

**CE4042**

**Neural Network & Deep Learning**

**Assignment 2 Report**

**Name: Neo Shun Xian Nicholas**

**Matric Number: U1820539F**

**Date: 11 November 2020**

Table of Contents

**Miscellaneous1**

**PART A: Object Recognition1**

Introduction2

Question 12

Question 22

Question 32

Question 42

**PART B: Text Classification4**

Introduction5

Question 12

Question 22

Question 32

Question 45

Question 52

Question 62

**Miscellaneous**

**Libraries/modules used in this assignment**

* numpy
* matplotlib
* imageio
* pickle
* tensorflow 2/keras
* csv
* re
* time
* pylab

**Coded in**

* Python
* Google Colaboratory (GPU instance)

**Files included in this project**

* Neo\_Shun\_Xian\_Nicholas\_A2\_report.pdf
* Neo\_Shun\_Xian\_Nicholas\_A2\_codes.zip
  + Part\_A.ipynb
  + Part\_B1.ipynb
  + Part\_B2.ipynb
  + Part\_B3.ipynb
  + Part\_B4.ipynb
  + data\_batch\_1
  + test\_batch\_trim
  + train\_medium.csv
  + test\_medium.csv

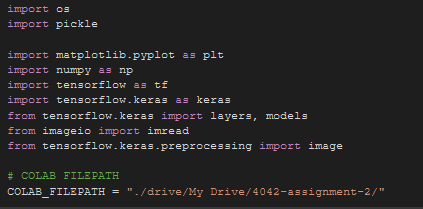
**PART A: Object Recognition**

**Introduction**

In this section, we are tasked to predict the correct class of the test dataset given the labelled training dataset. The dataset used in this section is the CIFAR-10 dataset. The dataset contains RGB colour images of size 32x32 and their corresponding label from 0 to 9.

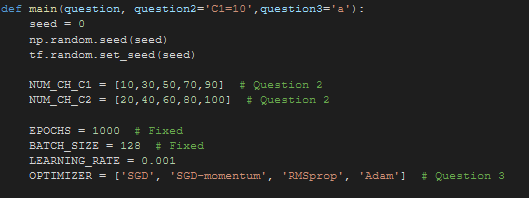
Import relevant libraries/modules

To execute and complete the analysis, these python libraries/modules are imported. Google Colaboratory (GPU instance) is used to run the analysis.



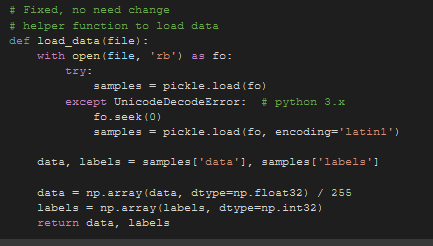
Constants used in Part A

For ease of coding and passing of parameters into the defined functions, these are the constants defined for all the questions in Part A.



Loading and pre-processing of data

Firstly, load both the dataset “data\_batch\_1” and “test\_batch\_trim” from Google Drive, using the helper function load\_data(file), as shown below.



For image dataset, the maximum value of each pixel of the RGB channel is 255. Hence, we will need to normalise our datasets that are passed into this function. We can do so by simply plickle.load our data and divide the numpy array value by 255. This ensures that the pixel values are normalised and stays within the range of 0 to 1 for better computations.

Helper functions

To ensure code readability and prevent repetition of similar codes, several helper functions are written:

***load\_data()***

* + Takes in the training and the testing data using pickle.load()
  + Returns the data in numpy array format and also the labels of the training data

***make\_model()***

* + Takes in the number of channels in the 2 layers and use it to train the model
  + Decide whether to implement dropout into the model or not
  + Returns the model

***train()***

* + Trains the model with different optimizers used
  + Saves the model

***plot\_image\_and\_feature\_map()***

* + Takes in a test image and use it to plot the feature map in various layers on the model

***plot()***

* + To plot the accuracy or loss of the model against the epoch count

***main()***

* + Acts as a switch to control which question should be executed in the code

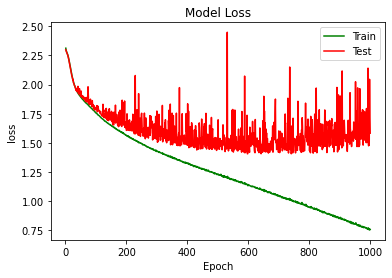
**Question 1**

Train the network using mini-batch gradient descent learning for 1000 epochs. Set the batch size to 128, and learning rate alpha=0.001.

a) Plot the (1) training cost, (2) test cost, (3) training accuracy, and (4) test accuracy

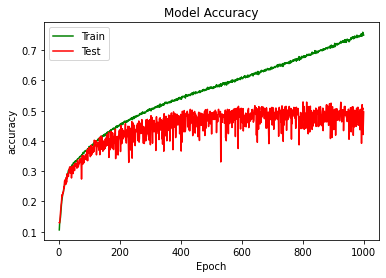
against learning epochs. One plot for the costs and one plot for the accuracies

**Training & Test Cost vs Epochs**

****

From the plot above, the train cost decreases gradually for 1000 epochs. The train cost can still be decreased further with more epochs. However, as can be observed from the test cost, the test cost plateaued and fluctuates for 1000 epochs at around the value of 1.50. Hence there may be signs of overfitting as the test cost begins to increase again from epoch 800 onwards.

**Training & Test Accuracy vs Epochs**



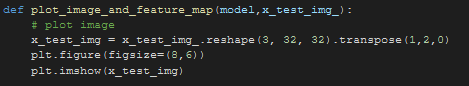
As seen from the plot above, the training accuracy increases gradually, while the test accuracy increases for the first 300 epochs and plateaued with an accuracy of 49.600%. There are still some fluctuations in the test accuracy as the training progresses.

b) For the first two test images, plot the feature maps at both convolution layers

(𝐶1 and 𝐶2) and pooling layers (𝑆1 and 𝑆2) along with the test images. (In total one

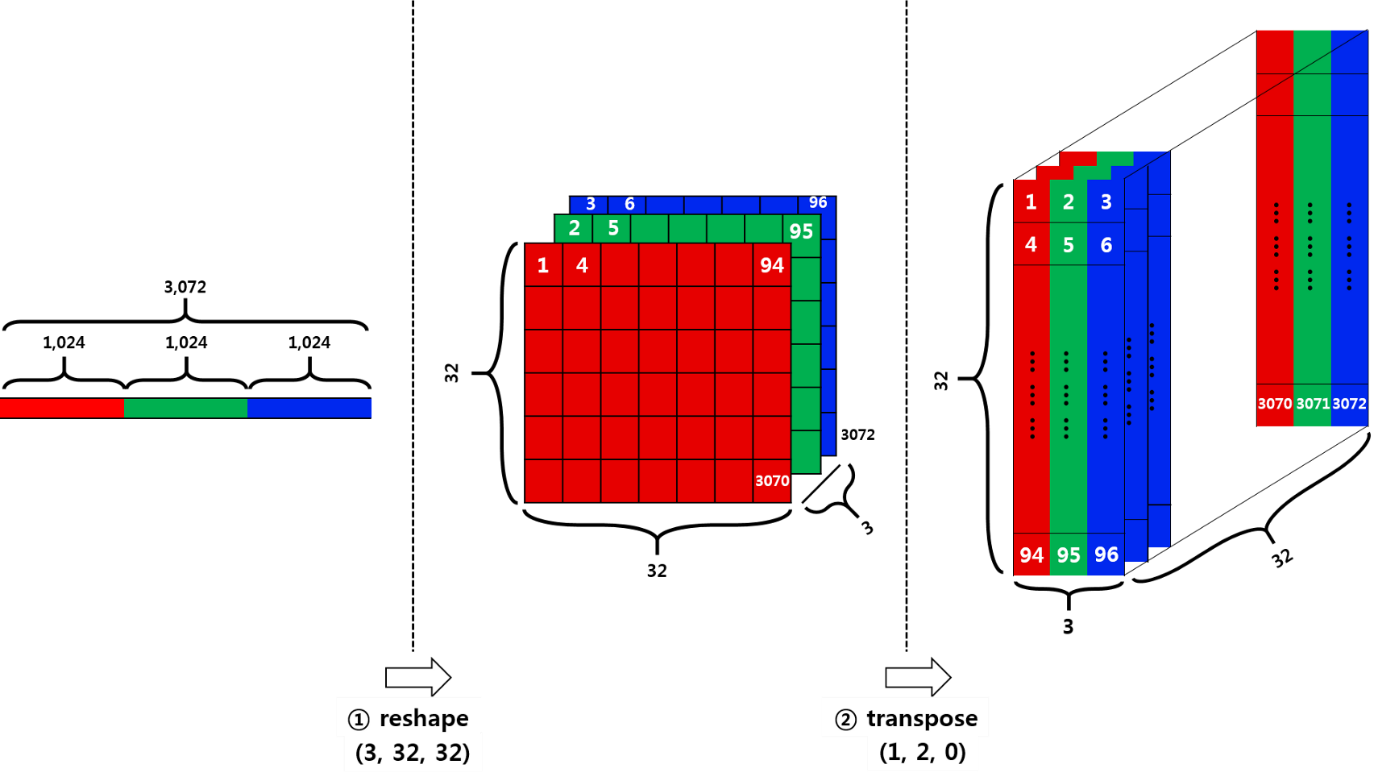
image and four feature maps)

To plot the test images, a function “plot\_image\_and\_feature\_map()” is defined and the row vector version of the test image is passed into this function. The row vector is then pre-processed, and it has 32\*32\*3=3072 elements. Two steps are required to reshape the row vector into the (width \* height \* num\_channel) from. The first step is to use the reshape function to reshape the image matrix. The second step is to use the transpose function in numpy. A code snippet of how the transformation is done is as shown from below:



From the CIFAR-10 dataset analysis by <https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c>, we first need to split the row vector into 3 pieces, each row represents a colour channel. For this, the resulting array will have (3x1024) matrix, making a (10000x3x1024) tensors in total. Then, divide the 3 pieces further by 32 because 32 is the width and height of the image. This results in (3x32x32) matrix, making it a (10000x3x32x32) tensors. As such, reshape function should be called with argument (10000x3x32x32).

From the pre-processing, observe that that one image data is represented as (num\_channel, width, height), a dimension that we are not expecting. We are expecting the data to be of the (width, height, num\_channel) dimension. To ensure that the image being fed in is the desired dimension, there is a need to swap the order of each axes using transpose. The transpose with argument (1,2,0) will be called as it will change the numpy array from (num\_channel, width, height) to (width, height, num\_channel). A diagram below illustrates the pre-processing.



Source: <https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c>

**Analysis for image 1**

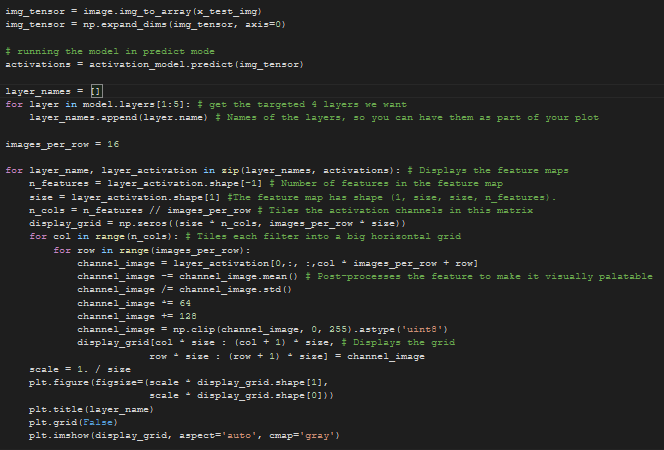
After pre-processing of the row vector, the first test image is as shown:



To plot the feature maps at both the convolution layers (C1 & C2) and pooling layers (S1 & S2), we first sift out the target layers first that we want to show (that is, C1, C2 , S1 & S2) as shown below.



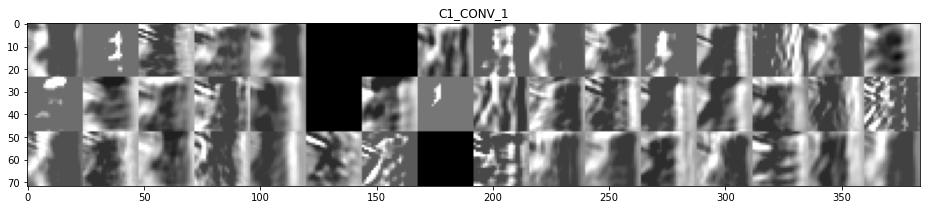
Then, we do some pre-processing to each of the layer so as to show the feature maps in the 4 layers. The corresponding code snippet is as shown below.



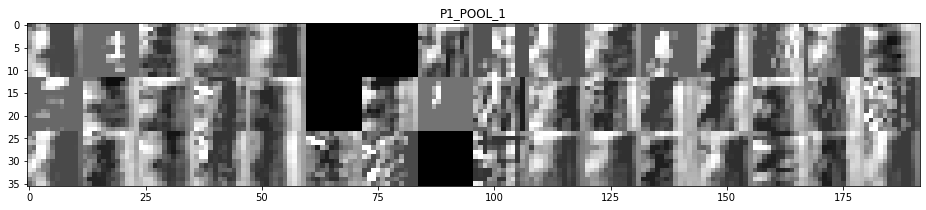
Source: <https://towardsdatascience.com/visualizing-intermediate-activation-in-convolutional-neural-networks-with-keras-260b36d60d0>

The four feature maps of the first image are as shown below:

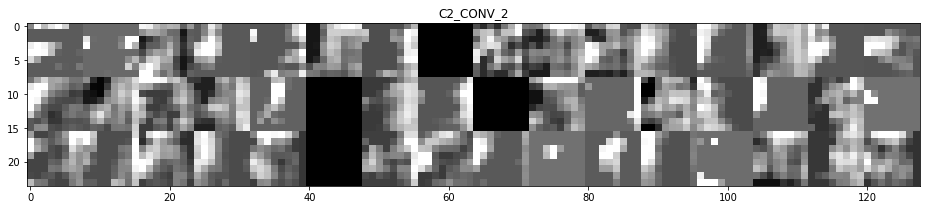
**Convolution layer C1**

****

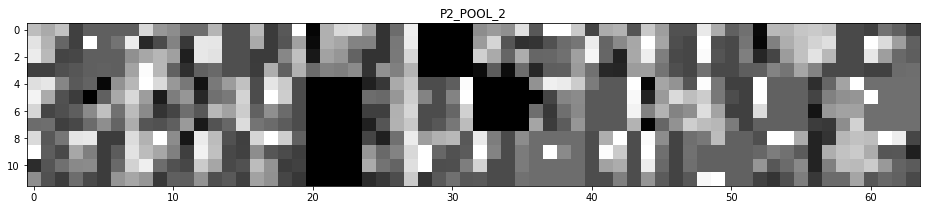
**Pooling layer S1**

****

**Convolution layer C2**

****

**Pooling layer S2**

****

As seen from the 4 feature maps above, full black images could be seen in the feature maps. This is because some of the filters are not activated yet. The first layer **C1,**retains almost all of the information present in the first test image. As we go deeper in the layers, the activations become increasingly abstract and less visually interpretable. They begin to encode higher-level concepts such as single borders, corners and angles. Higher presentations carry increasingly less information about the visual contents of the image, and increasingly more information related to the class of the image.

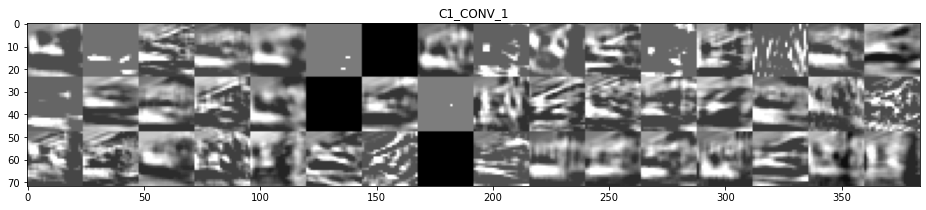
**Analysis for image 2**

Similarly, for the second test image, we obtained the following test image and feature maps.

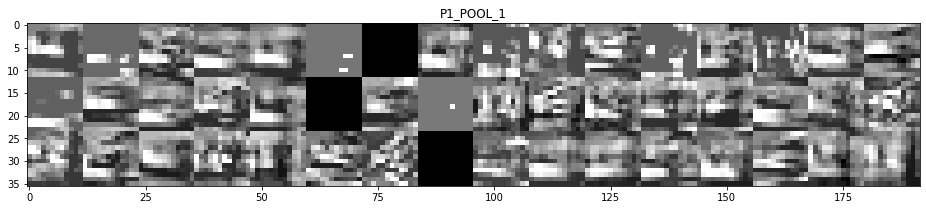


The four feature maps of the second image are as shown below:

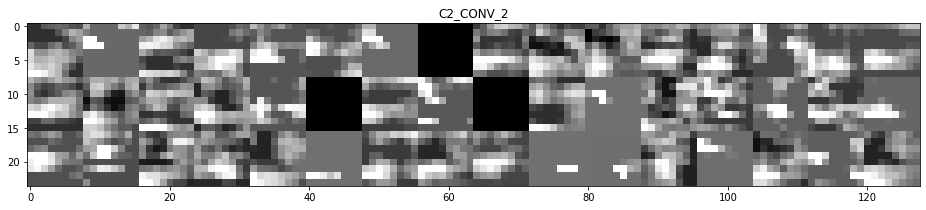
**Convolution layer C1**

****

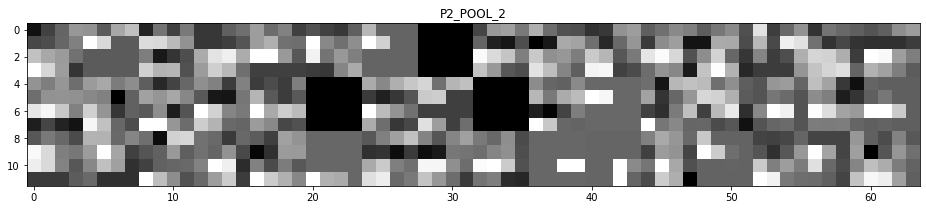
**Pooling layer S1**

****

**Convolution layer C2**

****

**Pooling layer S2**

****

**Question 2**

Use a grid search ( 𝐶1 = {10, 30, 50, 70, 90}, 𝐶2 = {20, 40, 60, 80, 100} , in total 25

combinations) to find the optimal combination of the numbers of channels at the

convolution layers. Use the test accuracy to determine the optimal combination. Report

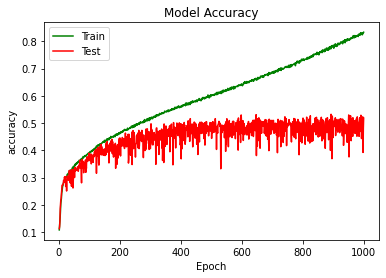
all 25 accuracies.

Using grid search, there will be 25 possible combinations of 𝐶1 & 𝐶2.Test accuracy is used to determine the optimal combination. Here are all the 25 test accuracies in terms of percentage (%).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 𝐶1 \ 𝐶2 | 20 | 40 | 60 | 80 | 100 |
| 10 | 46.550 | 49.250 | 47.500 | 48.700 | 48.900 |
| 30 | 46.350 | 46.700 | 46.250 | 51.200 | 48.750 |
| 50 | 47.350 | 49.350 | 48.700 | 47.100 | 51.350 |
| 70 | 48.400 | 51.000 | 51.050 | 50.550 | 48.050 |
| 90 | 49.250 | 50.300 | 50.450 | 49.850 | 51.850 |

From the table above, the highest test accuracy obtained is 51.850% with 𝐶1 and 𝐶2 equal to 90 and 100 respectively. Therefore, the optimal condition for 𝐶1 and 𝐶2 is 90 and 100 respectively.

Below is the accuracy plot against number of epochs of the most optimal condition.



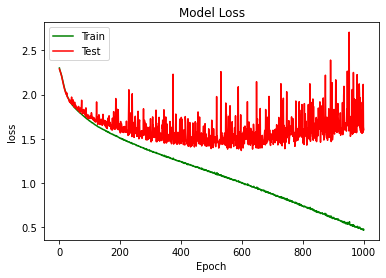
**Question 3**

**Optimal C1 = 90 and C2 = 100**

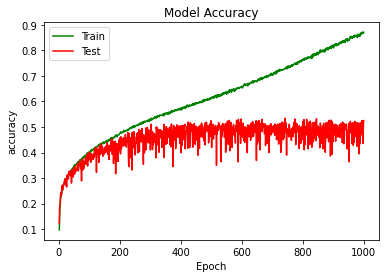
Using the optimal combination found in part (2), train the network by:

a) adding the momentum term with momentum 𝛾 = 0.1

**Costs:**

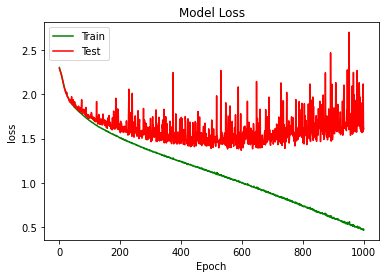
****

**Accuracy: 52.550% (Test)**

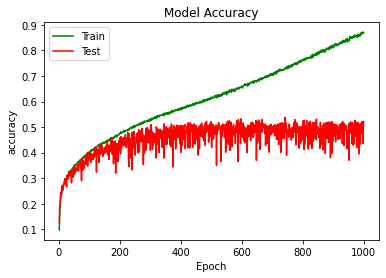
****

b) using RMSProp algorithm for learning

**Costs**

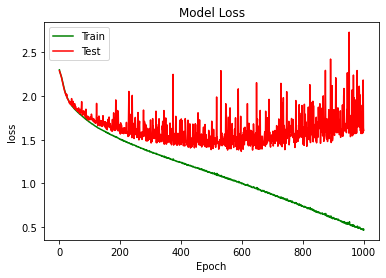


**Accuracy: 52.200% (Test)**

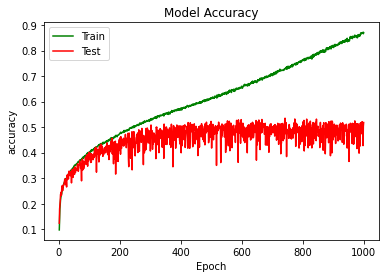


c) using Adam optimizer for learning

**Costs**

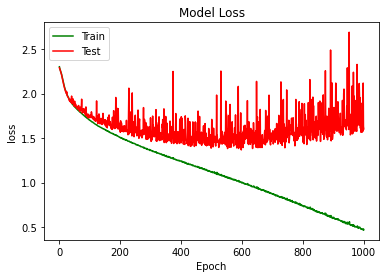


**Accuracy: 51.800% (Test)**

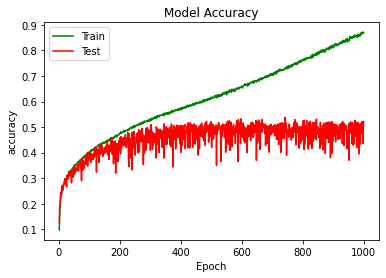


d) adding dropout (probability=0.5) to the two fully connected layers.

**Costs**



**Accuracy: 52.250% (Test)**



**Question 4**

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

A comparison table of all the models from question 1 to 3 are as shown below.

|  |  |
| --- | --- |
| **Model** | **Test Accuracy (%)** |
| C1 = 50, C2 = 60 | 49.600 |
| C1 = 10, C2 = 20 | 46.550 |
| C1 = 10, C2 = 40 | 49.250 |
| C1 = 10, C2 = 60 | 47.500 |
| C1 = 10, C2 = 80 | 48.700 |
| C1 = 10, C2 = 100 | 48.900 |
| C1 = 30, C2 = 20 | 46.350 |
| C1 = 30, C2 = 40 | 46.700 |
| C1 = 30, C2 = 60 | 46.250 |
| C1 = 30, C2 = 80 | 51.200 |
| C1 = 30, C2 = 100 | 48.750 |
| C1 = 50, C2 = 20 | 47.350 |
| C1 = 50, C2 = 40 | 49.350 |
| C1 = 50, C2 = 80 | 47.100 |
| C1 = 50, C2 = 100 | 51.350 |
| C1 = 70, C2 = 20 | 48.400 |
| C1 = 70, C2 = 40 | 51.000 |
| C1 = 70, C2 = 60 | 51.050 |
| C1 = 70, C2 = 80 | 50.550 |
| C1 = 70, C2 = 100 | 48.050 |
| C1 = 90, C2 = 20 | 49.250 |
| C1 = 90, C2 = 40 | 50.300 |
| C1 = 90, C2 = 60 | 50.450 |
| C1 = 90, C2 = 80 | 49.850 |
| C1 = 90, C2 = 100 | 51.850 |
| C1 = 90, C2 = 100 + momentum 𝛾 = 0.1 | 52.550 |
| C1 = 90, C2 = 100, RMSprop | 52.200 |
| C1 = 90, C2 = 100, Adam | 51.800 |
| C1 = 90, C2 = 100 + dropout (probability = 0.5) | 52.250 |

As seen from the above table, the model with C1 = 90, C2 = 100 and a momentum of 0.1 yields the highest accuracy of 52.550% among all the other models. Generally, with the addition of momentum, dropout or change the optimiser to RMSprop yields an even higher accuracy as compared to its base model. However, when SGD is changed to Adam as optimiser, the test accuracy drops, implying that Adam optimiser might not be suitable for this object recognition task. Furthermore, we can see that using dropout or RMSprop optimizer also improve the model’s accuracy.

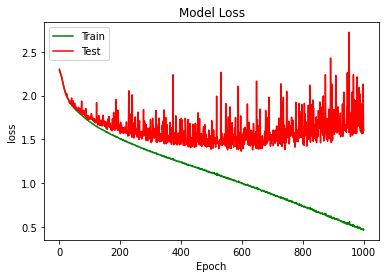
Further Analysis

Since adding momentum, dropout or using RMSprop as optimizer improves the test accuracy, 2 other further analysis of models will be done to see if the test accuracy of the model can be improved further:

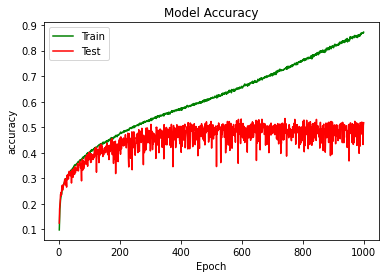
* + 1. C1 = 90, C2 = 100, SGD + Momentum + Dropout (probability = 0.5)
    2. C1 = 90, C2 = 100, RMSprop + Dropout (probability = 0.5)

1. C1 = 90, C2 = 100, SGD + Momentum + Dropout (probability = 0.5)

**Costs**

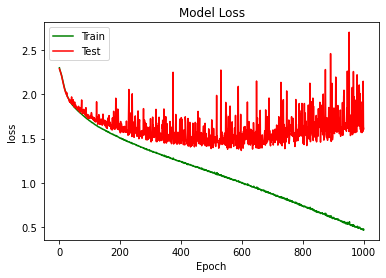


**Accuracy: 51.800% (Test)**

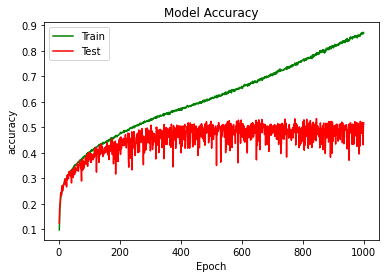


2. C1 = 90, C2 = 100, RMSprop + Dropout (probability = 0.5)

**Costs**



**Accuracy: 51.800% (Test)**



From the 2 analysis above, it seems that the model did not improve further but the accuracy has decreased. As such, the most optimal model is still C1 = 90, C2 = 100 + momentum 𝛾 = 0.1 with a test accuracy of 52.550%

**Overall Conclusion**

In conclusion, having analysed all the above model, we can see that in general, the Word RNN classifier with GRU model gives the best performance among the 4 classifiers we analysed. Then, we include a gradient clipping of 2 to further improve on the Word RNN classifier.

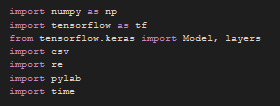
**PART B: Text Classification**

**Introduction**

In this section, we are tasked to predict the correct class of the test dataset given the labelled training dataset. The dataset used in this section is the train\_medium.csv and test\_medium.csv dataset from the first paragraph of the Wikipage entries. We will implement CNN and RNN layers at the word and character levels for the classification of texts in the paragraphs.

Import relevant libraries/modules

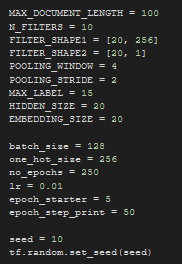
To execute and complete the analysis, these python libraries/modules are imported. Google Colaboratory (GPU instance) is used to run the analysis.



**Question 1**

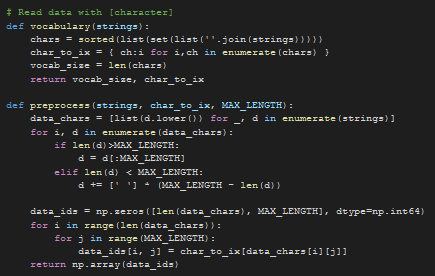
Constants used in question 1

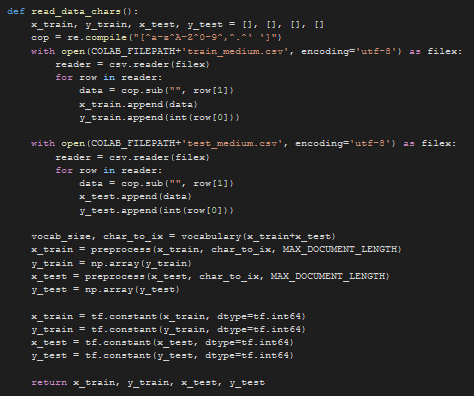
For ease of coding and passing of parameters into the defined functions, these are the constants defined for question 1.



Loading of data and pre-processing of data

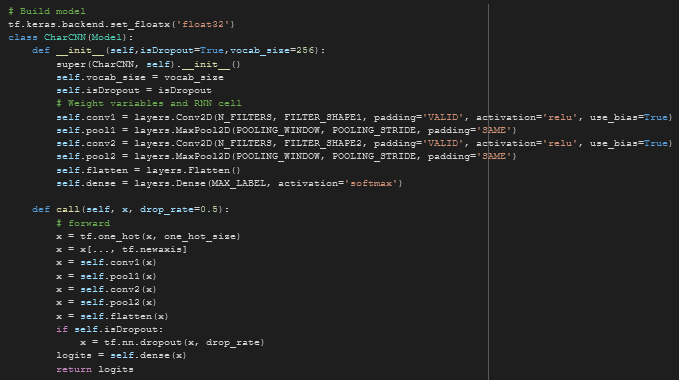
Firstly, load both the dataset “train\_medium.csv” and “test\_medium.csv” from Google Drive, using the helper functions vocabulary(), preprocess() and read\_data\_chars() as shown below.



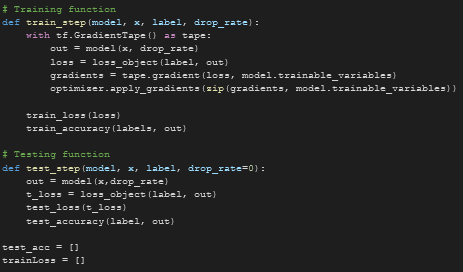


Design a Character CNN Classifier that receives character ids and classifies the input.

To implement the required model, a class name CharCNN is created to build the layers in the model as shown below:



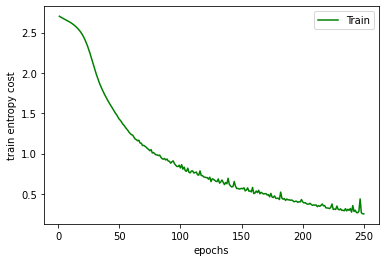
Then, the training and testing function is defined, named “train\_step” & “test\_step” respectively. For the train step, a manual training loop is used using the tf.GradientTape() class. Code snippets of how it is implemented is shown below:



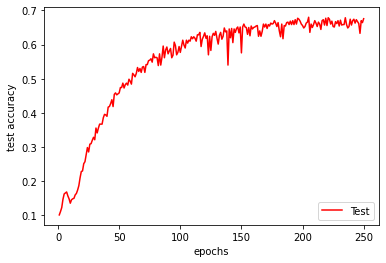
Plot the entropy cost on the training data and the accuracy on the testing data against

training epochs.

**Entropy Cost of the training data**

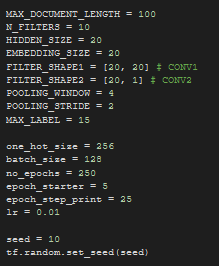


**Test Accuracy: 67.571%**

****

**Question 2**

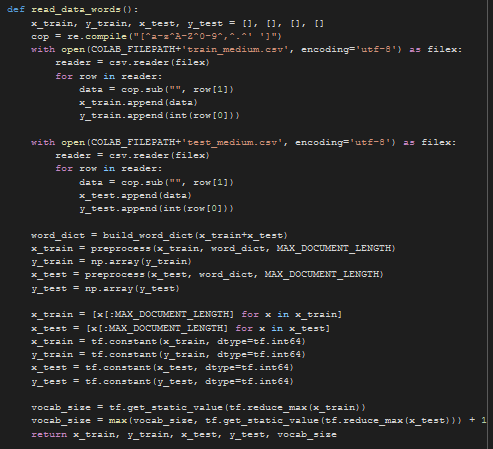
Constants used in question 2



Loading of data and pre-processing of data

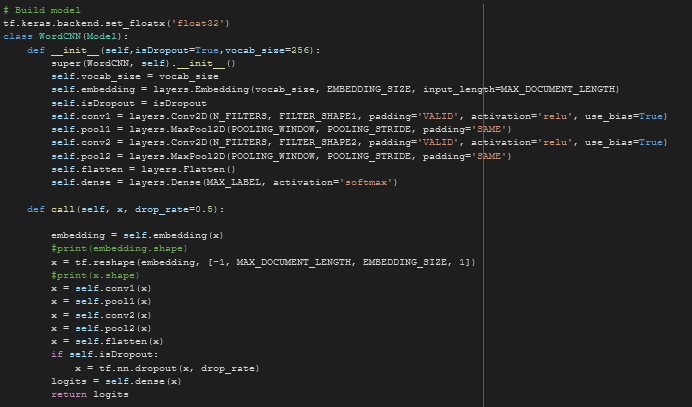
Firstly, load both the dataset “train\_medium.csv” and “test\_medium.csv” from Google Drive, using the helper functions clean\_str(), build\_word\_dict(), preprocess() and read\_data\_words() as shown below.





Design a Word CNN Classifier that receives word ids and classifies the input.

To implement the required model, a class name WordCNN is created to build the layers in the model as shown below:

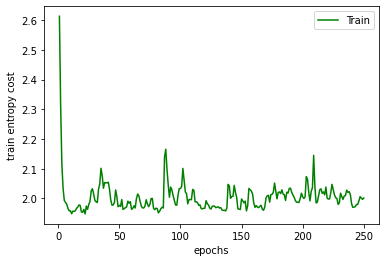


The training and testing functions are the same as the ones defined in question 1.

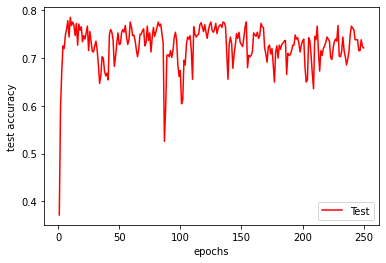
Plot the entropy cost on the training data and the accuracy on the testing data against

training epochs.

**Entropy Cost of the training data**

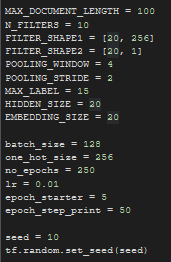


**Test Accuracy: 72.143%**



**Question 3**

Constants used in question 3

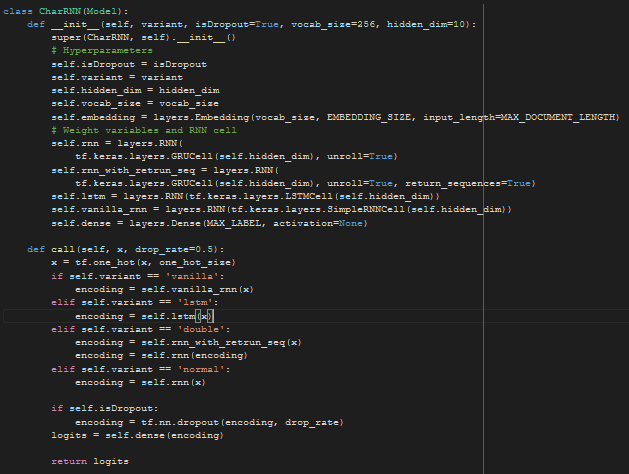


Loading of data and pre-processing of data

The loading and pre-processing of data is the same as the ones in question 1.

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

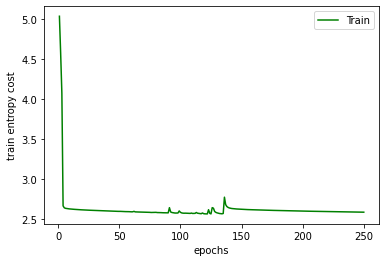
To implement the required model, a class name CharRNN is created to build the layers in the model as shown below:



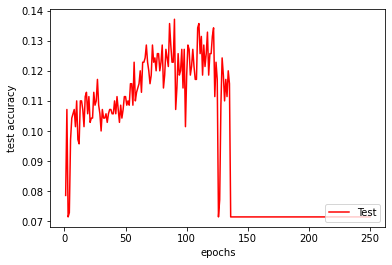
The training and testing functions are the same as the ones defined in question 1.

Plot the entropy cost on the training data and the accuracy on the testing data against training epochs.

**Entropy Cost of the training data**

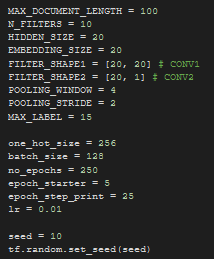


**Test Accuracy: 7.1429%**



**Question 4**

Constants used in question 4

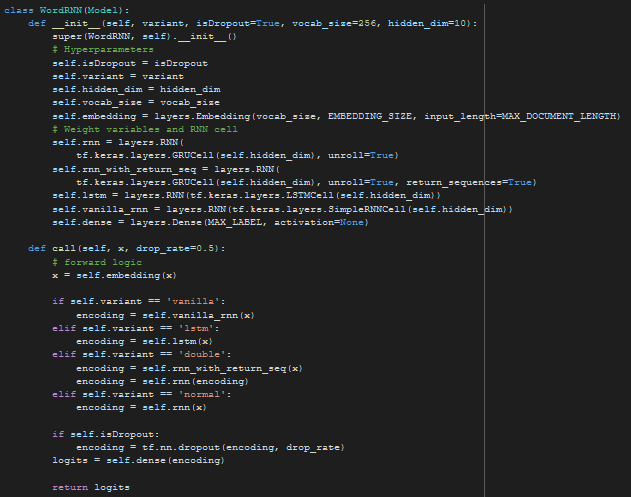


Loading of data and pre-processing of data

The loading and pre-processing of data is the same as the ones in question 2.

Design a Word RNN Classifier that receivesword ids and classifies the input.

To implement the required model, a class name WordRNN is created to build the layers in the model as shown below:

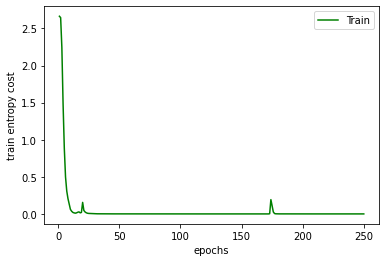


The training and testing functions are the same as the ones defined in question 1.

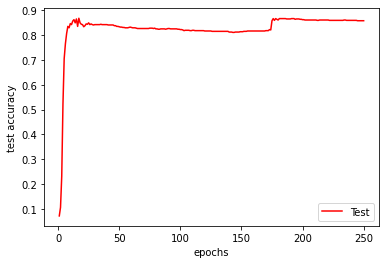
Plot the entropy on the training data and the accuracy on the testing data against

training epochs.

**Entropy Cost of the training data**



**Test Accuracy: 85.857%**



**Question 5**

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Test Accuracy (%)** | **Running Time for 250 epochs (s)** | **Running Time per epoch (s)** |
| **Character CNN Classifier** | 67.571 | 189.60 | 0.75839 |
| **Word CNN Classifier** | 72.143 | 250.07 | 1.0003 |
| **Character RNN Classifier** | 7.1429 | 3591.9 | 14.367 |
| **Word RNN Classifier** | 85.857 | 2420.6 | 9.6823 |

From the table above, **Character CNN Classifier** trains the fastest for 250 epochs and **Word RNN Classifier** has the highest test accuracy. Looking back at Question 1 to 4 plots, we can see that the test accuracy for **Character CNN Classifier & Word RNN Classifier** are more stable and has less fluctuations as compared to the other 2 models. With these analyses, the **Character CNN Classifier & Word RNN Classifier** model may be more suited for the text classification problem.

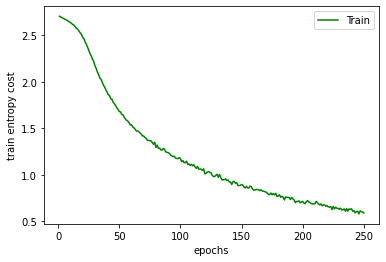
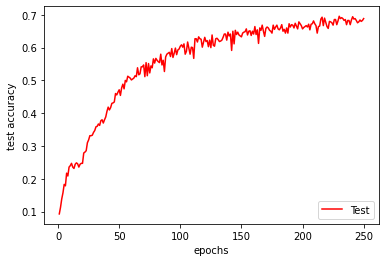
Experiment with adding dropout to the layers of networks in parts (1) – (4) and report the test accuracies. Compare and comment on the accuracies of the networks

with/without dropout.

In this experiment, the dropout probability will be set to 0.5.

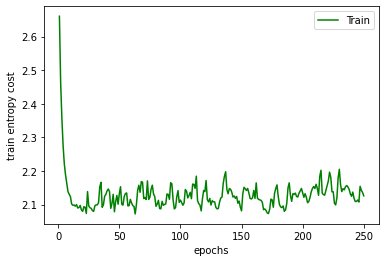
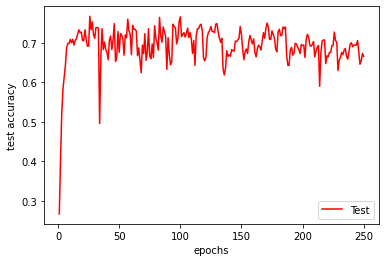
**Character CNN Classifier with dropouts**

Train entropy cost and test accuracy (68.857%)

** **

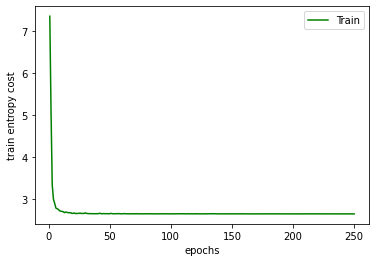
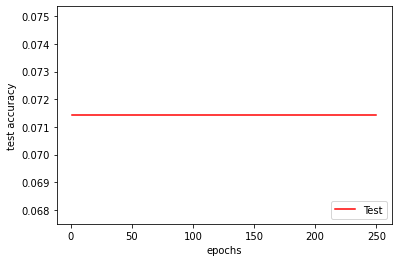
**Word CNN Classifier with dropouts**

Train entropy cost and test accuracy (66.571%)

** **

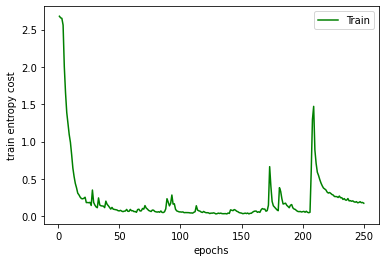
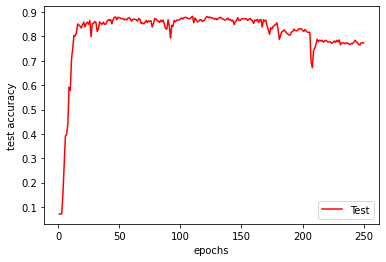
**Character RNN Classifier with dropouts**

Train entropy cost and test accuracy (7.1429%)

** **

**Word RNN Classifier with dropouts**

Train entropy cost and test accuracy (77.286%)

** **

**Comparison Table**

|  |  |  |
| --- | --- | --- |
| **Model** | **Test Accuracy Without Dropout (%)** | **Test Accuracy With Dropout (%)** |
| **Character CNN Classifier** | 67.571 | 68.857 |
| **Word CNN Classifier** | 72.143 | 66.571 |
| **Character RNN Classifier** | 7.1429 | 7.1429 |
| **Word RNN Classifier** | 85.857 | 77.286 |

From the table above, other than the Character CNN Classifier and the Character RNN Classifier, the test accuracies of the 2 other models with dropout are worse than the model without dropout. The gain in accuracy by the dropout model from the without dropout model of the **Character CNN Classifier** is **far less than the** loss in accuracy by the without dropout model from the with dropout model of the **Word CNN Classifier** and the **Word RNN Classifier**. This may mean that when building a word classifer, dropout might not be suitable as compared to the model with character classifier.

There might be another reason why dropout will hurt the performance of the model. The model that were build here were relatively small as compared to the dataset, causing dropout (a regularisation technique) to be unneccessary. Also, training for only 250 epochs might not be sufficient to train the model till convergence. Dropout hurts the performance at the start of the training but will result in the final converged error to be lower, implying that we should train with more epochs to see its effect in the lower cost.

In general, it would be better if the models are trained without dropout.

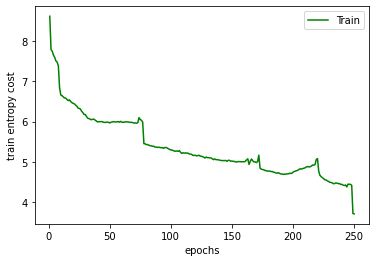
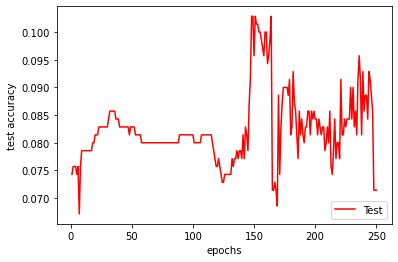
**Question 6**

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

**Variations to Character RNN Classifier**

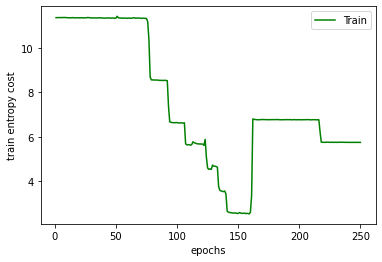
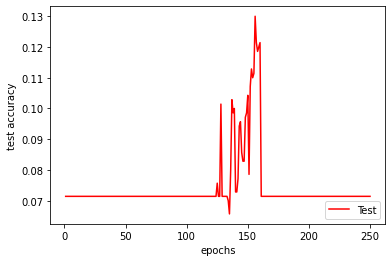
a)(i) Replace the GRU layer with a vanilla RNN layer

Train entropy cost and test accuracy (7.1428%)

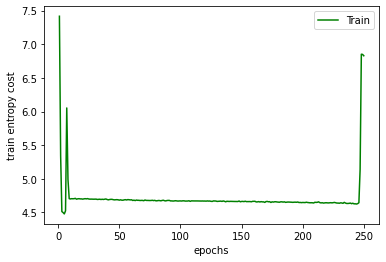
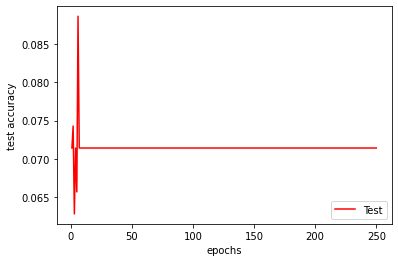
a)(ii) Replace the GRU layer with an LSTM layer

Train entropy cost and test accuracy (7.1428%)

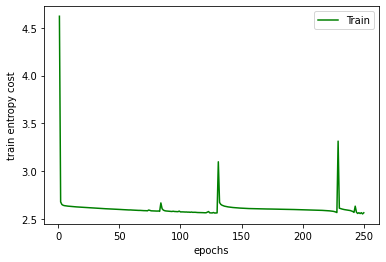
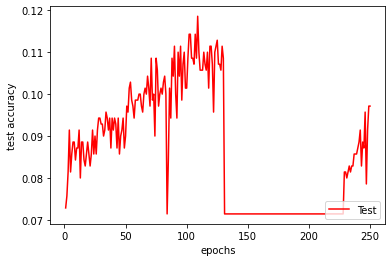
b) Increase the number of RNN layers to 2 layers

Train entropy cost and test accuracy (7.1428%)

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

Train entropy cost and test accuracy (7.1428%)

**Comparison Table**

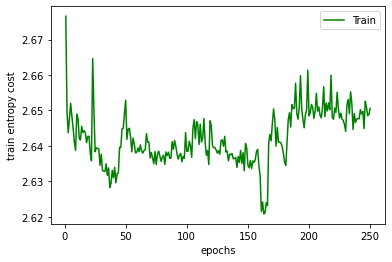
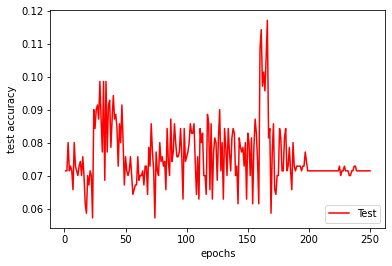
|  |  |
| --- | --- |
| **Model** | **Test Accuracy (%)** |
| Character RNN with GRU | 7.1248 |
| Character RNN with Vanilla RNN | 7.1248 |
| Character RNN with LSTM | 7.1248 |
| Character RNN with 2 layers of GRU | 7.1248 |
| Character RNN with GRU + gradient clipping | 7.1248 |

From the table above, we can see that the test accuracies of all the variations of the Character RNN model are pretty much the same, with some model having some fluctuations in the test accuracies with increasing epochs. Hence, performance did not improve.

**Variations to Word RNN Classifier**

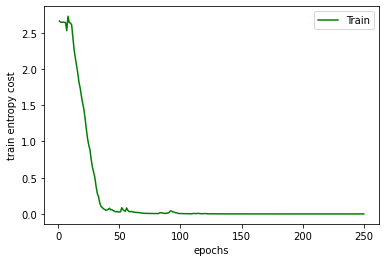
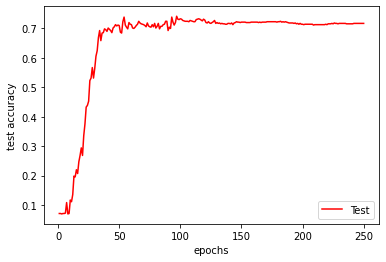
a)(i) Replace the GRU layer with a vanilla RNN layer

Train entropy cost and test accuracy (7.1428%)

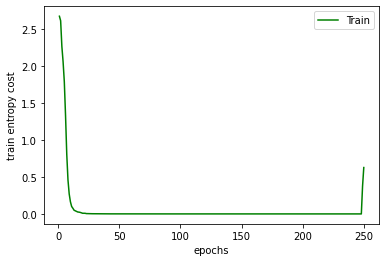
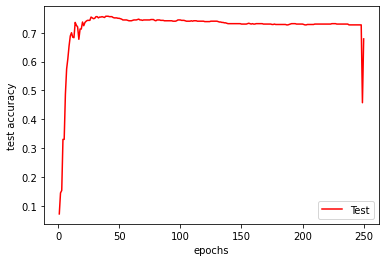
a)(ii) Replace the GRU layer with an LSTM layer

Train entropy cost and test accuracy (71.714%)

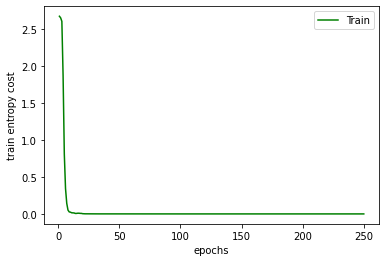
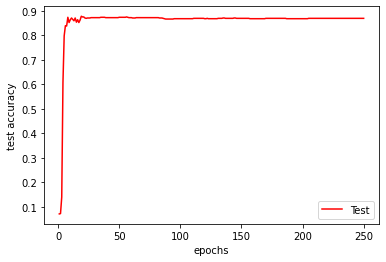
b) Increase the number of RNN layers to 2 layers

Train entropy cost and test accuracy (67.857%)

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

Train entropy cost and test accuracy (87.000%)

**Comparison Table**

|  |  |
| --- | --- |
| **Model** | **Test Accuracy (%)** |
| Word RNN with GRU | 85.857 |
| Word RNN with Vanilla RNN | 7.1248 |
| Word RNN with LSTM | 71.714 |
| Word RNN with 2 layers of GRU | 67.857 |
| Word RNN with GRU + gradient clipping | 87.000 |

From the table above, we can see that the test accuracies of most of the variations of the Word RNN model did not improve the test accuracy as compared to the base GRU model **except** the GRU model with gradient clipping in it. The accuracy with gradient clipping improved from 85.857% to 87.000%. Hence, adding gradient clipping to the GRU model helps to improve its performance.

**Overall Conclusion**

In conclusion, having analysed all the above model, we can see that in general, the Word RNN classifier with GRU model gives the best performance among the 4 classifiers we analysed. Then, we include a gradient clipping of 2 to further improve on the Word RNN classifier.