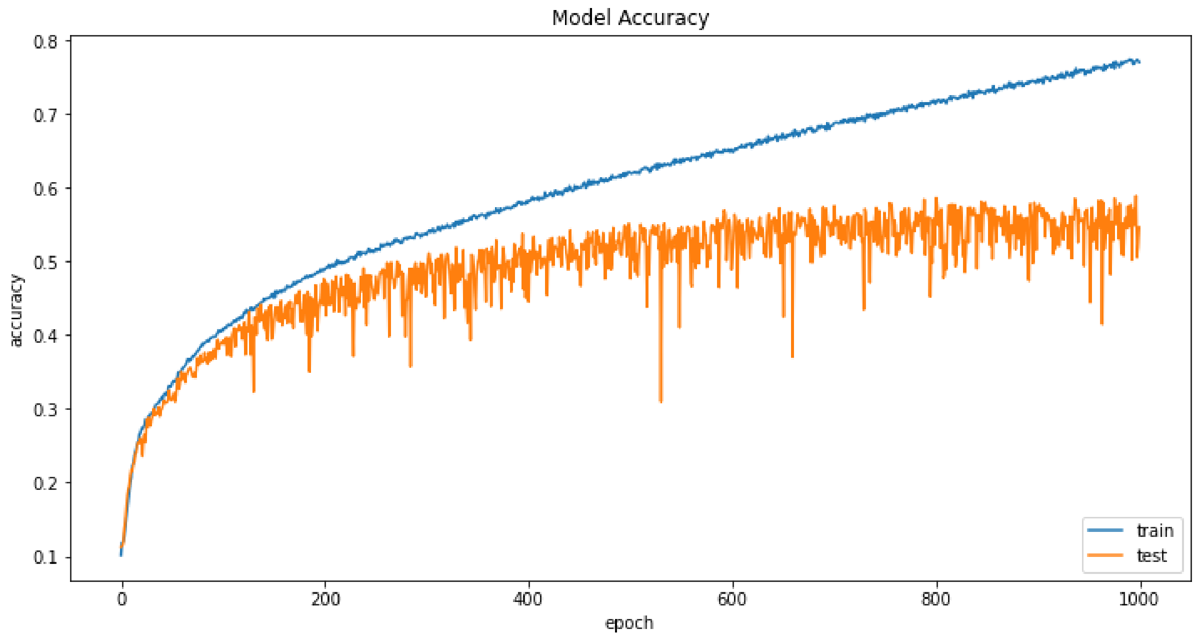


Figure 3. Train and test accuracies for baseline model

We obtain the best test accuracy for this model to be **0.5890** (58.9%). We can also see that both test loss and accuracy start to overfit at epoch = 200 and the overfitting is increasing as more epoch is trained. The train loss and train accuracy seem not to be converging; however, the test loss and accuracy converges at about 500 epochs.

b. Plot the feature maps at the convolution layers, max pooling layers and the input image to the model for the first 2 test images

The input image (as seen by the first convolution layer) and the feature maps corresponding to conv1, maxpool1, conv2, maxpool2 is shown in *figure 4*.

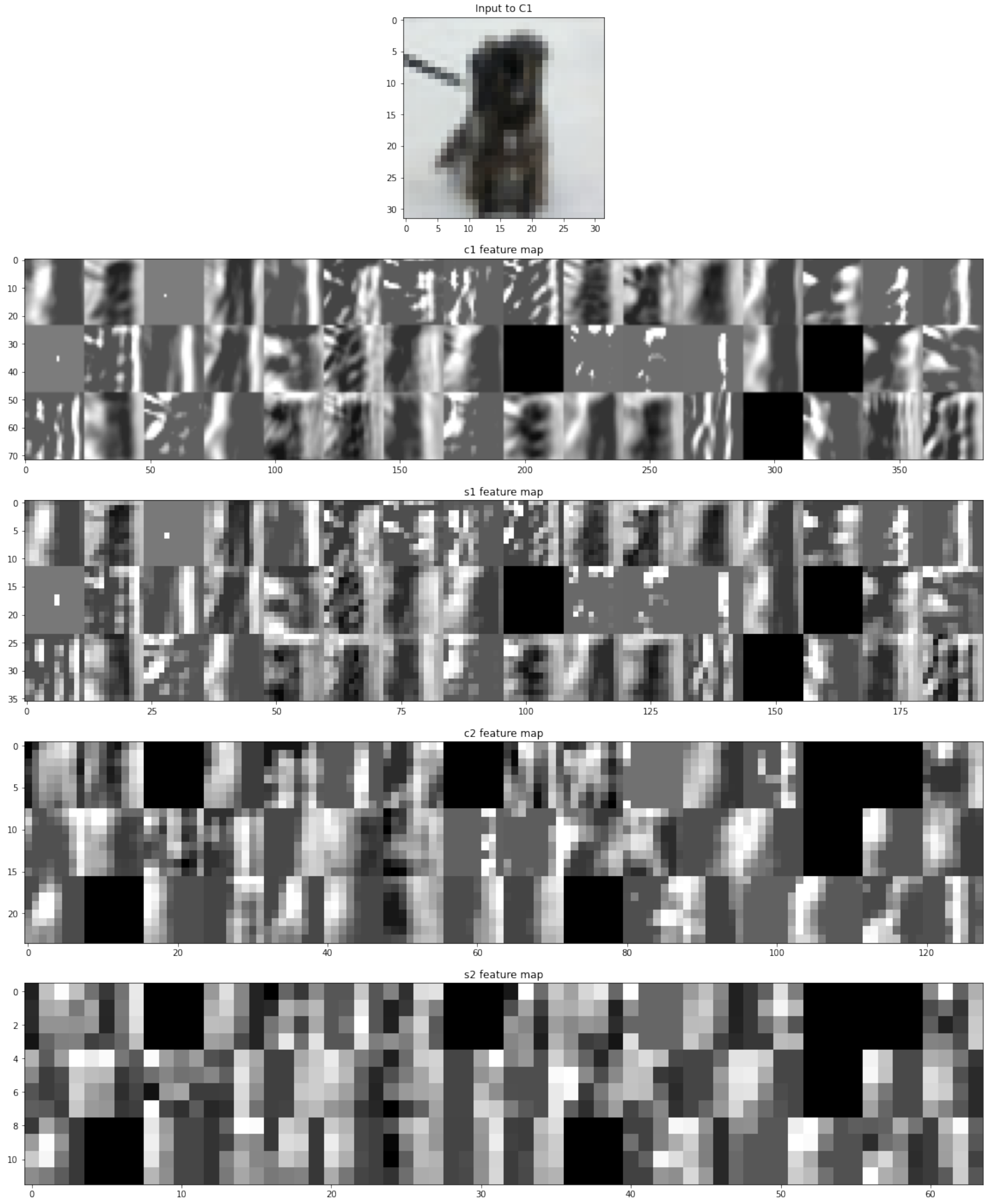


Figure 4. Input Image and Feature Maps for first test image

Likewise, the feature maps and input images (in RGB) for the second image of test set is shown in *figure 5*.

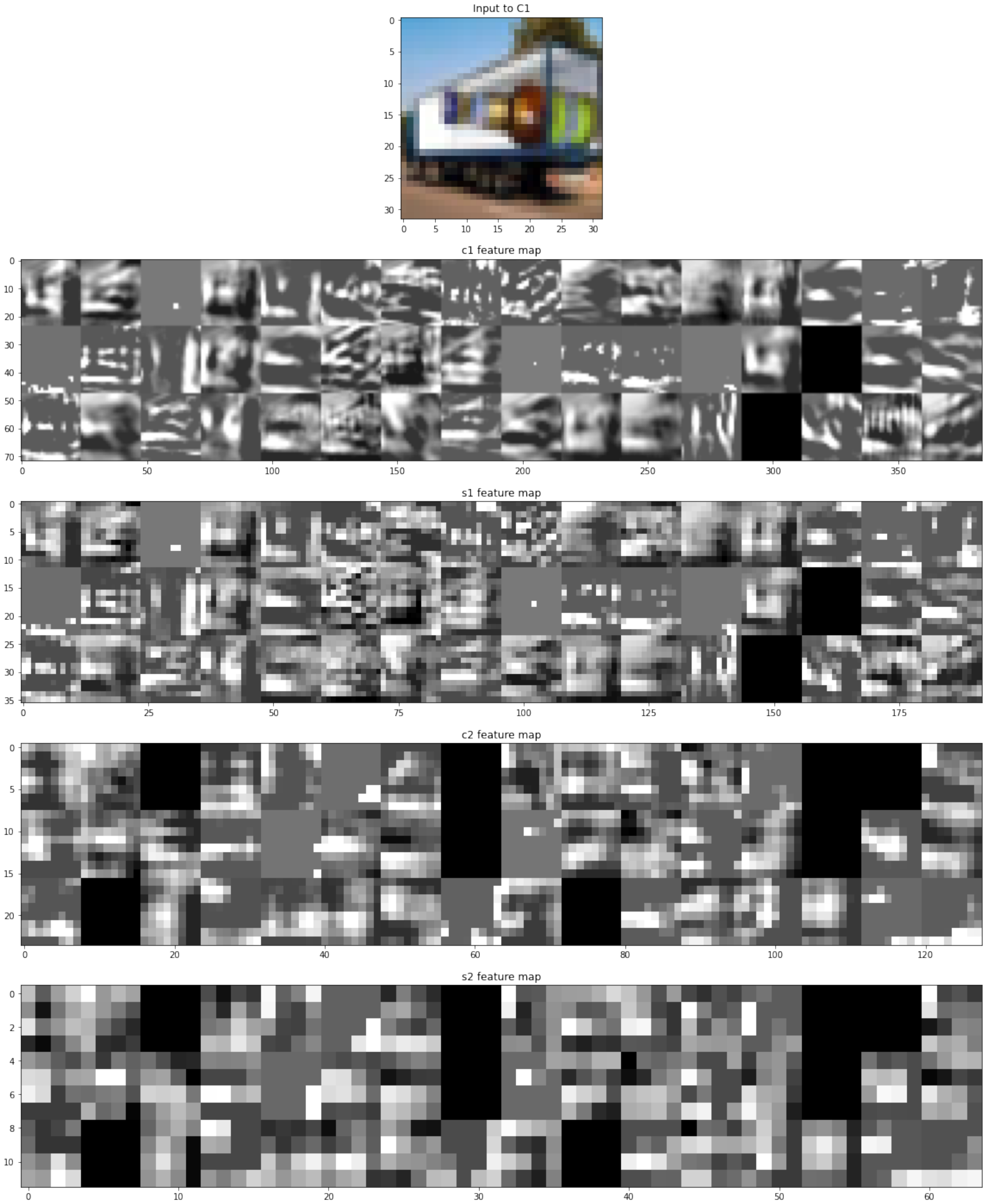


Figure 5. Input Image and Feature Maps for the 2nd test image

In the first convolution layer (c1), most of the full shapes of the input images are retained in the feature map. This is shown in *figure 4*, 1st row, 2nd and 4th images or *figure 5*, 1st row, 1st and 2nd images in the c1's feature map as the shape of the "dog" and "truck" is very similar to the input image shape. There are several filters that are not activated and are left blank (black color). This makes sense as for lower level of convolution layers, the general features such as the edges in the images are extracted, hence the generic shape of the object is maintained.

The first max pooling's (s1) output looks like the feature maps extracted after convolution layer 1, however it is in lower resolution as max pooling is supposed to under sample its input as expected. Again, this can be very clear *figure 4*, 1st row, 2nd and 4th images or *figure 5*, 1st row, 1st, and 2nd images of s1 feature map.

From the second convolution onwards, the activations produced shown in the set of feature maps (c2 and s2) becomes more and more abstract and harder for us to interpret. These feature maps detect and represents higher level features such as a single corner or angle, which might be more unique to the classes of images. At the second max pooling layer, the image has been under sampled so much, it is hard to tell what the pixels of feature maps are representing.

Hence, we have observed an expected behavior of CNNs where they can extract features from general, low-level to nuance, high level as the depth of network increased, where each feature map in the deeper level extracted a more high level feature related to the class we are trying to train to classify.

4.1.2. Question 2

Our task is to conduct a grid search for optimal combination of number of filters for convolution layer 1 and convolution layer 2. Search space is **{10,30,50,70,90} for the first convolution layer and {20,40,60,80,100} for the second convolution layer**.

After building models for these 25 combinations, we observe a plot of test accuracy vs combination of number of filters as shown in *figure 6*. The detailed, numeric results are shown in *table 1*.

Table 1. Results for Grid Search

No.	Num of Filter (Conv1)	Num of Filter (Conv2)	Test Accuracy
1	10	20	0.54400
2		40	0.55000
3		60	0.54550
4		80	0.55500
5		100	0.55600
6	30	20	0.56950
7		40	0.58200
8		60	0.58800
9		80	0.56550
10		100	0.58350
11	50	20	0.57150
12		40	0.58650
13		60	0.58300
14		80	0.58600
15		100	0.58550
16	70	20	0.58900
17		40	0.59100

18		60	0.60000
19		80	0.58700
20		100	0.60100
21	90	20	0.59250
22		40	0.59950
23		60	0.59950
24		80	0.58450
25		100	0.58600

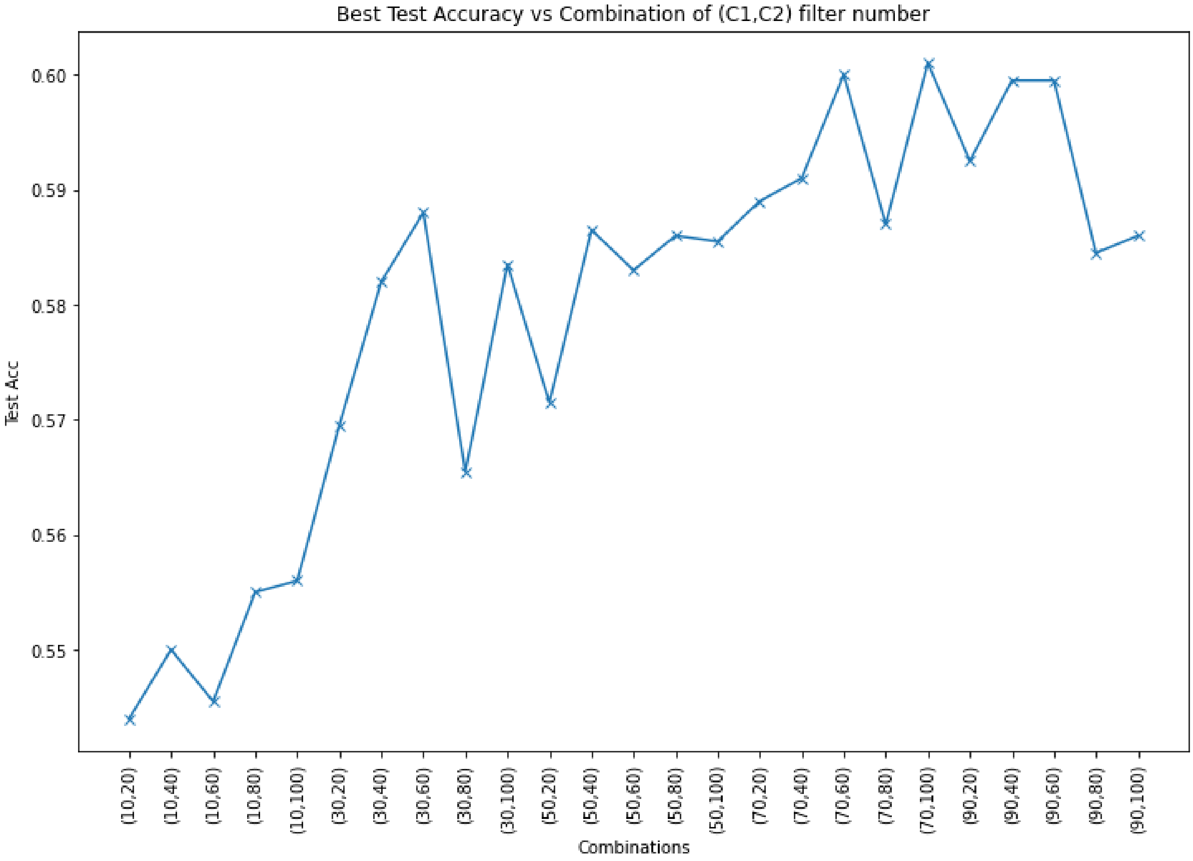


Figure 6. Best Test Accuracy vs Combination of number of filters in conv1, conv2

From table 1 and Figure 6, we can see that the combination number of filters for (conv1, conv2) of **(70,100) has the best result** which is **0.601**.

The trend observed here is when number of filters for each convolution layer increases, the best test accuracies generally increase. *This is expected since having higher number of filter maps for each of the convolution layer, the model can extract more features that better generalized the object being classified, hence increasing the performance of the model*. At certain number of filters, the lift in test accuracy from increasing the number of filters in each layer becomes saturated (from 70,40 onwards in figure 6), which led to oscillation in test accuracy.

4.1.3. Question 3

In this question, we investigate the effects of different optimizers and regularization techniques on the performance of the optimal model determined in question 2.

a. Adding momentum term = 0.1 for SGD

The training and test losses for this SGD-momentum optimizer are plotted against the training epochs and gives rise to the plot in *Figure 7*.

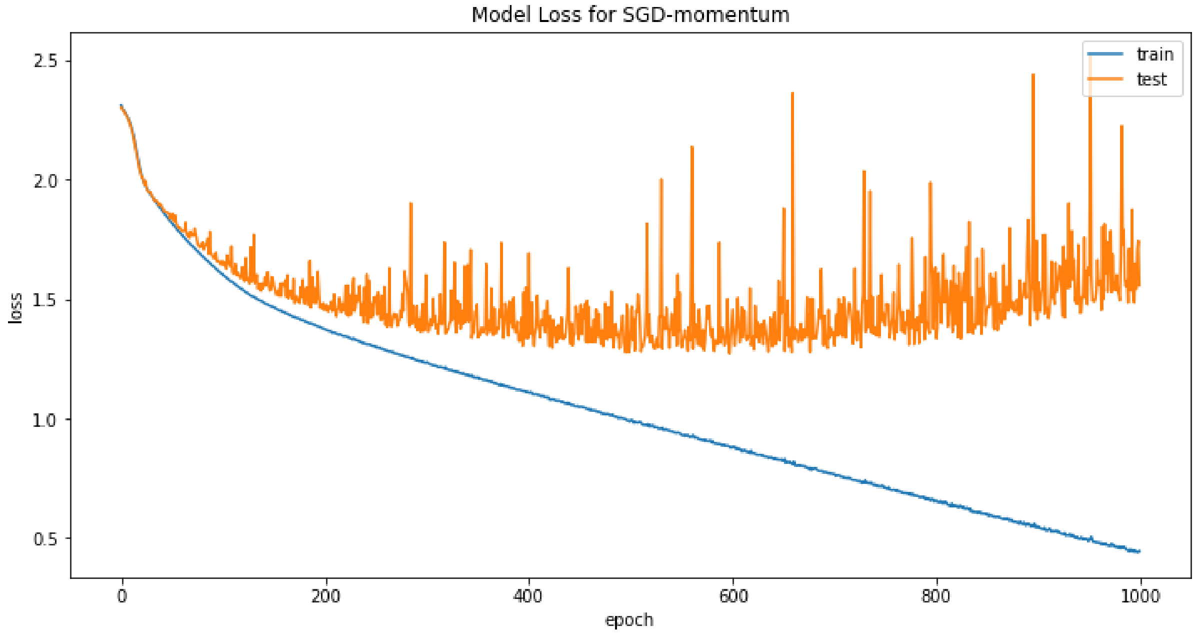


Figure 7. Train and Test cost vs Epochs for SGD-momentum 0.1

The training and test accuracies for this SGD-momentum optimizer are plotted against the training epochs and gives rise to the plot in *Figure 8*.

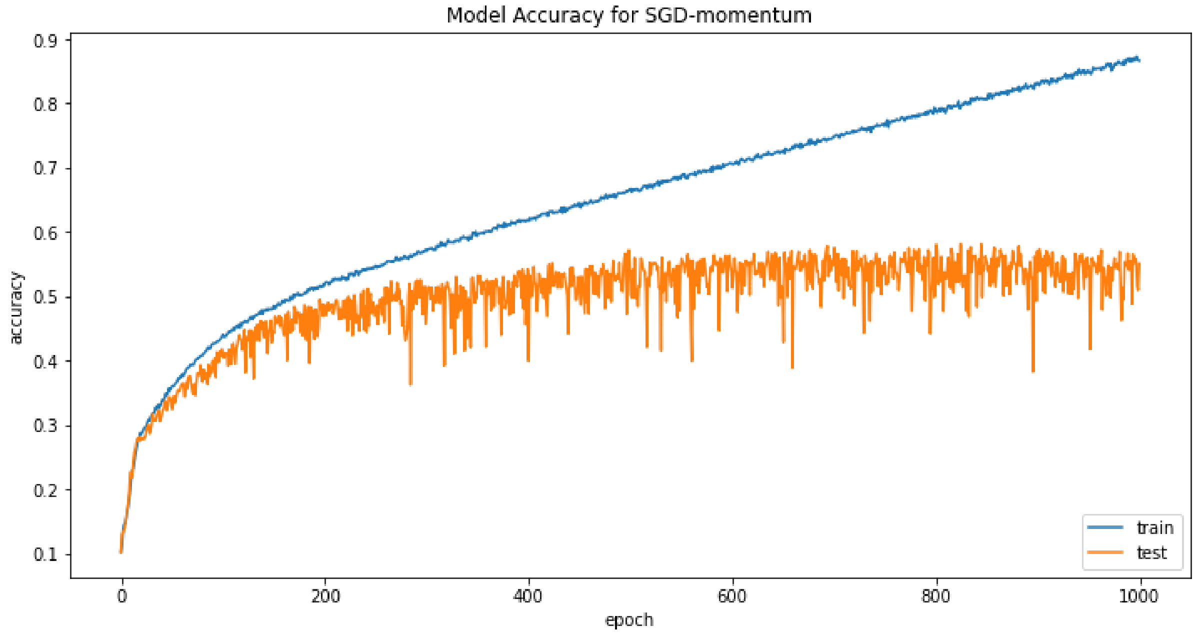


Figure 8. Train and Test accuracies vs epochs for SGD-momentum 0.1

The best test accuracy for this model is **0.58150**. The model's test loss and test accuracy start to converge at about 380 epochs. Overfitting occurs from about epoch 200 onwards for both metrics. Training loss and training accuracy seems not to be converging in 1000 epochs, as they are continuously decreasing and increasing with each epoch respectively. This trends are very similar to what observed in vanilla SGD in 4.1.1

b. RMSProp optimizer

The plot of training and test losses for RMSProp vs the training epochs are shown in *figure 9* below.

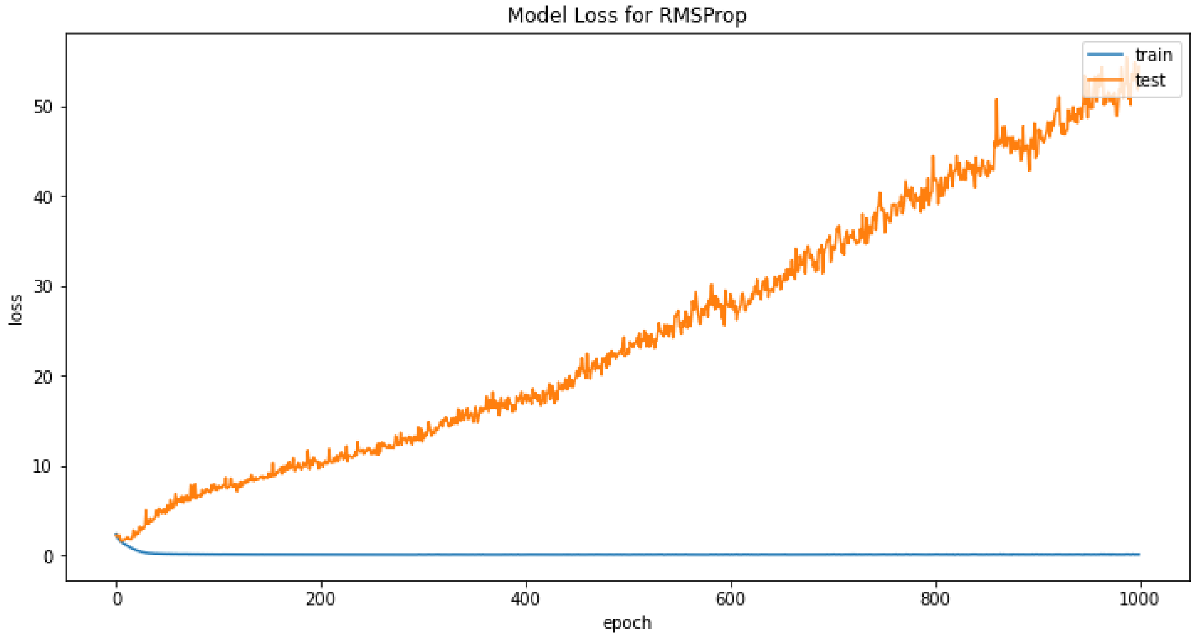


Figure 9. Training and test costs for RMS prop optimizer model

The plot for training and test accuracies for RMSProp vs the training epochs are shown in *Figure 10*.

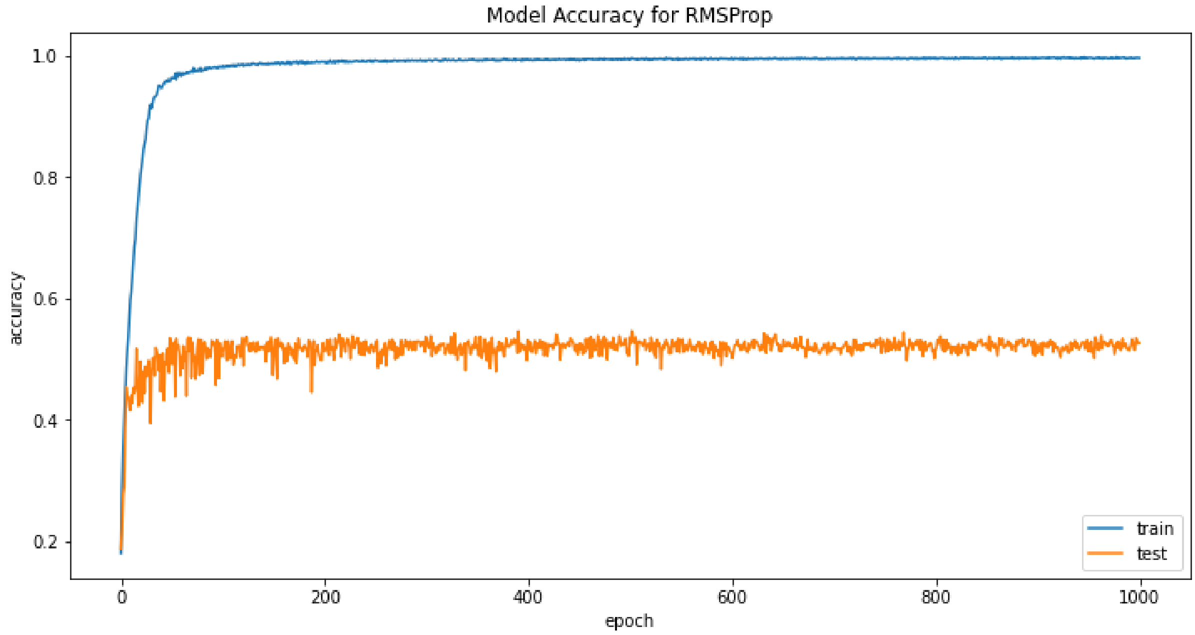


Figure 10. Training and test accuracies for RMS prop optimizer model

The training loss and training accuracy converges very early at about 30-50 epochs into training process. The testing accuracy also converges at about this number of epochs, and the best testing accuracy recorded is **0.54650**. **The testing loss, however, cannot converges** as it achieves it minimum very early at 30-50 epochs and then continuously increase with more epoch. The overfitting between train and test metrics are very big in this case, comparing to the SGD curve in 4.1.1

c. Adam optimizer

The training and test costs vs epoch plot for Adam optimizer is shown in *figure 11*.

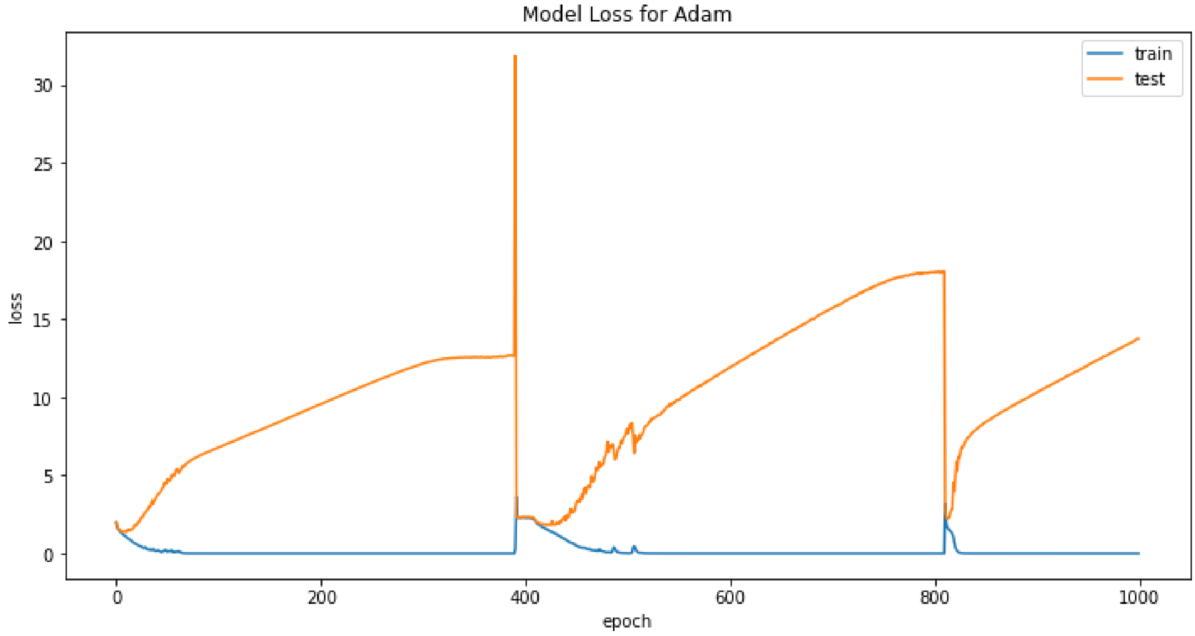


Figure 11. Train and Test cost vs training Epochs for Adam optimizer

The training and test accuracies vs training epoch for Adam optimizer is shown in *figure 12.*

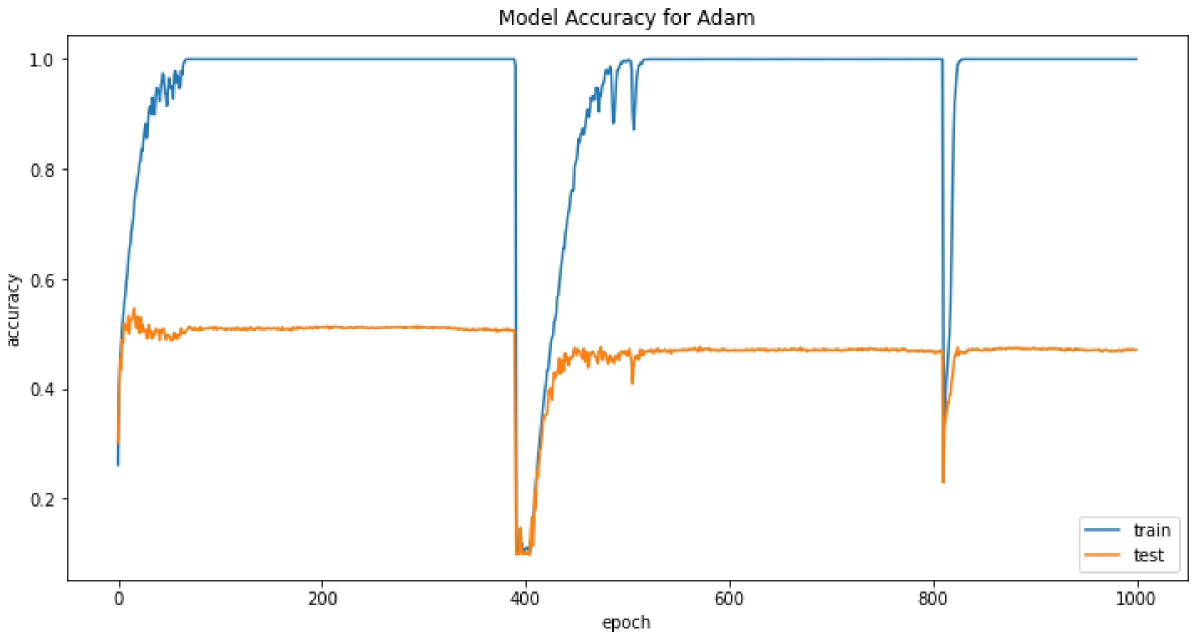


Figure 12. Train and test accuracies vs epoch for Adam optimizer model

A general trend. we can see in both *figure 11 and 12* is that the train/test loss and train/test accuracy tend to be unstable, with a lot of drastic jumps up and down. The test loss and accuracy can be achieved quite early on in training, about 30-50 epochs just like in RMSProp. The best test accuracy for Adam model is **0.54650**, same with RMSProp. However, as training proceeds on, the training accuracy become worse, with total vanish around 400 epochs and big drop at about 800 epochs (*figure 12*). The test cost on the other hand explodes and then drop in a repetitive manner throughout training (*figure 11*). Like RMS prop, there are serious overfitting in this model during training for long epochs too.

d. Dropout of 0.5 to the 2 fully connected layers.

We make use of SGD optimizer, with no momentum for this experiment. The only difference between the experiment and that in a and question 1 is that we add on dropout with 0.5 rate before the fully connected layers. The graph of training and testing loss for this model vs training epochs is shown in *Figure 13* below.

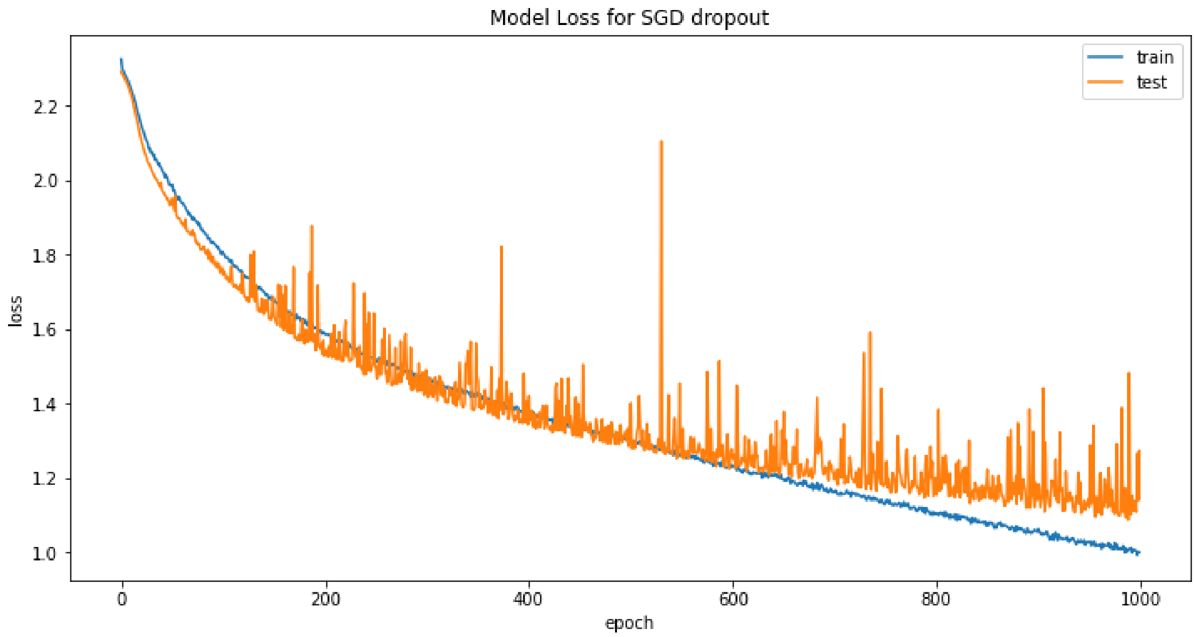


Figure 13. Training and testing costs vs Epoch for SGD dropout

The plot of training and testing accuracies vs the training epochs are also shown in *Figure 14*.

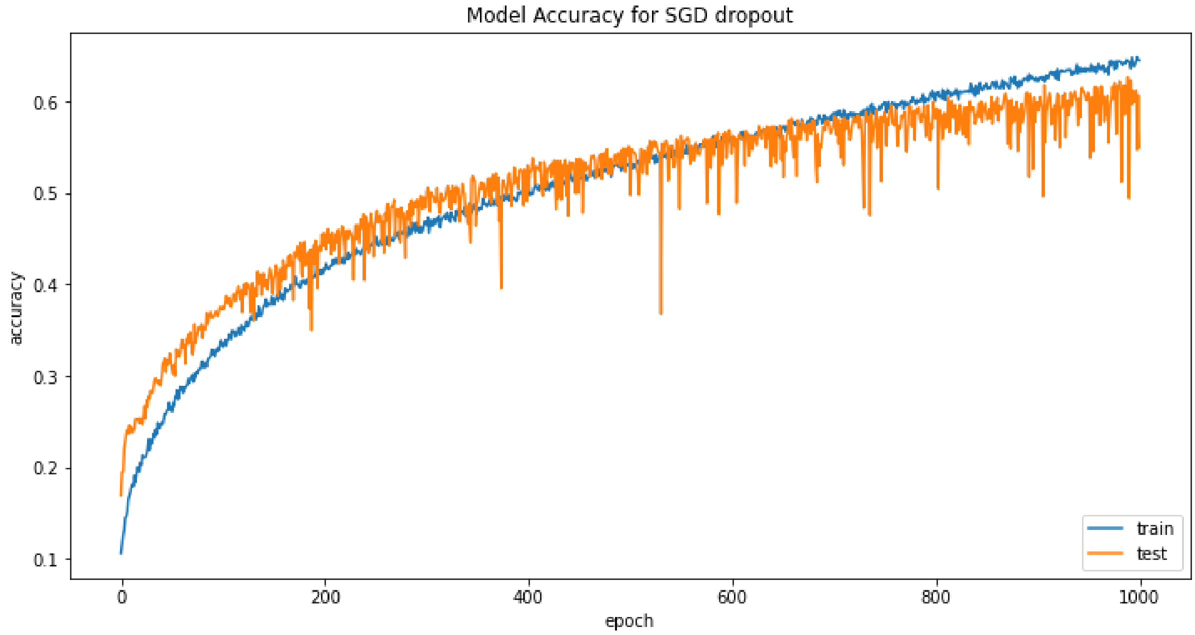


Figure 14. Training and testing costs vs Epoch for SGD dropout

We can observe that the best testing accuracy we have in this experiment is **0.62650**. The loss and accuracy only overfit at about 800 epochs. The testing loss and accuracy have not yet converged in 1000 epochs, but judging from the plot, they will surely converge with more training epochs, and at better values than reported in SGD with momentum 0.1 and vanilla SGD optimizer. *Also, the degree of overfitting in this model is very small comparing to other models in this question. This is expected as dropout is often used to reduce overfitting of model since it forces the model to learn all the features in the images and enhance the generalization ability of the model in predicting the results*.

e. Overall results

The plot of all losses together and the plot of all accuracies together for these experiments are shown in *figure 15 and 16* respectively

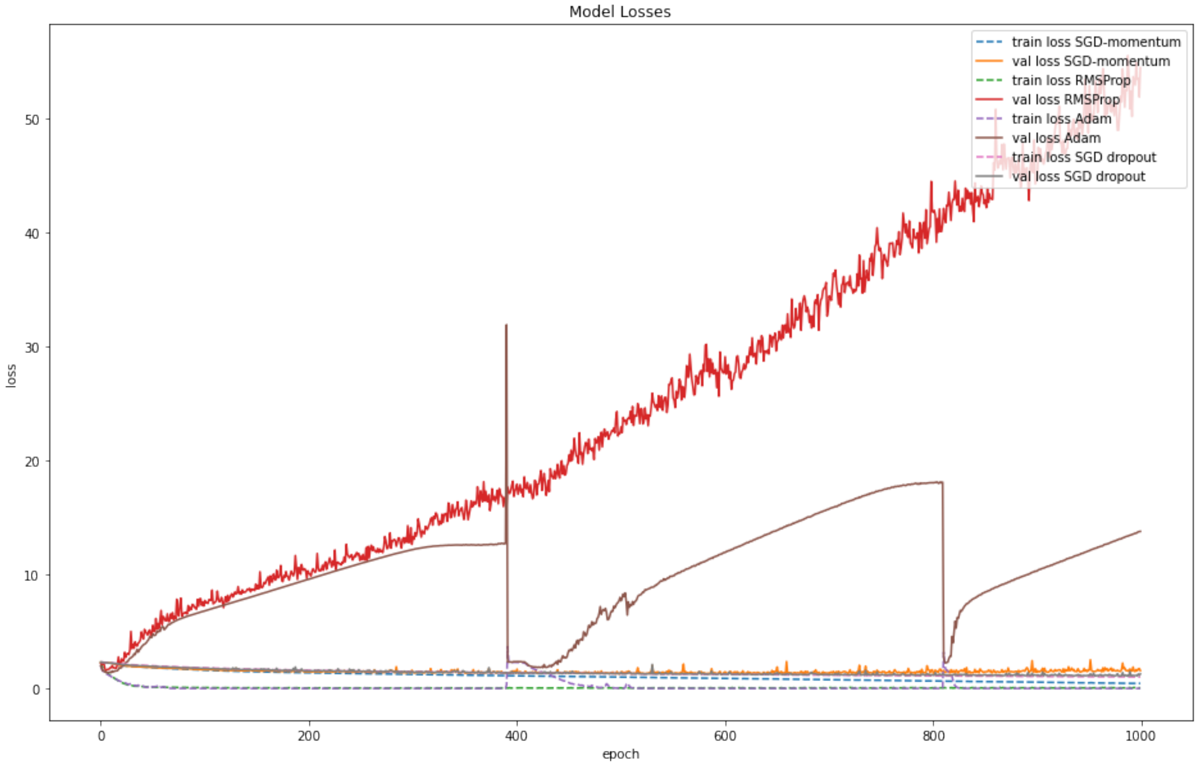


Figure 15. Plot of losses vs training epochs

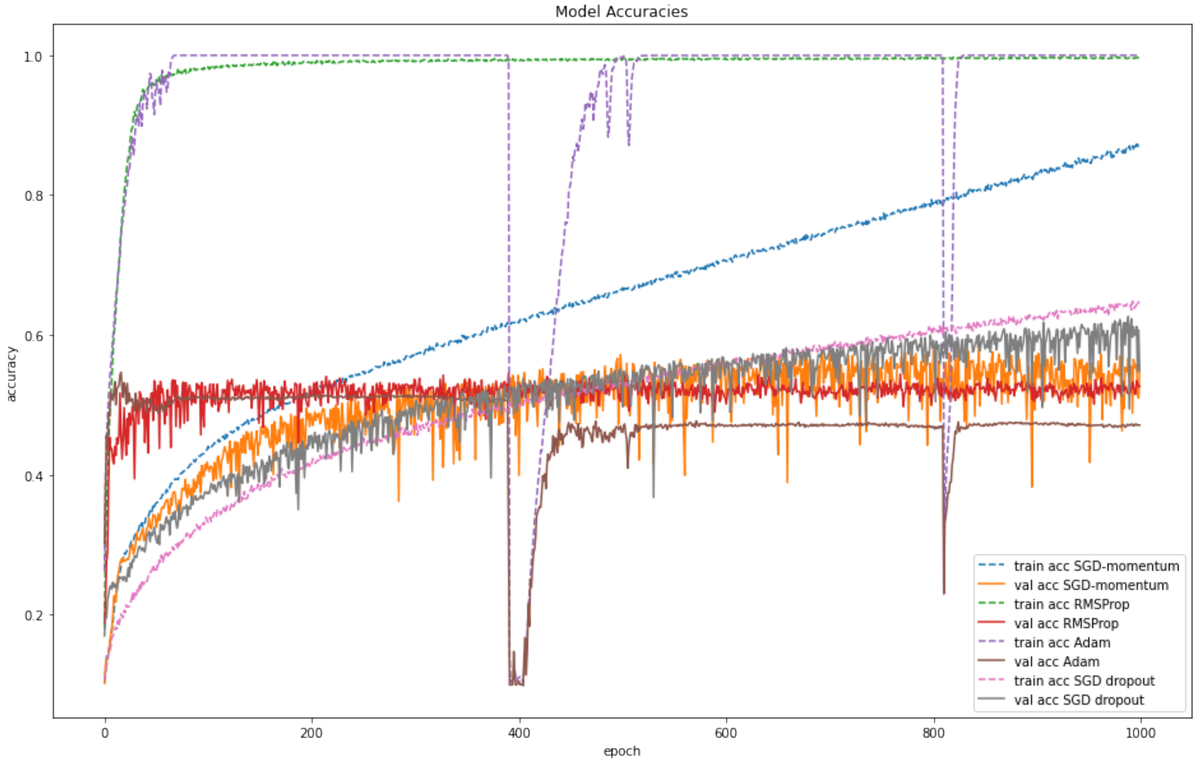


Figure 16. Plot of accuracies vs training epochs

From our observation in a, b, c, d, and *figure 15* and *figure 16*. We can see **that SGD with no momentum and added dropout layer of rate 0.5 gives the best performance in test accuracy**. The grey solid curve in the loss plot (*figure 15*) is the lowest (amongst all test loss) at convergence after 1000 epoch while the grey solid curve in the accuracy plot (*figure 16*) is the highest (amongst all test accuracy) at 1000 epoch. Therefore, SGD with dropout is the best performing model in the four experiments. Also, it yields very small overfitted model, shown from the distance between the grey curve (test) and the pink dashed curve (train) in both *figure 15 and 16*.

Another observation is that **Adam and RMS Prop models can reach their min/max values for loss and accuracy very early on in the training process**. However, they are **not able to reach convergence** with many instabilities jumps and drops in their curves. This can be shown when we observe the brown and red solid curves in *figure 15 and 16* above. Another problem with these models **is huge overfitting margin between train and test curve**.

The **SGD optimizer can give us better test accuracy (than rmsprop and adam) and allow for stable convergence**, despite the longer training epochs needed to get convergence. Their overfitting is also much smaller. **With dropout added, the time taken to reach convergence gets even longer for SGD, however a much better test accuracy can be reached** comparing to momentum SGD, vanilla SGD and therefore RMSProp and Adam. A balance between time for training and optimal test performance need to be consider here.

The plot of best testing accuracy for each model is shown in *figure 17*, which agrees to the observation that SGD dropout, no momentum has the best performance as its test accuracy is highest amongst the 4 models. We shall be using SGD dropout model as the best model for question 3.

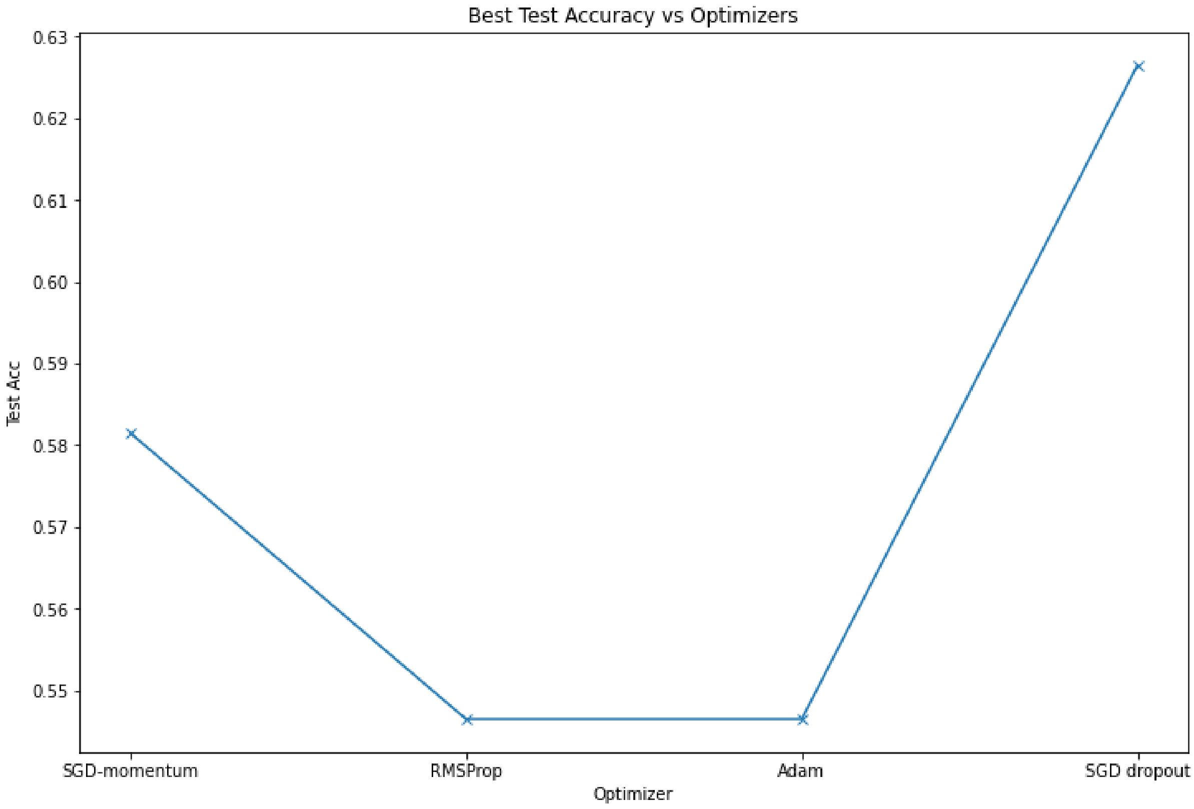


Figure 17. Best test accuracy vs optimizer model

4.1.4. Question 4

Comparing performances based on test accuracy.

The best test accuracy in **question 1** (number of filters = 50, 60; no dropout), **question 2** (number of filters = 70, 100; no dropout) and **question 3** (number of filters = 70, 100; dropout = 50%) are **0.58900, 0.60100 and 0.62650 respectively**. The model in question 2 with optimal number of filters is better than the baseline model, while the model in question 3, with dropouts is better than model 2 based on test accuracy.

The **improvement** between question and question 2 optimal model will be accounted by the **increased in number of filters for feature maps in each of the convolution layer** (from 50 -> 70 and from 60 -> 100). This is because more filters will lead to more features being extracted at each layer, helping the model to learn the images better for prediction, hence increasing it accuracy.

On the other hand, the **improvement** in between question 2 and question 3 optimal model is due to the **dropout regularization** as some of the output from the convolution and max pool layers are dropped when entering the fully connected layer, forcing our classifier to learn all the features extracted, and give rise to better generalization ability. This therefore reduce the overfitting in this question 3 optimal model and increase the testing accuracy of the model.

So, in conclusion, optimal model of question 3 is the most optimal model for our experiments to benchmark against CIFAR-10. This is the result of getting optimal number of filters for the hidden convolution layers and applying dropout regularization to the fully connected layer of the network.

4.2. Part B

4.2.1. Question 1

The training entropy cost/loss and the testing accuracy of the character-level CNN classifier vs the training epochs are shown in *figure 18* and *19*, respectively.

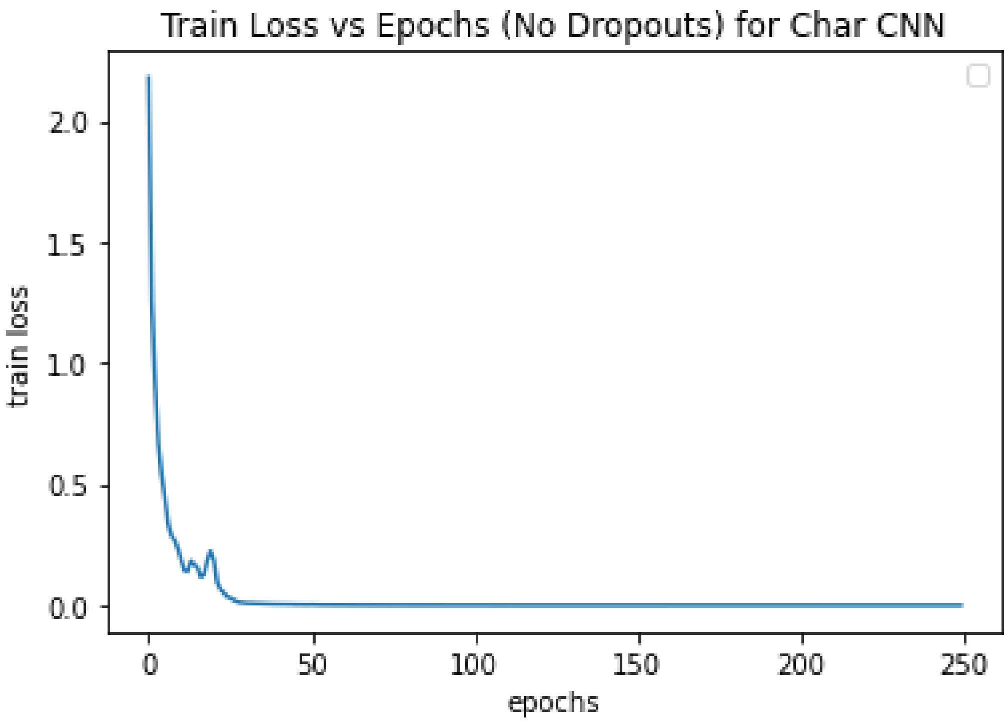


Figure 18. Train loss vs epoch for Char CNN

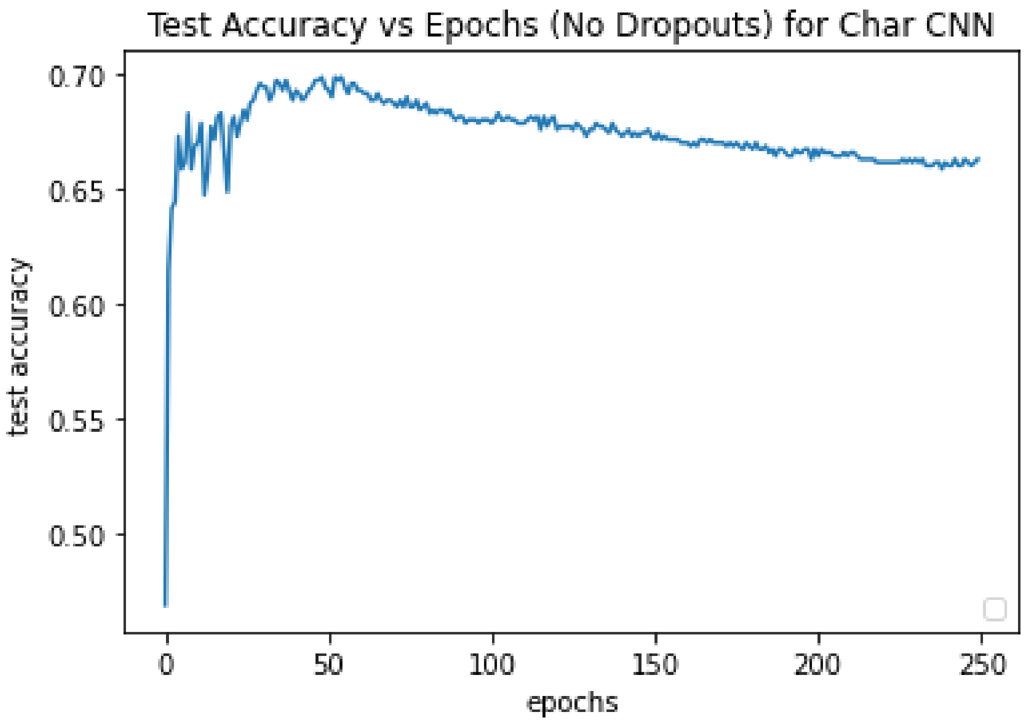


Figure 19. Test accuracy vs epoch for Char CNN

Best test accuracy obtained for this model is **0.69857**.

4.2.2. Question 2

The training entropy cost/loss and the testing accuracy of the word-level CNN classifier vs the training epochs are shown in *figure 20 and 21*, respectively.

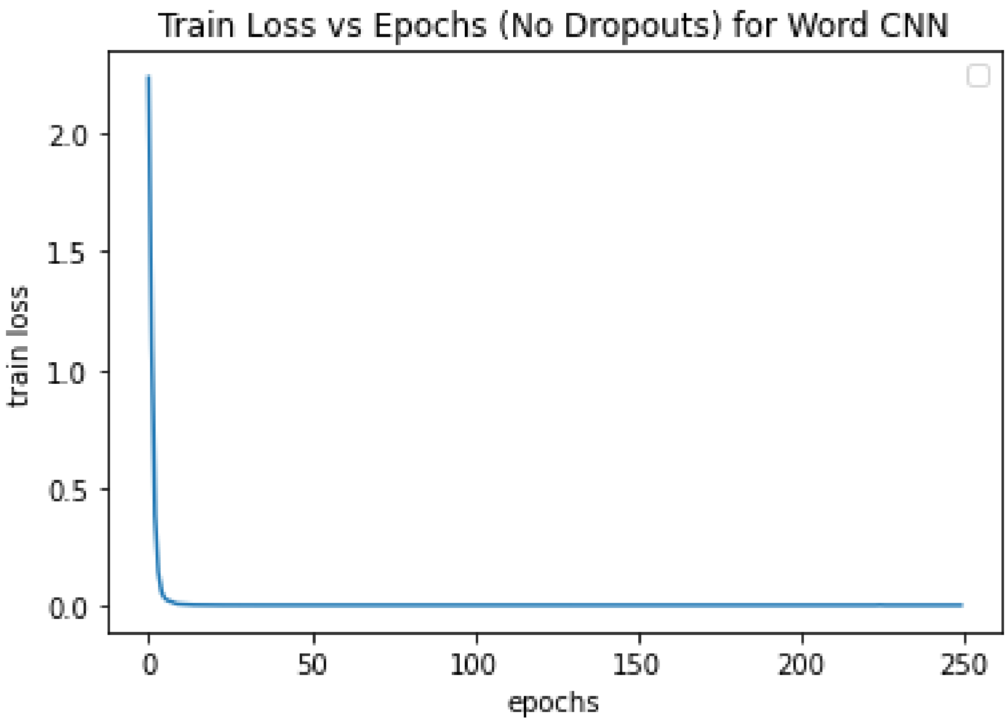


Figure 20. Train loss vs epochs for Word CNN

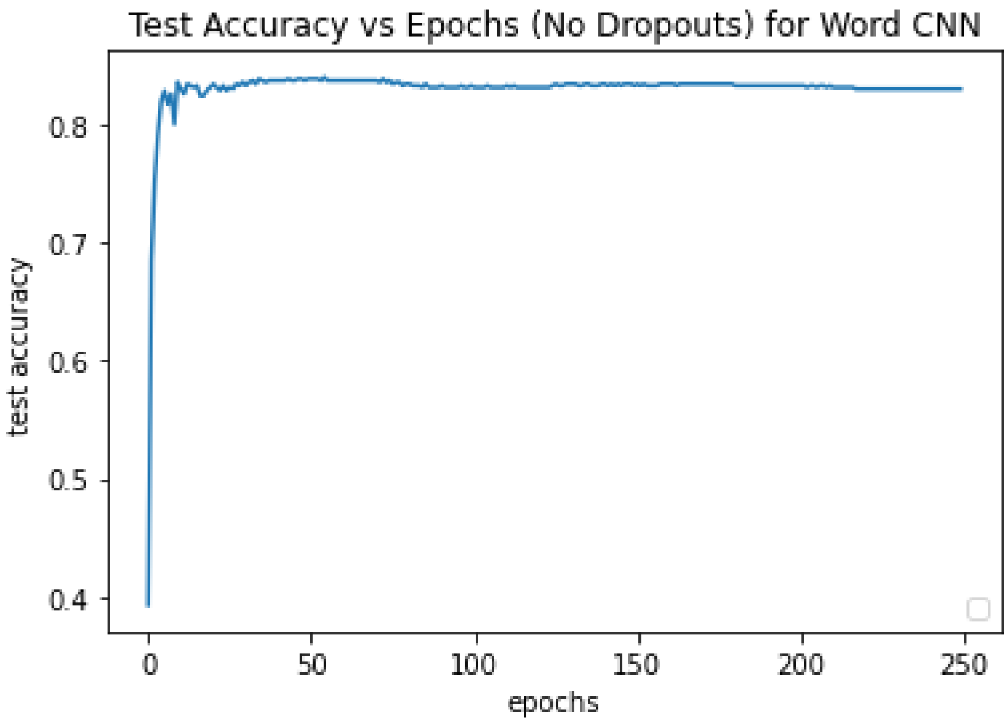


Figure 21. Test accuracy vs epochs for Word CNN

The best test accuracy for this model is **0.84**

4.2.3. Question 3

The training entropy cost/loss and the testing accuracy of the character-level RNN classifier vs the training epochs are shown in *figure 22 and 23*, respectively.

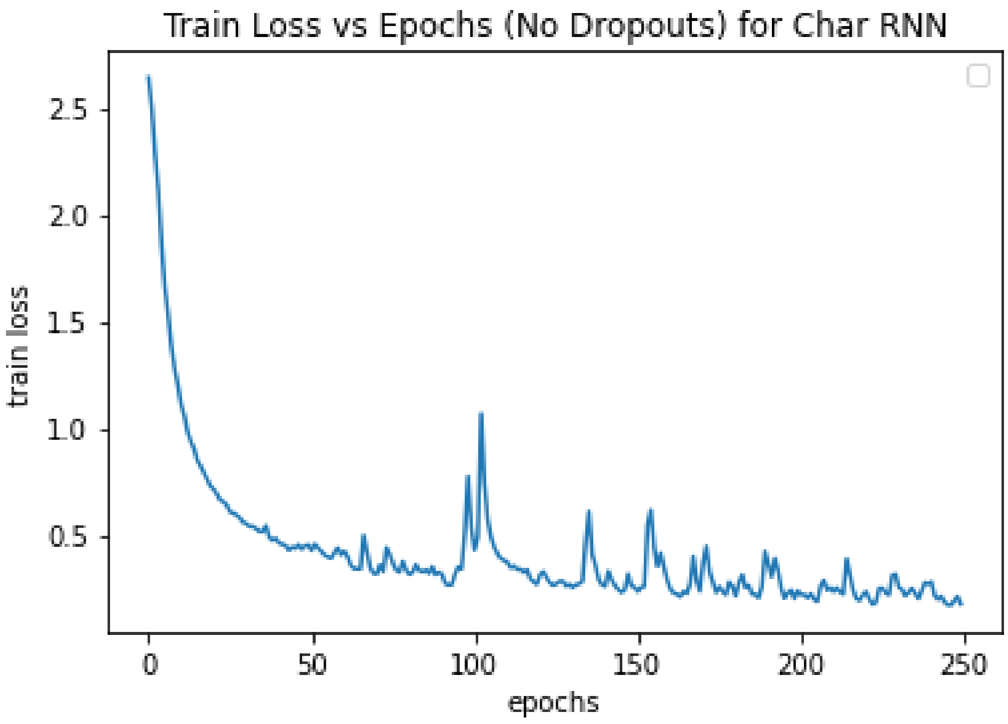


Figure 22. Train loss vs epochs for Char RNN

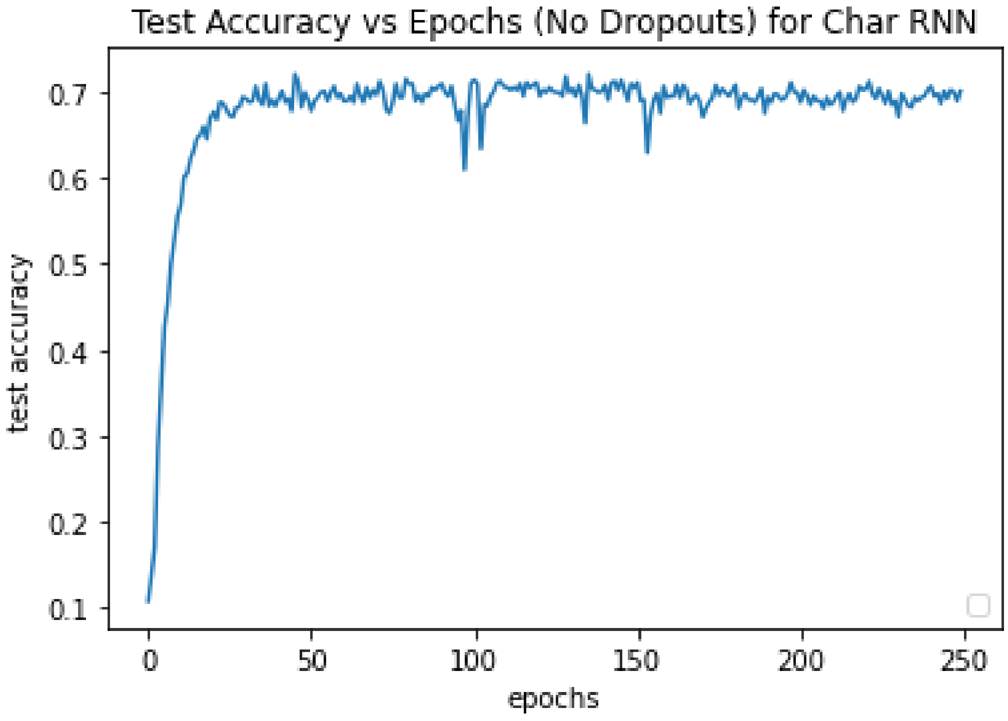


Figure 23. Test accuracy vs epochs for Char RNN

The best test accuracy for this model is **0.72143**

4.2.4. Question 4

The training entropy cost/loss and the testing accuracy of the word-level RNN classifier vs the training epochs are shown in *figure 24 and 25*, respectively.

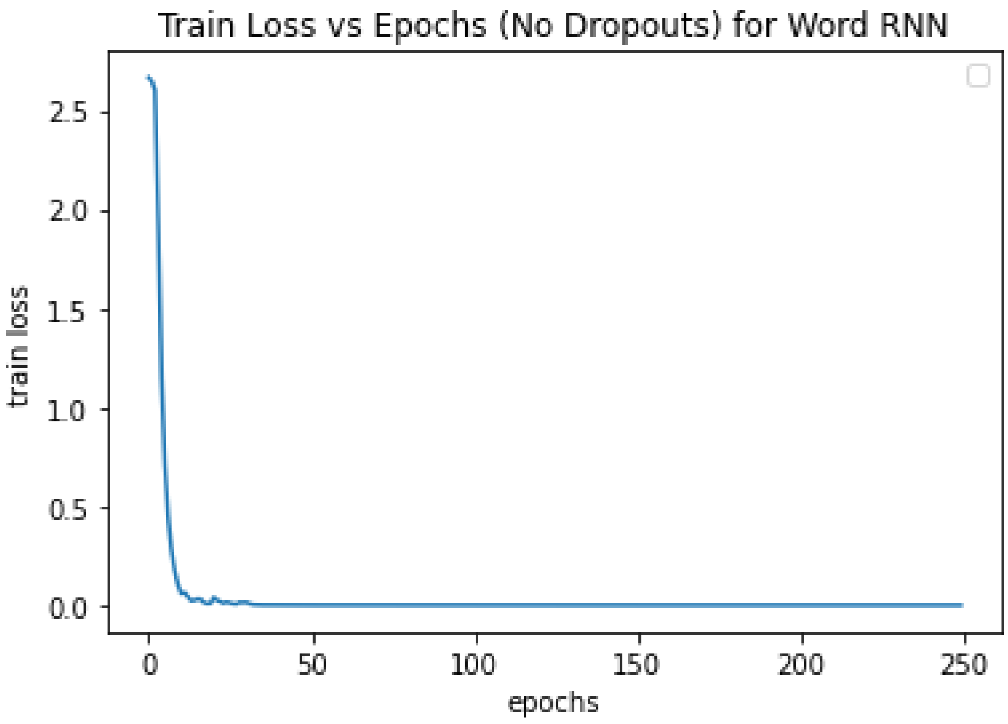


Figure 24. Train loss vs epochs for word rnn

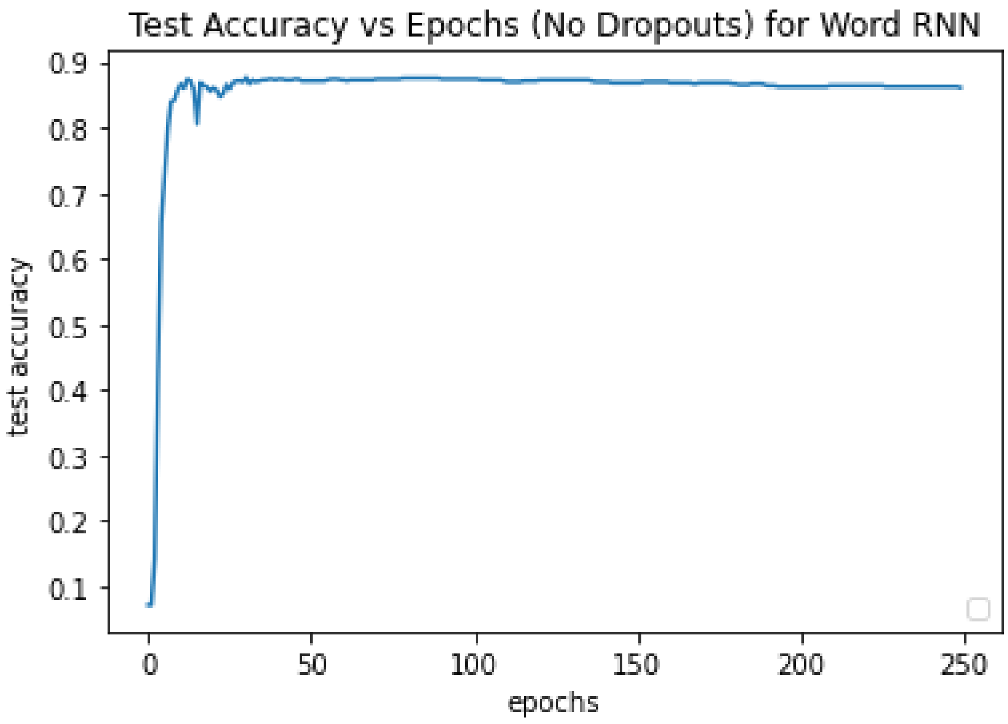


Figure 25. Test accuracy vs epochs for word rnn

Best test accuracy for this model is **0.87714**.

4.2.5. Question 5

Compare test accuracies and training time of the question 1 – 4 models.

The character-level CNN and RNN test accuracies are 0.69857 and 0.72143 respectively (question 1B and 3B). The word-level CNN vs RNN test accuracies are 0.84000 and 0.87714, respectively (2B and 4B).

We can see that the **RNNs models with GRU cells have higher test accuracies than the corresponding CNNs models for both word-level and character-level modelling**. This shows that RNNs with their persistent unit in their cells has better capability to model and predict categories of sequence of text comparing to CNNs. (0.87714 > 0.84 for Word RNN to CNN; 0.72143 > 0.69857 for Char RNN to CNN)

Also, **language modelling at word-level gives higher test accuracies comparing to language modelling at character-level** as shown in the paragraphs above. (Word RNN > Word CNN > Char RNN > Char CNN)

The **training time for Word CNN is twice longer that of Character CNN**. The training time for RNN models are very close to each other for level of language modelling. **Both are about 7 times the training time taken for Word CNN** (*table 2*). This is expected as RNNs require many forward and backward pass in multiple directions to train their weights, leading to very long training time. If training time is a concern, using CNNs might give a good acceptable performance in test accuracy with very short training time (*table 2).* **However, the performance of RNN, which requires longer training time is superior to that of CNN.**

Table 2. Test accuracies and training time for the models in question 1-4

Model	Test accuracy	Training time (in seconds)
Character CNN	0.69857	153
Word CNN	0.84000	332
Character RNN	0.72143	2330
Word RNN	0.87714	2238

Compare test accuracies of the dropout models with non-dropout models

The train loss and test accuracy vs training epochs for models in question 1-4 **with dropouts** are shown in *figure 26-33* below.

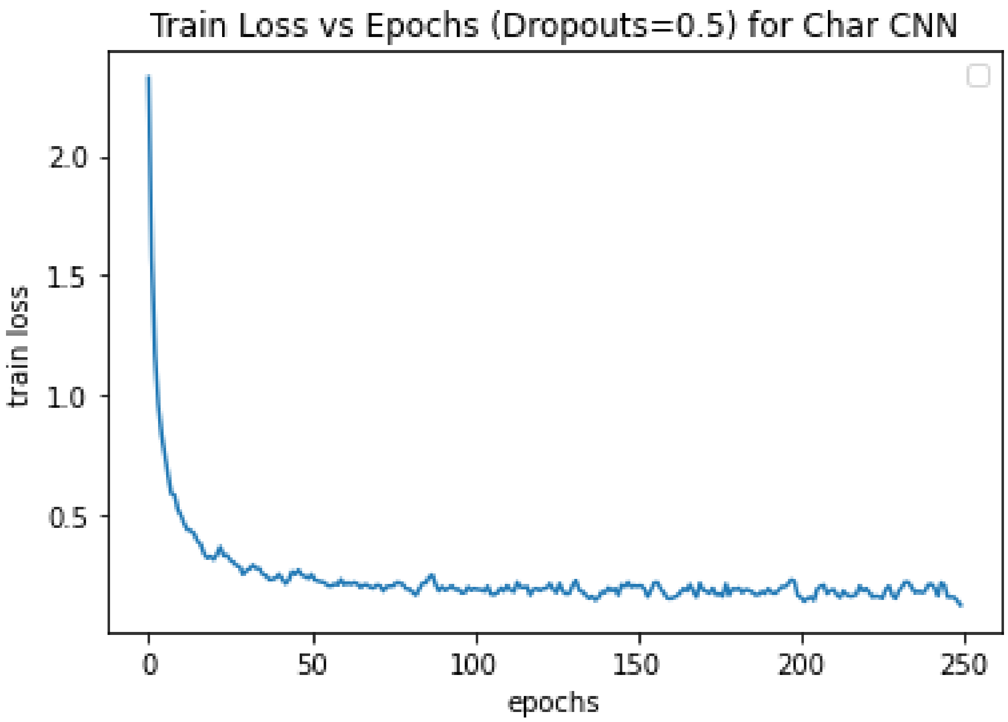


Figure 26. Train loss vs epochs for Char CNN with dropout

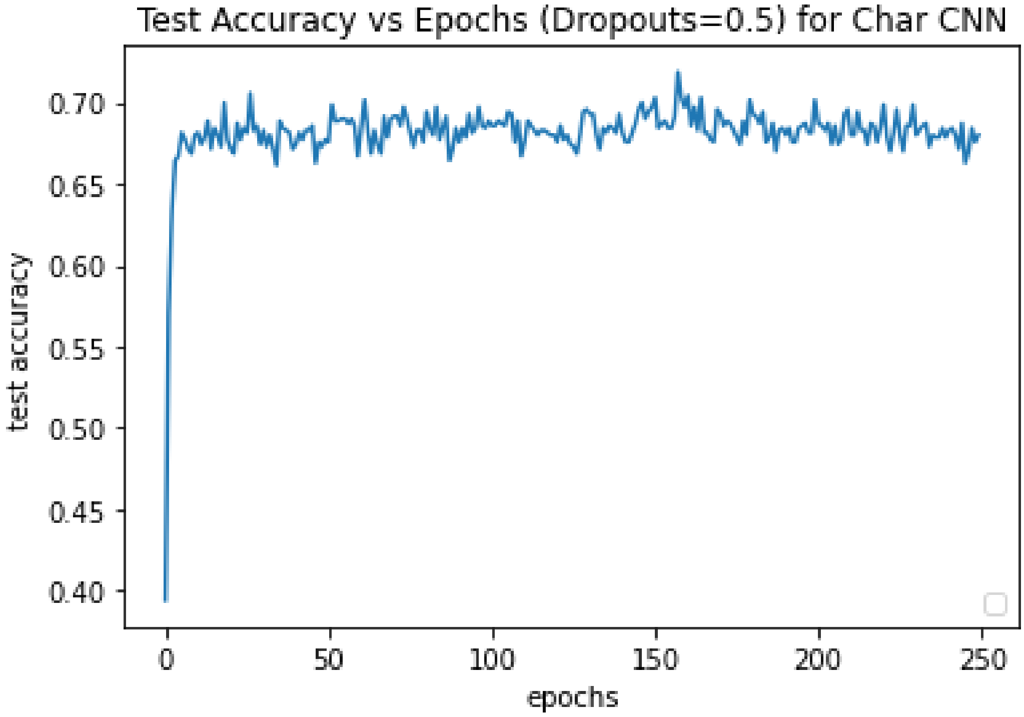


Figure 27. Test accuracy vs epochs for char cnn with dropout

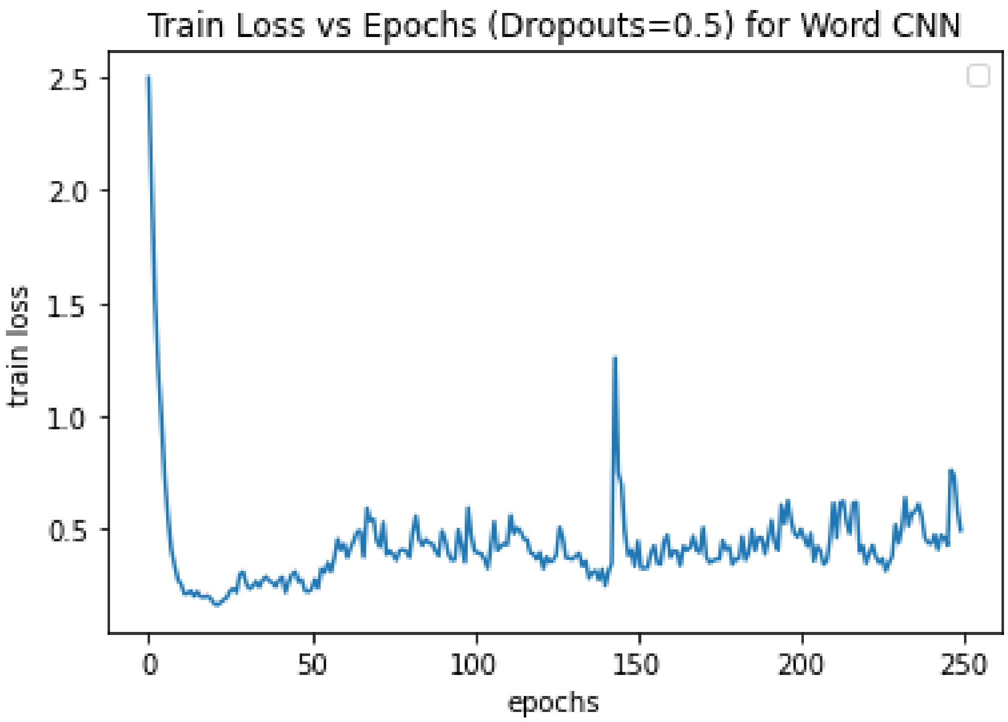


Figure 28. Train loss vs epochs for word cnn with dropout

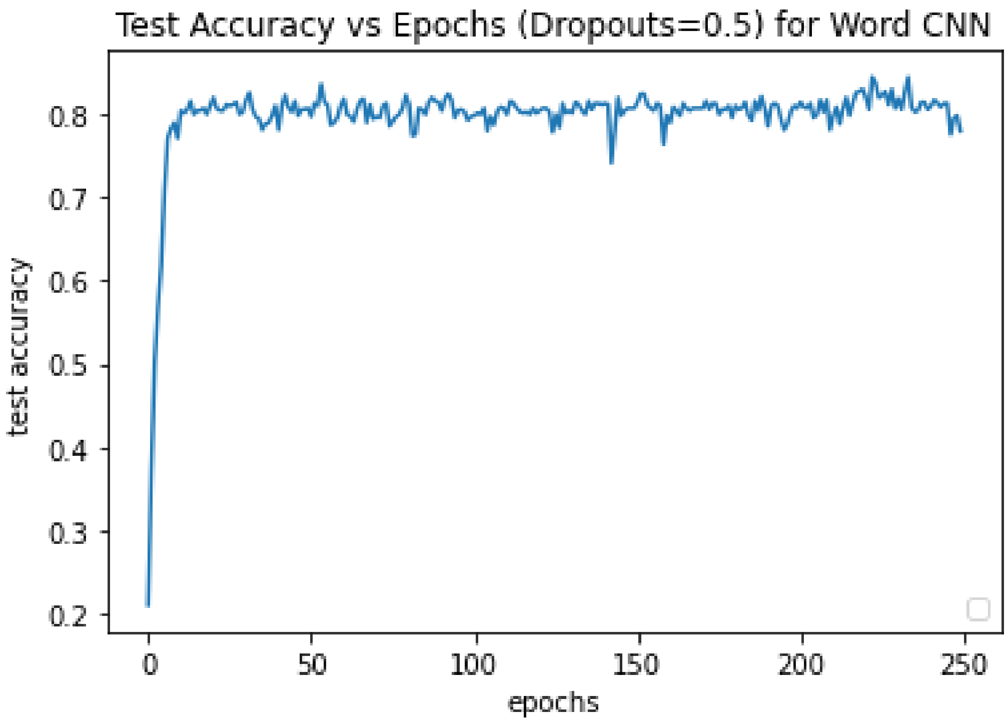


Figure 29. Test accuracy vs epoch for word cnn with dropout

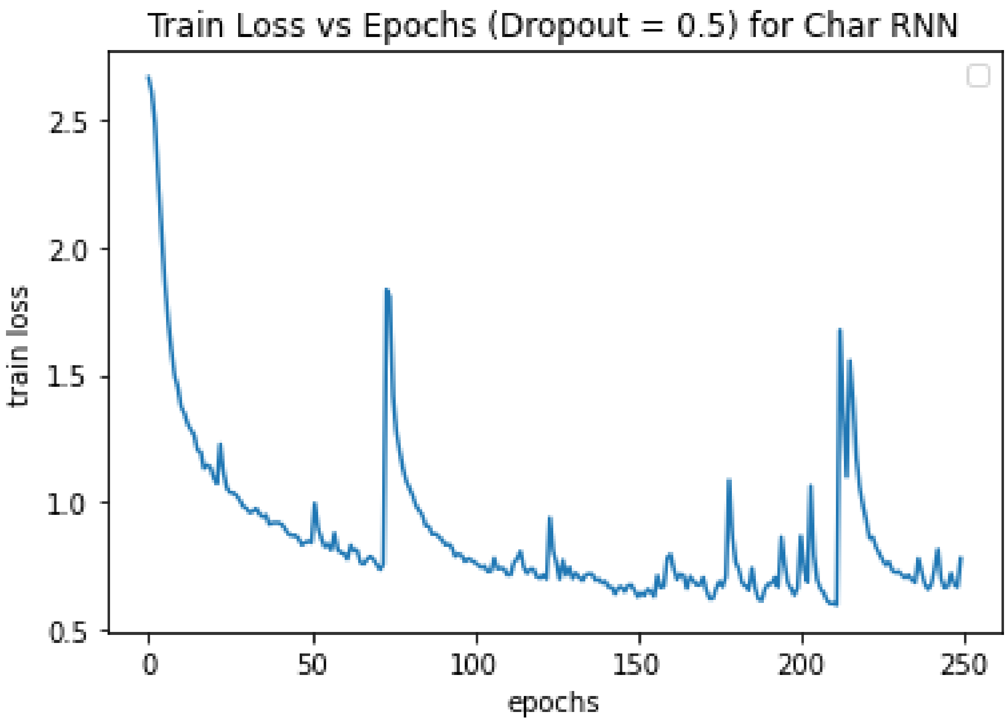


Figure 30. Train loss vs epochs for Char RNN with dropout

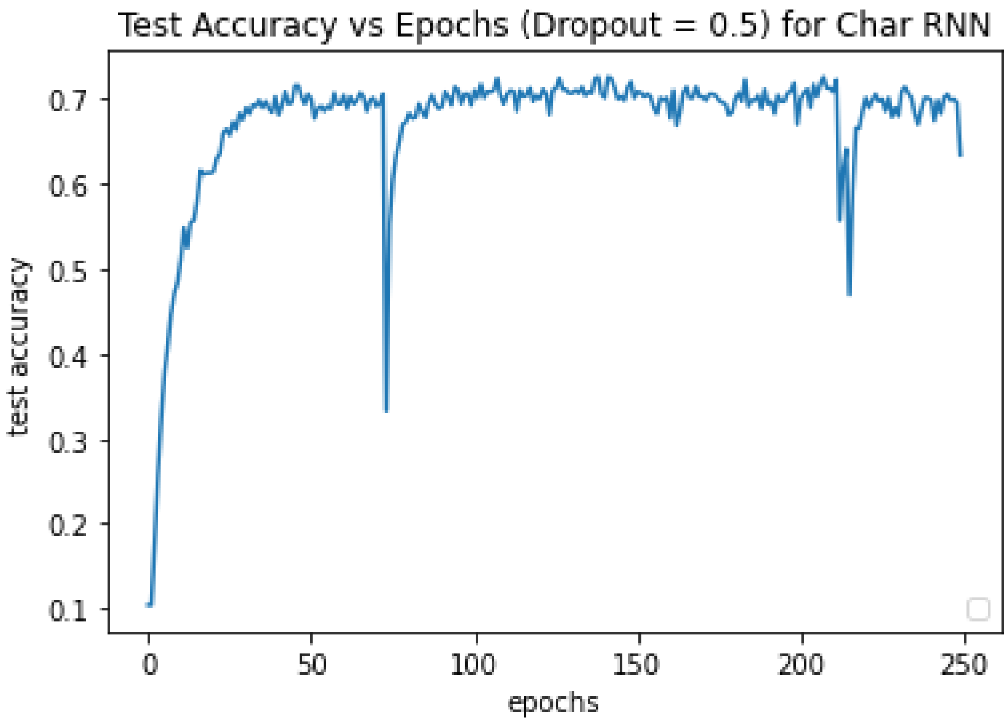


Figure 31. Test accuracy vs epochs for Char RNN with dropout

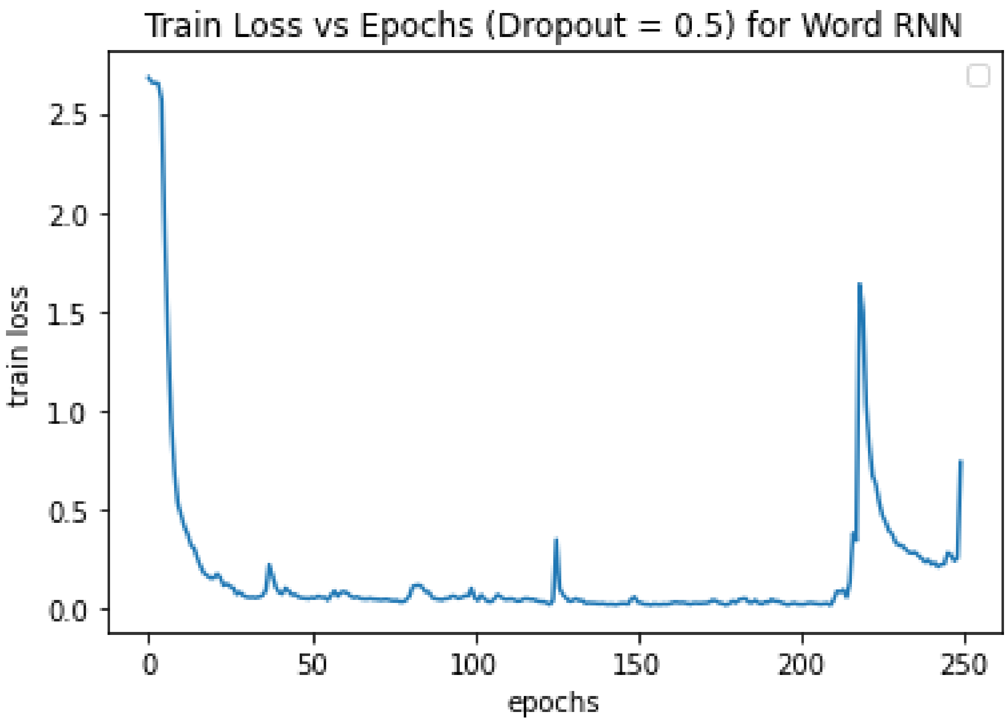


Figure 32. Training loss vs epochs for Word RNN with dropout

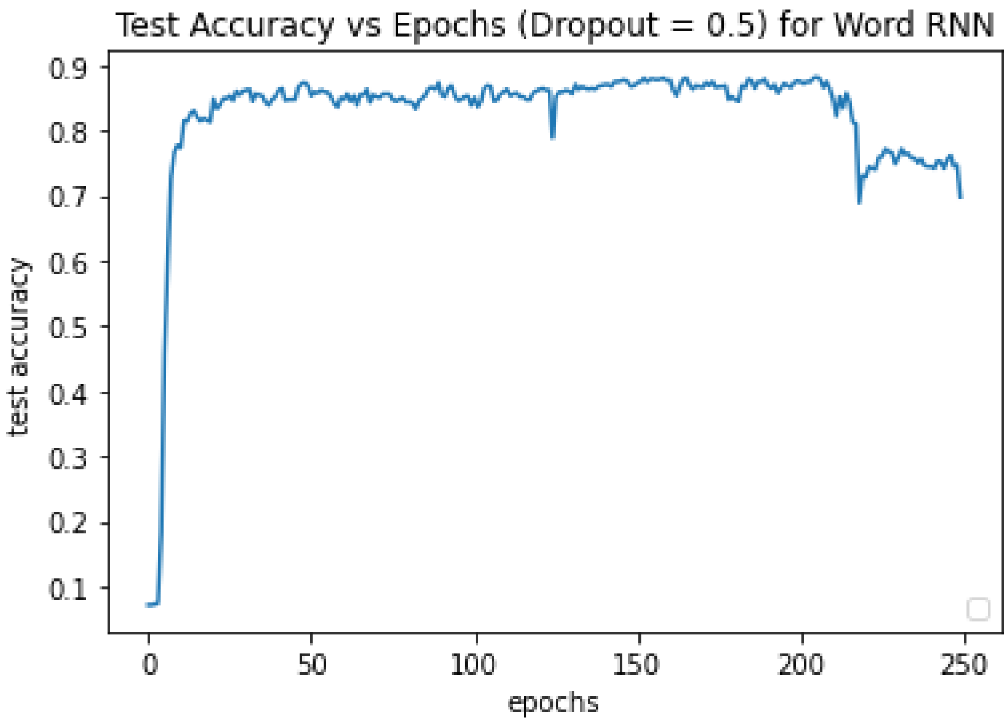


Figure 33. Test accuracy vs epochs for word rnn with dropouts

The consolidated test accuracies of models in question 1-4 with and without dropouts are shown in *table 3*. **Clearly, by adding dropout to the models, we can achieve higher test accuracy values and hence better performances**. This is **observable for both word and character level with both CNN and RNN modes** and can be explained by dropout forcing our model to learn all features hence generalize the model more.

Table 3. Test accuracies with and without dropout

Model	Without Dropout	With Dropout
Char CNN	0.69857	0.71857
Word CNN	0.84000	0.84286
Char RNN	0.72143	0.72571
Word RNN	0.87714	0.88143

4.2.6. Question 6

This question is assumed to be independent from question 5, meaning that we will not be using dropout in these models as question is asking for implementations in question 3 and 4 (RNNs without dropouts).

a. Experiments with GRU, Vanilla RNN and LSTM cells.

Character Model

The individual train loss and test accuracy for GRU, RNN and LSTM unstacked models for character level are shown in the notebook 6B-char-a.ipynb. The combined train loss and test accuracy vs epochs of these 3 models are shown in *figure 34 and 35*.

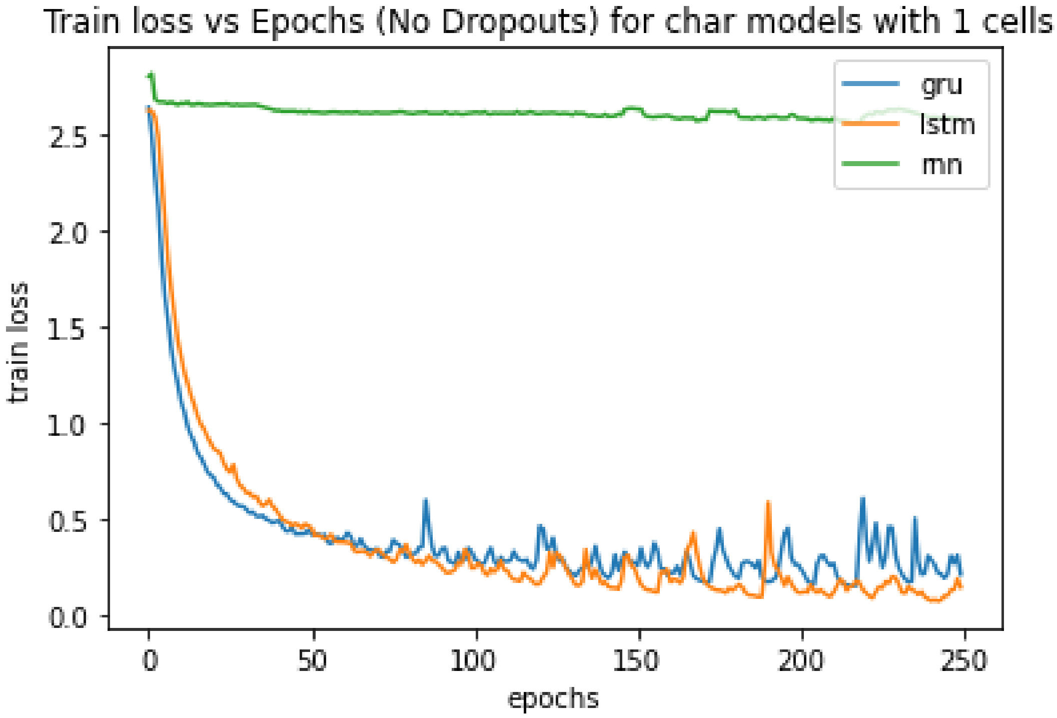


Figure 34. Train loss vs Epochs for unstacked char RNN models

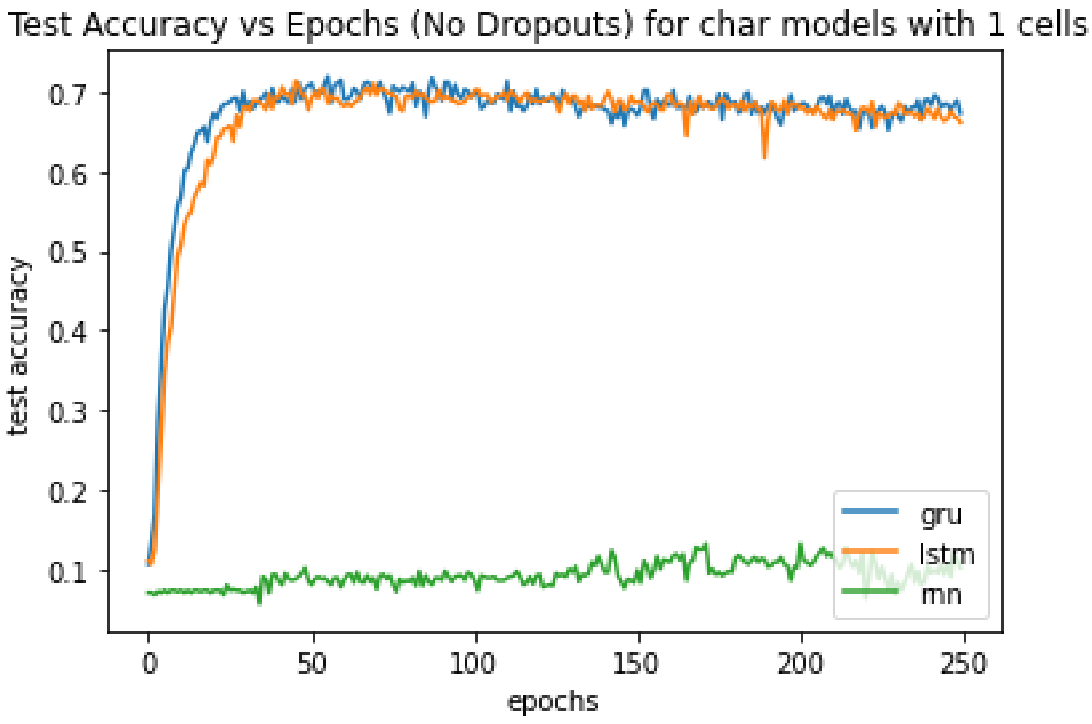


Figure 35. Test accuracy vs Epochs for unstacked char RNN models

Word Model

The individual train loss and test accuracy for GRU, RNN and LSTM unstacked models for word level is shown in the notebook 6B-word-a.ipynb. The combined train loss and test accuracy vs epochs of these 3 models are shown in *figure 36 and 37*

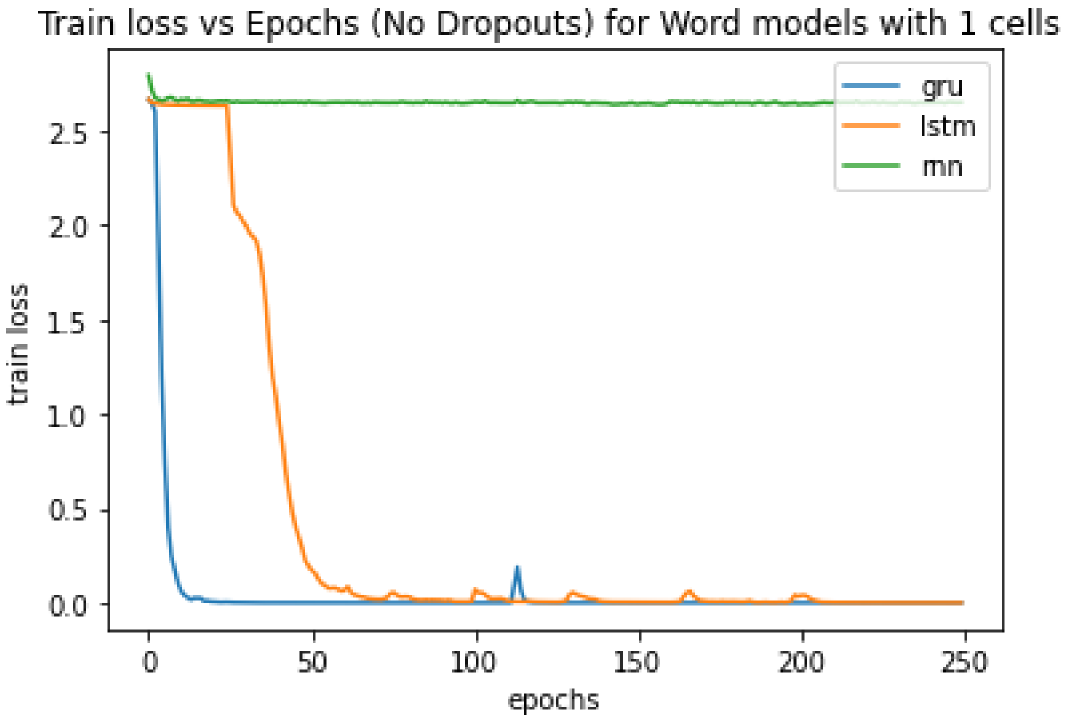


Figure 36. Train loss vs epochs for word, unstacked RNN

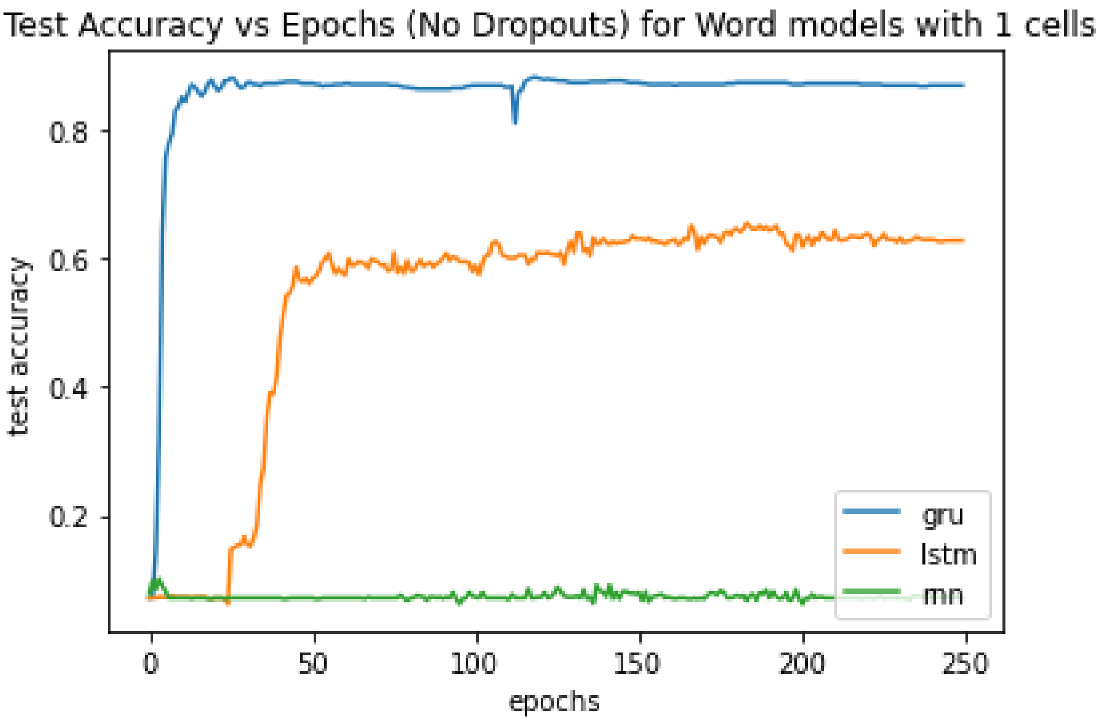


Figure 37. Test accuracy vs epochs for word, unstacked RNNs

Results

Test accuracies of these models are shown in Table 4. From table 4, *figure 36 and 37*, **GRU cell type has the best performance for both character level and word level RNNs**. It is then **succeeded by LSTM cell type in both cases**. Vanilla has very bad performance in both cases of character and word RNNs (about 0.1 accuracy). This is explained from its inability to learn well to long sequences with complicated contexts as it does not have the various gates to learn and forget like GRU and LSTM, which have better results here. Another observation is for character level modelling, LSTM and GRU has very close performance, while in word modelling, GRU cell has superior performance for the task.

Table 4. Test accuracies for different RNN cell types

Model type	Cell type	Test accuracy
Character RNN	GRU	0.71857
	LSTM	0.71286
	Vanilla RNN	0.13286
Word RNN	GRU	0.88143
	LSTM	0.65429
	Vanilla RNN	0.10143

b. Experiments with Stacked RNN cells.

Character Model

The individual train loss and test accuracy for 2-layer stacked GRU, RNN and LSTM models for character level are shown in the notebook 6B-char-b.ipynb. The combined train loss and test accuracy vs epochs of these 3 models are shown in *figure 38 and 39*.

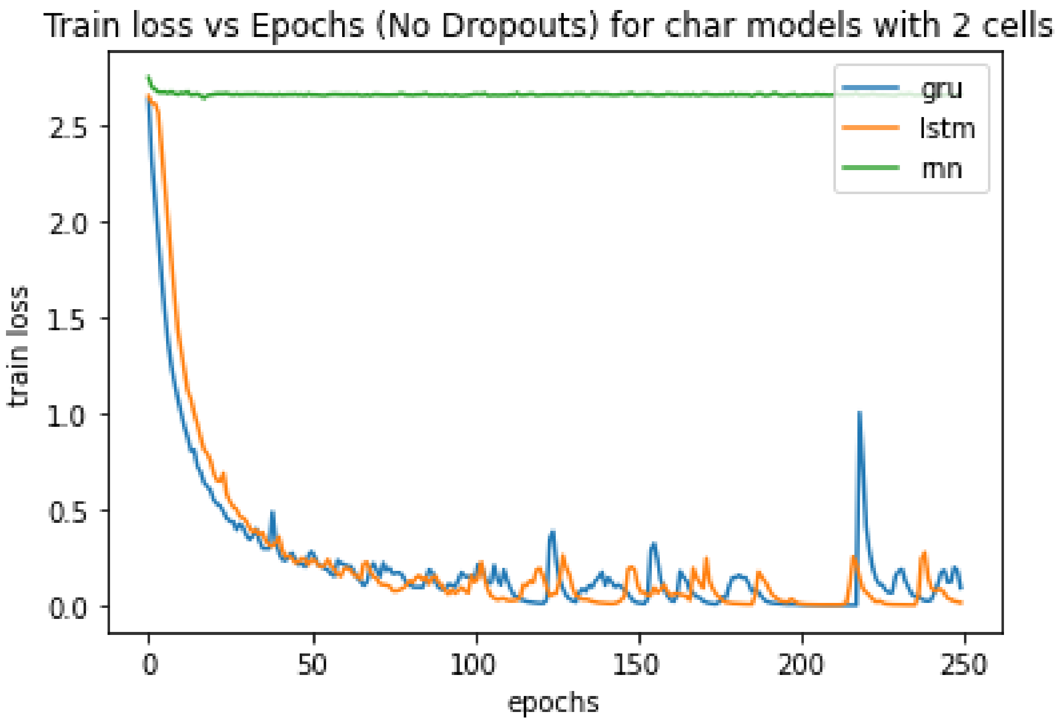


Figure 38. Train loss vs epoch for the Stacked RNNs

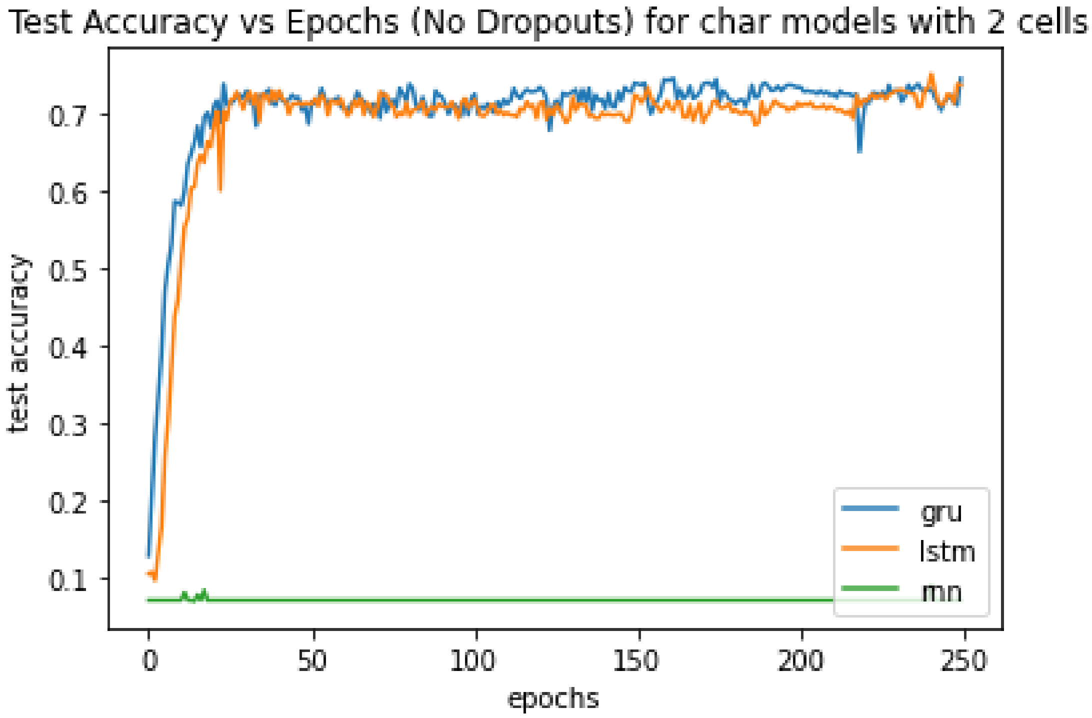


Figure 39. Test accuracy vs epoch for the Stacked RNNs

<u>Word Model</u>

The individual train loss and test accuracy for 2-layer stacked GRU, RNN and LSTM models for character level are shown in the notebook 6B-word-b.ipynb. The combined train loss and test accuracy vs epochs of these 3 models are shown in *figure 40 and 41*.

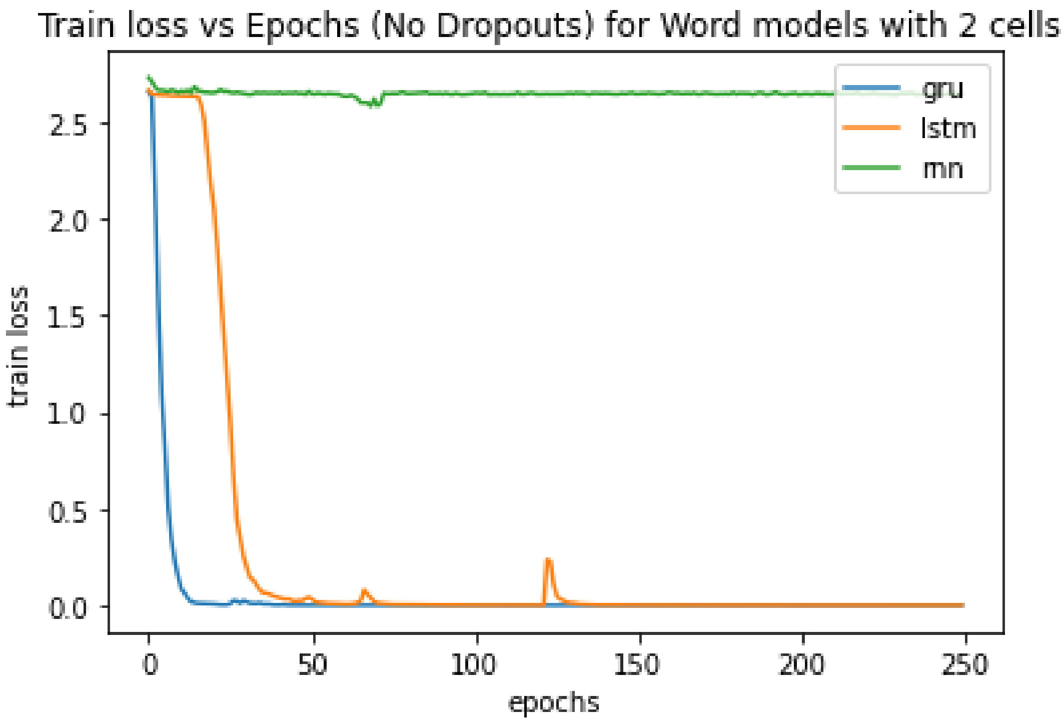


Figure 40. Train loss vs Epochs for Stacked RNN

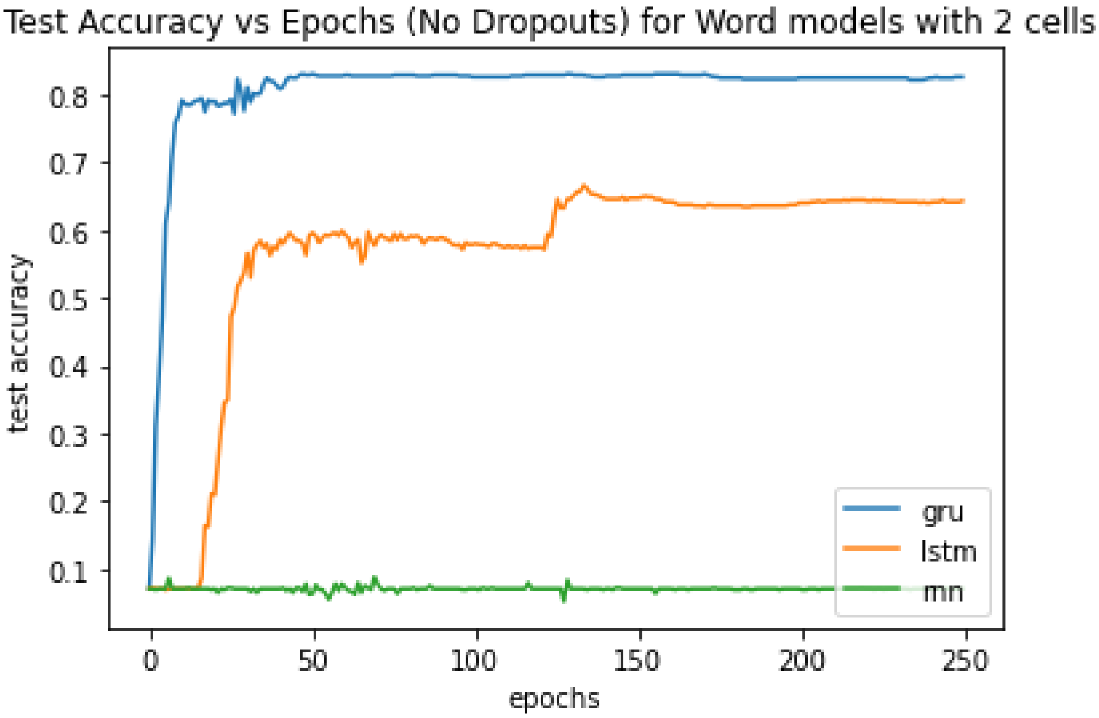


Figure 41. Test accuracy vs Epochs for Stacked RNNs

Results

Table 5. Test accuracies across Stacked RNNs for different language modelling level

Model Type	Cell Type	Test accuracy

Character	Stacked GRU	0.74429
	Stacked LSTM	0.75000
	Stacked Vanilla RNN	0.09143
Word	Stacked GRU	0.83000
	Stacked LSTM	0.66571
	Stacked Vanilla RNN	0.08857

For **character level modelling, the stacked LSTM has slightly higher test accuracy comparing to the stacked GRU (*table 5*)**. The Vanilla RNN still has very bad performance (less than 0.1 test accuracy). On the other hand, the trend for test accuracies of Word level modelling models are very similar to that for unstacked model (GRU > LSTM > Vanilla RNN test accuracies).

c. Experiments with Gradient clipping.

Unstacked

Character level: the train loss and test accuracy for character level, unstacked RNNs are shown in *figure 42 and 43* below.

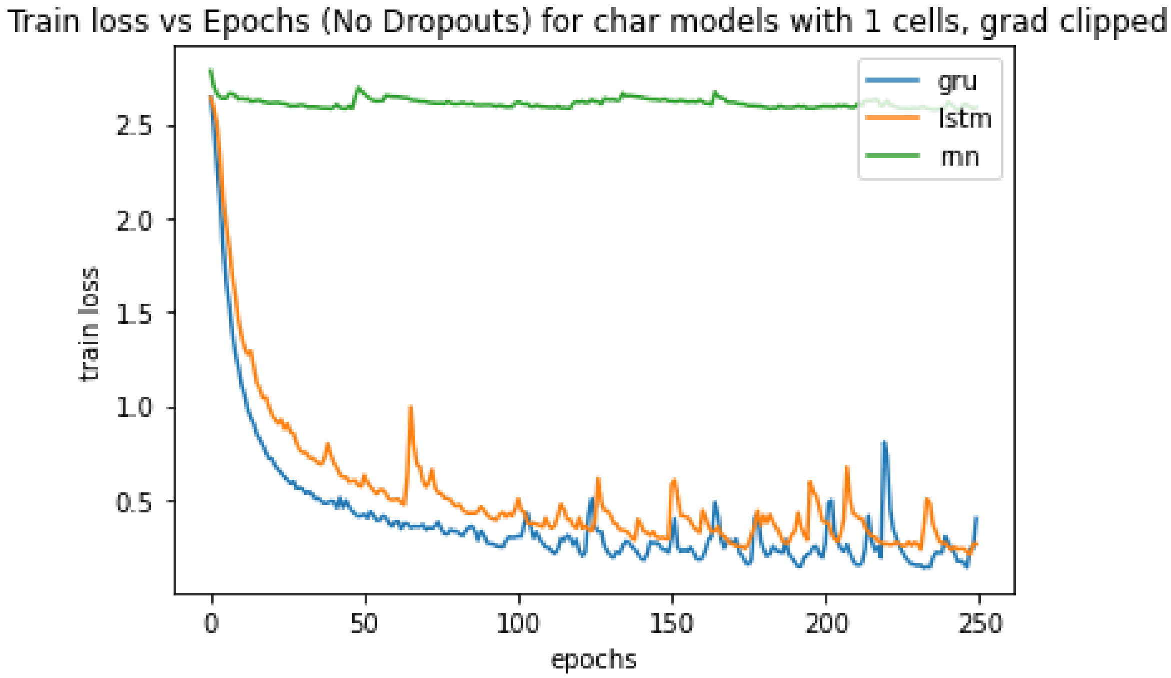


Figure 42. Train loss vs Epochs for grad clipped unstacked Char RNNs

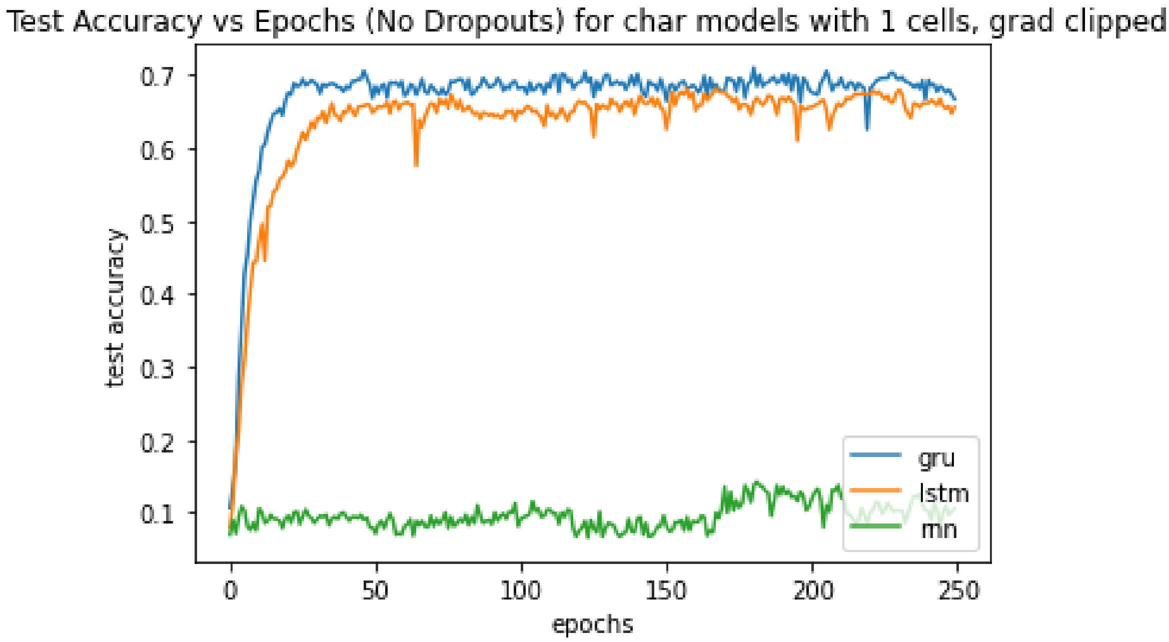


Figure 43. Test accuracy vs epochs for unstacked, char model with gradient clipped

Word level: the train loss and test accuracy for word level, unstacked RNNs are shown in *figure 44 and 45* below

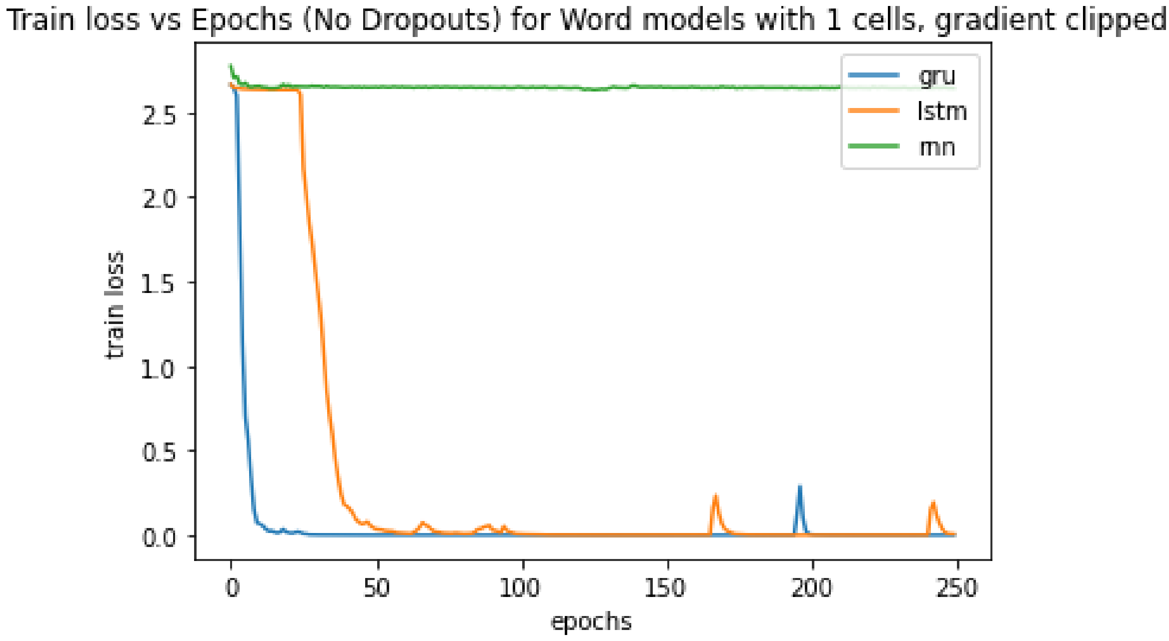


Figure 44. Train loss vs epoch for gradient clipped, unstacked word RNNs

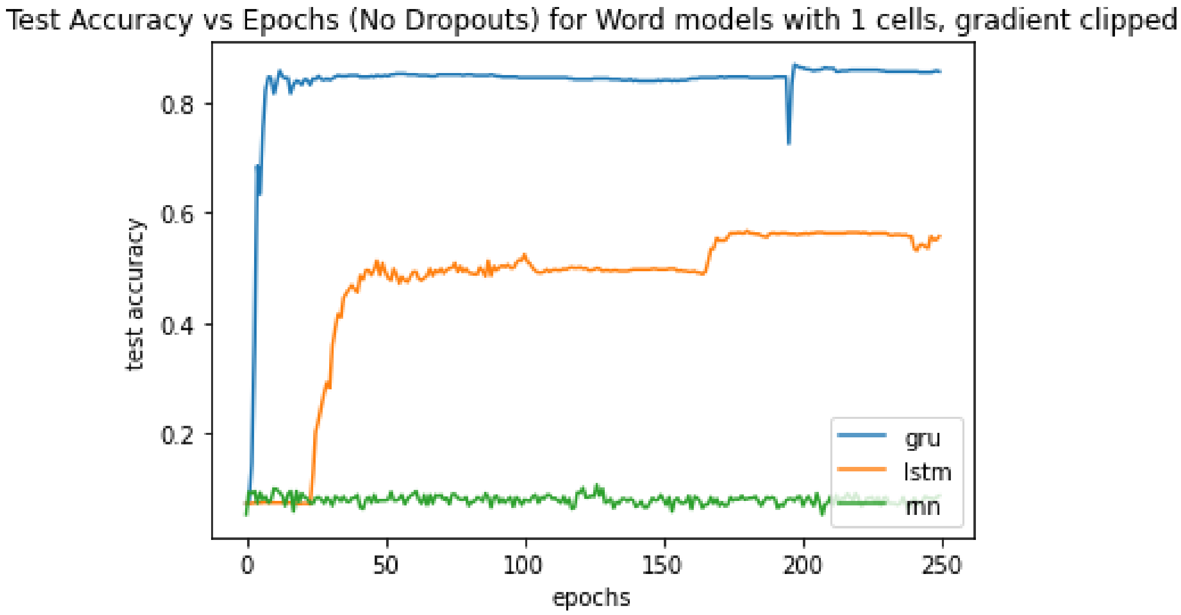


Figure 45. Test accuracy vs training epochs for gradient clipped, unstacked word RNN

Stacked

Character level: the train loss and test accuracy for character level, stacked RNNs are shown in figure *46 and 47* below.

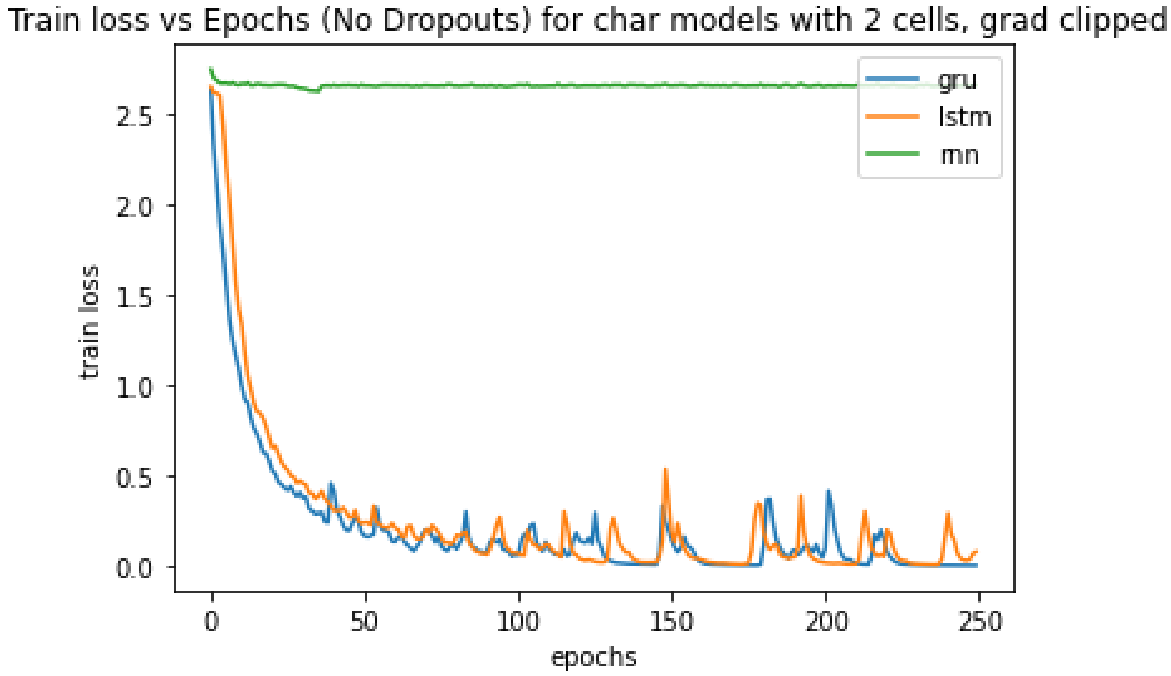


Figure 46. Train loss vs epochs for gradient clipped, stacked character RNNs

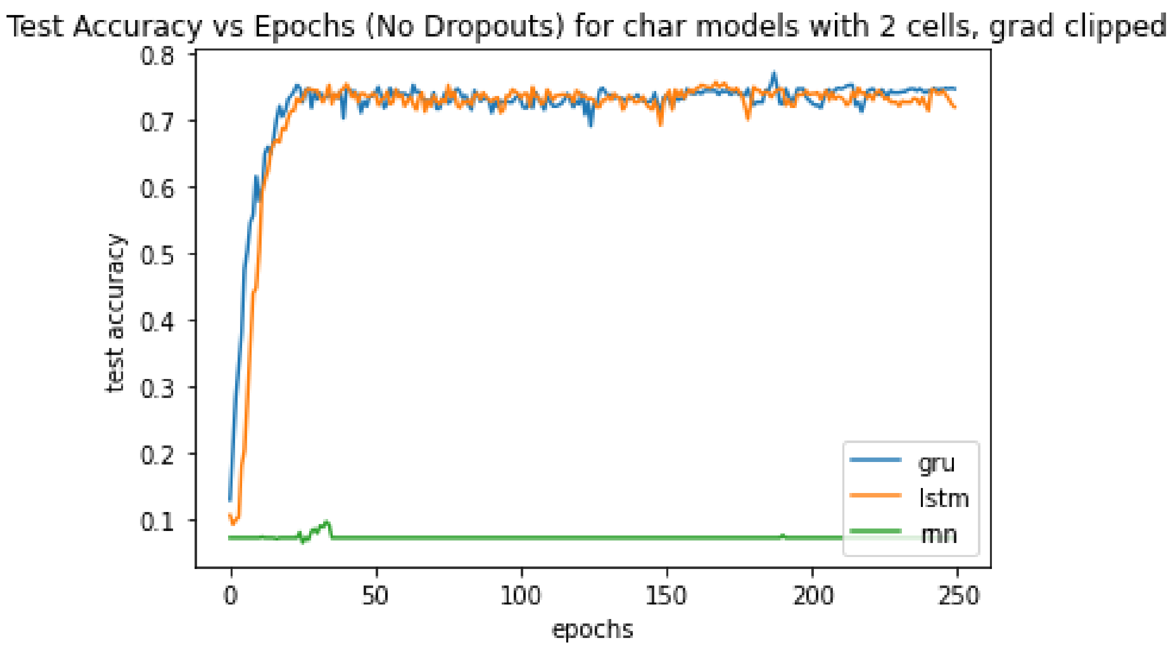


Figure 47. Test accuracy vs epochs for gradient clipped, stacked character RNNs

Word level: the train loss and test accuracy for word level, stacked RNNs are shown in figure *48 and 49 below*.

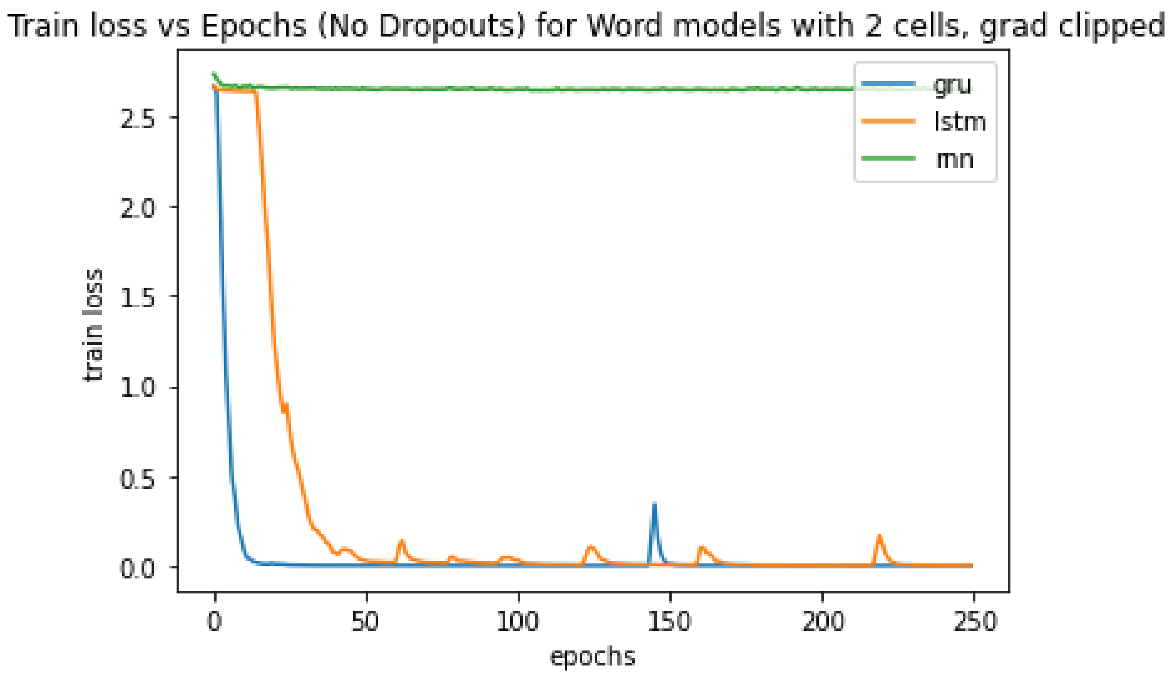


Figure 48. Train loss vs epochs for gradient clipped, stacked word RNNs

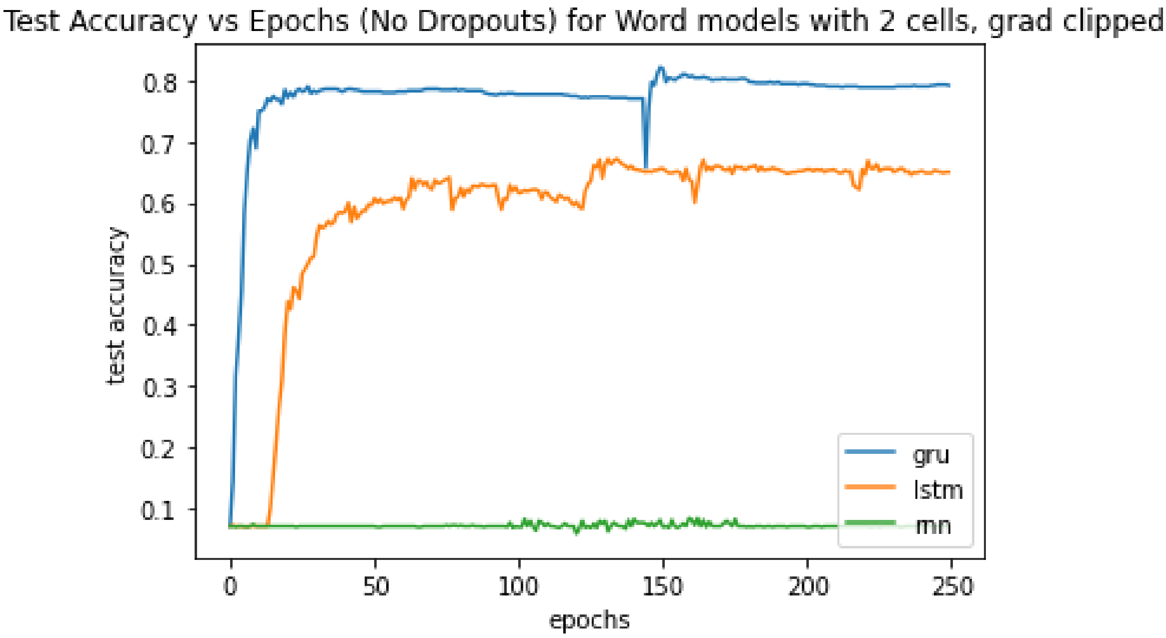


Figure 49. Test accuracies vs epochs for gradient clipped, stacked word RNNs

Results

The results for the twelve models are shown in the *table 6* below.

Table 6. Gradient clipping test accuracies for 6 models.

Model type	Cell Type	Test Accuracy
Character	GRU	0.71000
	LSTM	0.68143
	Vanilla RNN	0.14286
	Stacked GRU	0.77000
	Stacked LSTM	0.75571
	Stacked Vanilla RNN	0.09571
Word	GRU	0.87000
	LSTM	0.56714
	Vanilla RNN	0.10571
	Stacked GRU	0.82143
	Stacked LSTM	0.67143
	Stacked Vanilla RNN	0.08429

The **gradient clipped, unstacked GRU, word RNN has the highest test accuracy** while the Stacked Vanilla, word RNN has the lowest test accuracy. For character level modeling, the stacked GRU, character RNN has the best performance while the least test accuracy belongs to the Stacked Vanilla RNN.

d. Summary of findings

The combined results for experiment a, b, c is shown in *table 7*.

Table 7. Results for stacked, unstacked, clipped, unclipped experiments

Modelling Level	Cell Type	No clipping		Clipvalue = 0.2	
		Unstacked	Stacked	Unstacked	Stacked
Character	GRU	0.719	0.744	0.710	0.770
	LSTM	0.713	0.750	0.681	0.756
	Vanilla RNN	0.133	0.091	0.143	0.096
Word	GRU	0.881	0.830	0.870	0.821
	LSTM	0.654	0.666	0.567	0.671
	Vanilla RNN	0.101	0.089	0.106	0.084

Overall best, Best for character and best for word level

The best model for character level modelling is the Stacked GRU RNN with clipped value = 0.2 (test accuracy is 0.770).

The best model for word level modelling is the Unstacked, Unclipped GRU RNN (test accuracy is 0.881).

The worst model for character level modelling and word level modelling are vanilla RNNs, regardless of stacked, unstacked with gradient clipping or not.

Both LSTM and GRU (stacked and unstacked) are good models for character modelling (very similar test accuracies across stacked, unstacked, clipped, non-clipped experiments). For example, unstacked LSTM and GRU has similar 0.719 and 0.713 performance (no clipping), or stacked GRU and LSTM with clipping has 0.770 and 0.756 test accuracy respectively.

For word modelling, GRU is the best choice with test accuracies of at least 0.82 (82%) and above for all experiment stacked, unstacked, clipped, and non-clipped.

The best model overall for text classification given our dataset will be Word-level, unstacked, unclipped GRU RNN.

Effect of Stacked RNNs on different cell type and model type

For GRU RNNs in character-based modelling, stacking more layer of the cell generally increases the test accuracy of the model (with or without clipping). This trend is reversed in word-based modelling where stacking more layer will generally decrease the test accuracy of GRU RNNs.

For Vanilla RNNs, stacking more layers of the vanilla cell will strictly decrease the testing accuracies of the performance, regardless of modelling level (character or word) or the clipping of gradient.

For LSTM RNNs, stacking more layers will lead to higher testing accuracies and hence performance of models. This is regardless of different gradient clipping.

In short, for LSTM, by increasing the number of layers, we can improve the performance of the stacked model against this task. For GRU, this is only applicable for character level and not word level RNNs. More layers might lead to better performance as there are more memory unit to remember and learn the sequence better.

Effect of Gradient clipping on different cell type and model type.

The test accuracies and hence the performance of unstacked and stacked vanilla RNNs (both word and character level) are comparable between gradient clipping and without clipping. We compare between clipped stacked with unclipped stacked, clipped unstacked with unclipped unstacked.

The test accuracies of unstacked and stacked LSTM remain similar with and without gradient clipping. An outlier for this trend is the unstacked LSTM for word level modeling where a big dip in test accuracy is observed with the clipping (0.654 -> 0.567)

The test accuracies of unstacked and stacked GRU remain similar with and without gradient clipping. An outlier for this trend is the stacked GRU RNN, which has a big jump in performance with clipping.

In conclusion there are no concrete evidence that using clipping can enhance the testing accuracy of the model, which is used to evaluate their performances in this assignment. This might be the case because judging from the various training loss plots for question 6 a and b (figure 34, 36, 38 and 40), there are no gradient explosion in training the RNNs, hence the effect of with and without gradient clipping is invisible in part c.

Most of the values are very similar and there are a few outliers that might or might not indicate anything useful from using clipping in our case to lift the performance.

5. Conclusion

In this assignment, we have attempt to build CNNs and RNNs for the two tasks: Image Recognition and Text Classification.

For image recognition (a.k.a classification), higher number of filter maps at each convolution layers will lead to better test accuracy performance for the model. Besides, from our empirical observations, we can see that SGD with dropout regularization = 0.5 gives us the best performing model. The tradeoff between speed of training and optimal convergence between various optimizers have also been discussed, with preference for SGD should training resources are unlimited and best test accuracy is the goal for training and evaluation of model.

For text classification, we explore the use of CNNs and RNNs in solving this task at two different modeling level words and characters. We find that CNNs takes shorter time to train comparing to RNNs, but their performance is not optimal for the problem, comparing to the RNN models. We also found that the text classification task can be resolved at better performance if we choose word level rather than character level. We attempt to lift the performance by using dropouts on both CNNs and RNNs and the result is positive. On the other hand, we explore different RNN cells, different number of RNN cells stacked and gradient clipping in training better model. The results shown that we can obtain better model with stacking the RNN cells and by using GRU cells. There is however no concrete trend that we can devised from using gradient clipping on the performance of the model.

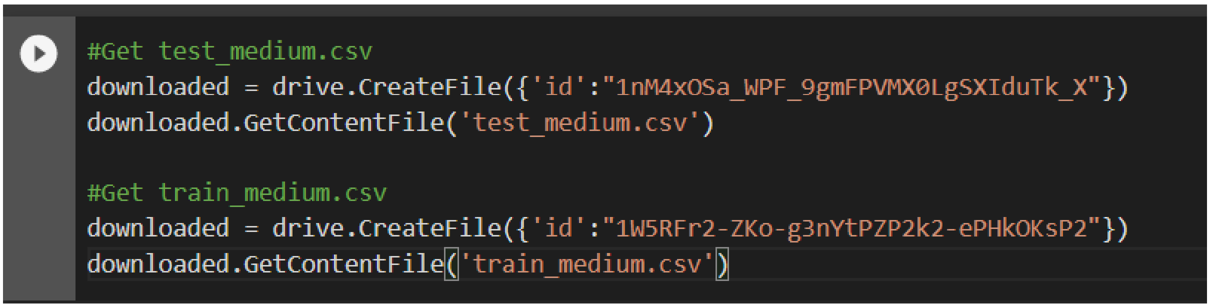
6. Annex

6.1. Annex A: Notebook indexes and steps to replicate results

The project is carried out on Google Colaboratory platform, therefore the submitted notebook are best to be run on Colab. To replicate the results of each notebook, make sure that you have uploaded the dataset ".csv" files to your own drive and obtain the "IDs" of these files. Please refer to the following links should you need help in how to obtain your .csv files id:

- https://buomsoo-kim.github.io/colab/2018/04/16/Importing-files-from-Google-Drive-in-Google-Colab.md/

The dataset can be accessed in the collab notebook as shown here. Replace the id here with your file id obtained from the guide above.



```
#Get test_medium.csv
downloaded = drive.CreateFile({'id':"1nM4xOSa_WPF_9gmFPVMX0LgSXIduTk_X"})
downloaded.GetContentFile('test_medium.csv')

#Get train_medium.csv
downloaded = drive.CreateFile({'id':"1W5RFr2-ZKo-g3nYtPZP2k2-ePHkOKsP2"})
downloaded.GetContentFile('train_medium.csv')
```

You must also have a google account to make use of their Google Cloud platform, which is a prerequisite for running Colab notebooks with dataset stored on Google Drive or to store your results (image, .h5 models) to your PC.

Once this is iron out, the notebook should be able to run.

In the submission, files look something like this:

```
Assignment2
        +PartA
                -1A.ipynb
                -2A.ipynb
                -3A.ipynb
                -data_batch_1
                -test_batch_trim
        +PartB
                +1B
                        -1B.ipynb
                +2B
                        -2B.ipynb
                +3B
                        -3B.ipynb
                +4B
                        -4B.ipynb
                +5B
                        -1B-dropout.ipynb
                        -2B-dropout.ipynb
                        -3B-dropout.ipynb
                        -4B-dropout.ipynb
                +6B
                        -6B-char-a.ipynb
```

-6B-word-a.ipynb
-6B-char-b.ipynb
-6B-word-b.ipynb
-6B-char-c.ipynb
-6B-word-c.ipynb
-6B-char-c-stacked.ipynb
-6B-word-c-stacked.ipynb

PartA folder contains the code for Part A of the assignment. The notebook 1A,2A and 3A store the results and code for question 1, 2 and 3 of this part (baseline, grid search and optimizers) accordingly

PartB folder contains the code for Part B of the assignment. The folder 1B-4B includes notebook and results for the CNNs and RNNs for question 1-4. The folder 5B contains the dropout included version of these models.

The 6B folder contains the notebook for question 6. 6B-char-a and 6B-word-a are notebook for experimenting with RNN, LSTM and GRU. 6B-char-b and 6B-word-b are notebooks for experimenting with stacking 2 layers. Lastly 6B-char-c, 6B-char-c-stacked, 6B-word-c and 6B-word-c-stacked are the notebook for gradient clipping for character and word RNNs with and without stacked architecture of cells.