CZ4042 Neural Networks

Assignment

Assignment 2: Object Recognition of the CIFAR-10 dataset and Text classification of Wikipage entries

Submitted by:

Eddy Lim Qing Yang	U1620115B	eddy0006@e.ntu.edu.sg
See Kar Teck	U1622068J	ksee007@e.ntu.edu.sg

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

AY2019/2020 SEMESTER 1

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

In this report, 2 problems will be explored:

- Part A: Object Recognition of the CIFAR-10 dataset. Experiments will be discussed in Section 3.1.
- Part B: Text classification of Wikipage entries. Experiments will be discussed in Section 3.2.

1.1 Report overview

- In Section 2, experimental setup and details will be discussed.
- In Section 3, diagrams and data obtained from the experiments conducted will be shown. Results and answers to all queries will be discussed, along with the conclusions to the findings.

Chapter 2

Methods

In this chapter, the experimental setup done will be discussed.

2.1 Part A: Object Recognition of the CIFAR-10 dataset

In this problem, experiments were carried out in a google colab virtual machine (VM), using python 3.x and CPU clock of 2.3 GHz and Tesla K80 as the GPU, with 12GB gpu ram (specifications obtained from [1]).

The imports used in this problem is shown in Listing 2.1.

```python
import math
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pickle
from tqdm import tqdm_notebook as tqdm
import sys
```

Listing 2.1: Imports used for problem 1

2.2 Part B: Text classification of Wikipage entries

Similarly in this problem, experiments were carried out in a google colab virtual machine (VM), using python 3.x and CPU clock of 2.3 GHz and Tesla K80 as the GPU, with 12GB gpu ram (specifications obtained from [1]).

The imports used in this problem is shown in Listing 2.2

```python
import numpy as np
import pandas
import tensorflow as tf
import matplotlib.pyplot as plt
import csv
import time
```

```python
from tqdm import tqdm_notebook as tqdm
```

Listing 2.2: Imports used for problem 2

Chapter 3

Experiment Results and conclusion

3.1 Part A: Object Recognition of the CIFAR-10 dataset

The details of this dataset are as follows:

> The dataset contains RGB colour images of size 32 X 32 and their labels from 0 to 9. You will be using the batch-1 of the dataset, which contains 10,000 training samples. Testing is done on 2,000 test samples. The training data and testing data are provided in files 'data_batch_1' and 'test_batch_trim' files, respectively. Sample code is given in file start_project_2a.py Design a convolutional neural network consisting of:
>
> - An Input layer of 3x32x32 dimensions
> - A convolution layer C_1 of 50 filters of window size 9x9, VALID padding, and ReLU neurons. A max pooling layer S_1 with a pooling window of size 2x2, with stride = 2 and padding = 'VALID'.
> - A convolution layer C_2 of 60 filters of window size 5x5, VALID padding, and ReLU neurons. A max pooling layer S_2 with a pooling window of size 2x2, with stride = 2 and padding = 'VALID'.
> - A fully connected layer F_3 of size 300.
> - A softmax layer F_4 of size 10.

The definition of the cnn network as required by the question is shown in Listing 3.1, whereas the loss minimisation and accuracy prediction is shown in Listing 3.2.

```python
def cnn(images, num_filter_1, num_filter_2, dropout=False):
    # NHWC format
    images = tf.reshape(images, [-1, IMG_SIZE, IMG_SIZE, NUM_CHANNELS])

    #Conv 1, 50 filters of window size 9x9, VALID padding, and ReLU
    with tf.variable_scope('CNN_Layer1'):
        conv1 = tf.layers.conv2d(
            images,
            filters=num_filter_1,
            kernel_size=[9,9],
                padding='VALID',
                activation=tf.nn.relu)
        pool1 = tf.layers.max_pooling2d(
                conv1,
                pool_size=2,
                strides=2,
                padding='VALID')
        if dropout:
                pool1 = tf.layers.dropout(pool1, 0.25)

    with tf.variable_scope('Char_CNN_Layer2'):
        conv2 = tf.layers.conv2d(
                pool1,
                filters=num_filter_2,
                kernel_size=[5,5],
                padding='VALID',
                activation=tf.nn.relu)
        pool2 = tf.layers.max_pooling2d(
                conv2,
                pool_size=2,
                strides=2,
                padding='VALID')
        if dropout:
                pool2 = tf.layers.dropout(pool2, 0.25)

    dim = pool2.get_shape()[1].value * pool2.get_shape()[2].value * pool2.
   get_shape()[3].value
    pool2_ = tf.reshape(pool2, [-1, dim])

    # Fully connected layer size 300
    f3 = tf.layers.dense(pool2_, 300, activation=tf.nn.relu)
    if dropout:
        f3 = tf.layers.dropout(f3, 0.5)

    #Softmax, size 10. Note that softmax happens at softmax_entropy step
    f4 = tf.layers.dense(f3, NUM_CLASSES, activation=None)

    return conv1, pool1, conv2, pool2, f4
```

Listing 3.1: Definition of the cnn network

```
x = tf.placeholder(tf.float32, [None, IMG_SIZE*IMG_SIZE*NUM_CHANNELS])
y_ = tf.placeholder(tf.float32, [None, NUM_CLASSES])

conv_1, pool_1, conv_2, pool_2, logits = cnn(x, 50, 60)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_, logits
   =logits)
loss = tf.reduce_mean(cross_entropy)
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(y_, 1))
correct_prediction = tf.cast(correct_prediction, tf.float32)
accuracy = tf.reduce_mean(correct_prediction)
```

Listing 3.2: Loss minimisation and accuracy prediction

Additionally the handling of batches sizes are shown in Listing 3.3.

```
# Handle in batches
for start, end in zip(range(0, len(train_X), batch_size), range(batch_size,
    len(train_X), batch_size)):
     _, batch_cost = sess.run([train_step, loss], {x: train_X[start:end], y_:
     train_Y[start:end]})
     train_cost_.append(batch_cost)
```

Listing 3.3: Batch size setup

Finally, the training results used in this experiment is assumed to be the ones obtained from cross-validation obtained from Listing **??**, while the test set used are assumed to be the test set obtained from the initial 70:30 split from the original data.

3.1.1 Question 1

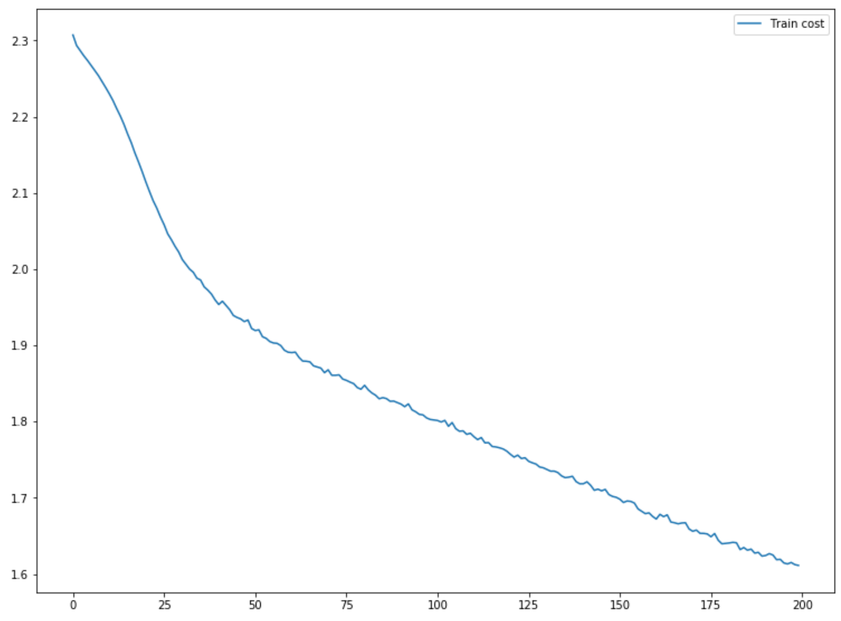
Train the network by using mini-batch gradient descent learning. Set $batchsize = 128$, and learning rate $\alpha = 0.001$. Images should be scaled.

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

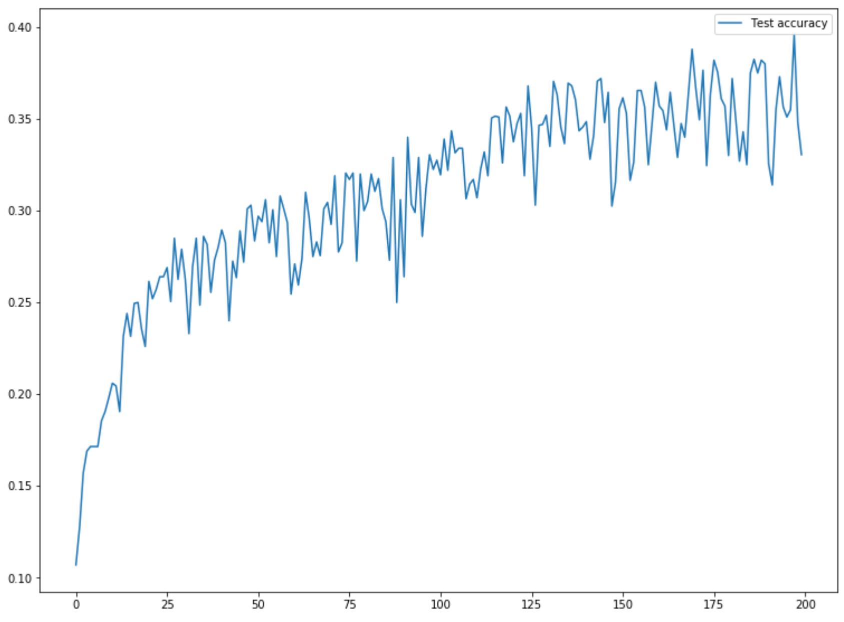
b. For any two test patterns, plot the feature maps at both convolution layers (C_1 and C_2) and pooling layers (S_1 and S_2) along with the test patterns.

Part A

The following plot in Figure 3.8 is obtained for Q1a.



(a) Training cost against epochs for cnn



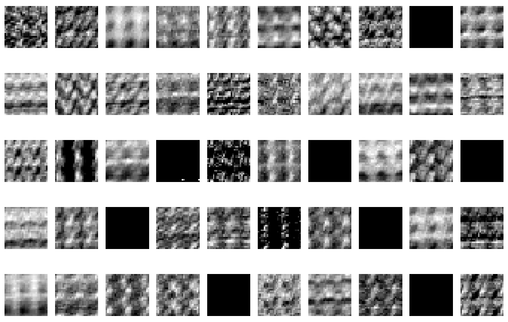
(b) Test accuracy against epochs for cnn

Figure 3.1: Plot of training cost and testing accuracy against epochs for CNN object recognition

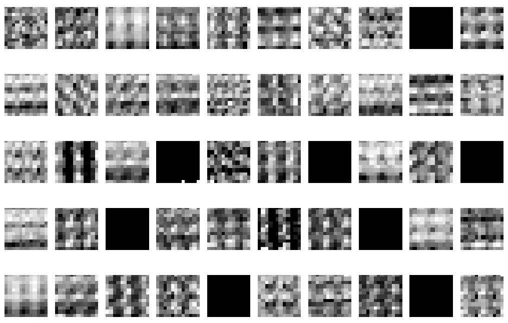
Part B

Test Pattern 1 is shown in Figure 3.2.

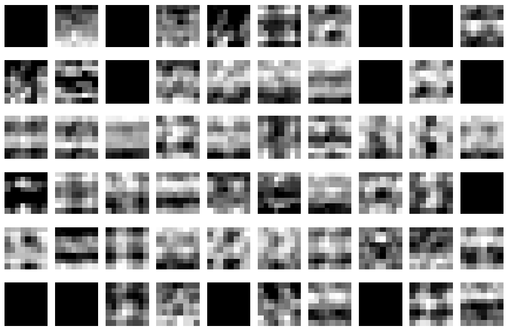
(a) Test Pattern



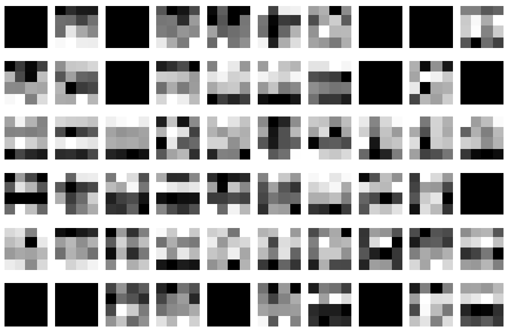
(b) C_1 Feature Map



(c) S_1 Feature Map



(d) C_2 Feature Map

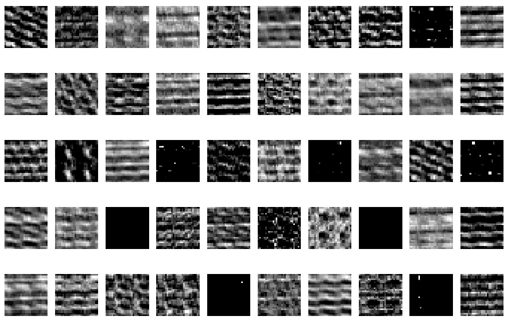


(e) S_2 Feature Map

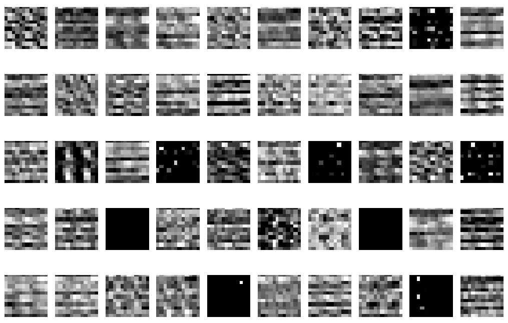
Figure 3.2: Feature maps of test pattern 1

Test Pattern 2 is shown in Figure 3.3.

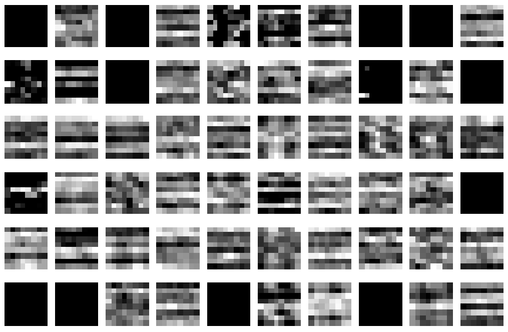
(a) Test Pattern



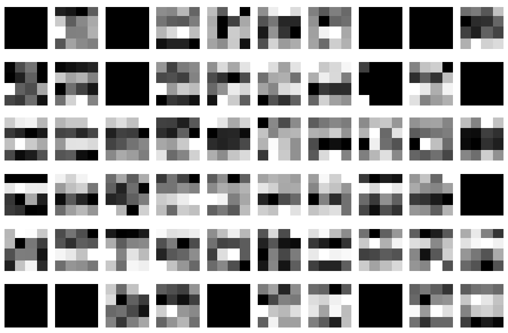
(b) C_1 Feature Map



(c) S_1 Feature Map



(d) C_2 Feature Map



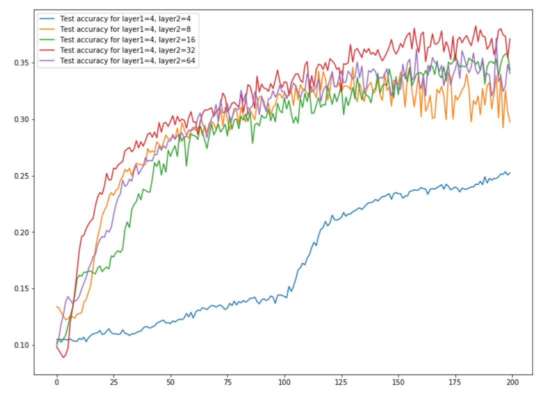
(e) S_2 Feature Map

Figure 3.3: Feature maps of test pattern 2

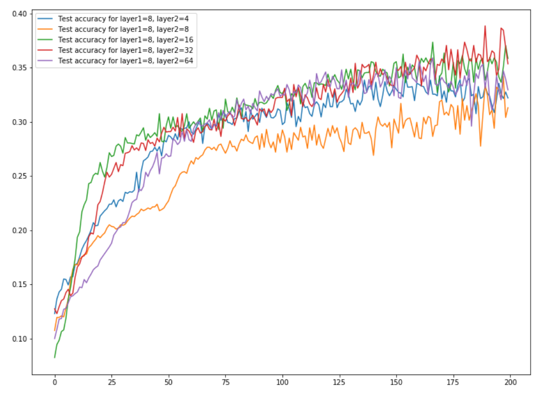
3.1.2 Question 2

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

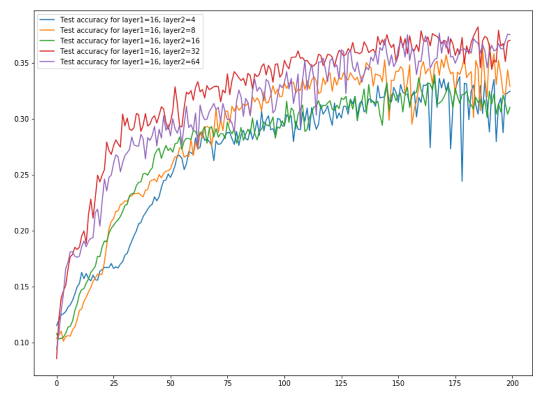
Part A



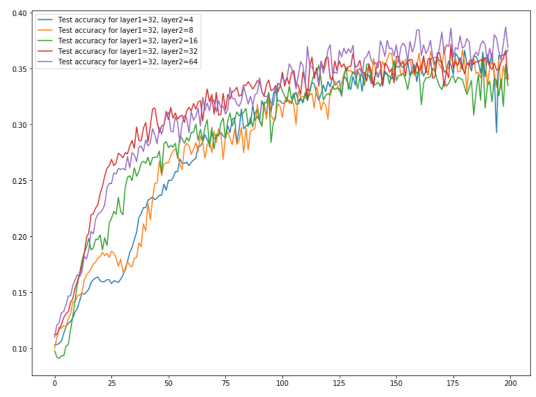
(a) Training cost for cnn



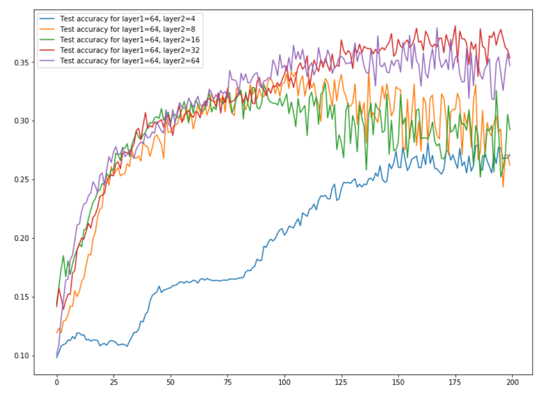
(b) Training cost for cnn



(c) Test accuracy for cnn



(d) Test accuracy for cnn



(e) Test accuracy for cnn

Figure 3.4: Plot of testing accuracy against epochs for different number of feature maps

From the results as shown in Figure 3.4d, having 32 feature maps in convolution layer 1 and 64 feature maps in convolution layer 2 resulted in the highest accuracy among all the others.

3.1.3 Question 3

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

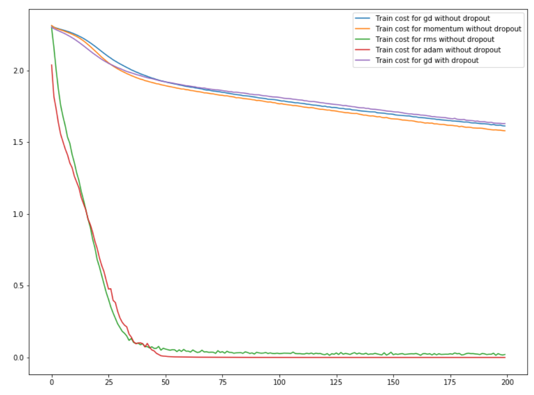
a. Adding the momentum term with momentum $\gamma = 0.1$.

b. Using RMSProp algorithm for learning

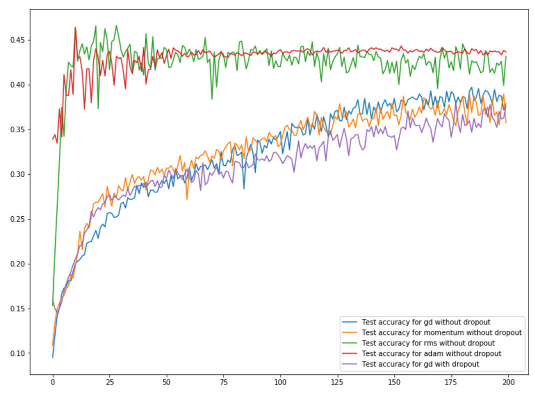
c. Using Adam optimizer for learning

d. Adding dropout to the layers

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



(a) Training cost for CNN



(b) Testing accuracy for CNN

Figure 3.5: Plot of training cost and testing accuracy against epochs for different optimizers

3.1.4 Question 4

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

For this experiment, we compared the performances against the original network without dropout. For network with momentum added, the accuracy is similar to the original network but its has a better learning rate as shown as the gradient being steeper. This is due to the fact that momentum causes the algorithm converges faster because oscillations are kept in one direction hence allowing a higher learning rate. Also, the RMSProp and Adam optimizers converges a lot faster in Figure 3.5a and has higher accuracy in Figure 3.5a than the other networks using other optimizers.

For the network using RMSProp, it has significantly better accuracy and performance compared to just adding momentum. It has a very high learning rate and it takes a larger step towards the minima. This is because it restricts the window of past gradients that are accumulated to be some fixed size. This ensures that learning continues to make progress even after many iterations of updates have been done. It uses an exponentially decaying average to discard the history from extreme past and this causes the algorithm to converge at a faster rate.

The network using the Adam Optimizer has the best performance because it uses an adaptive learning rate method. Learning rates are stored for every parameter in the Adam Optimizer and it uses the squared gradients to scale the learning rate and the decaying average of the gradient instead of gradient itself. This resulted in high learning rate similar to when RMSProp is used and it has less fluctuations in the accuracy as reflected in Figure 3.5.

3.2 Part B: Text classification of Wikipage entries

The details of the dataset are given as follows:

> The dataset used in this project contains the first paragraphs collected from Wikipage entries and the corresponding labels about their category. You will implement CNN and RNN layers at the word and character levels for the classification of texts in the paragraphs. The output layer of the networks is a softmax layer. The training and test datasets will be read from 'train_medium.csv and 'test_medium.csv' files. The training dataset contains 5600 entries and test dataset contains 700 entries. The label of an entry is one of the 15 categories such as people, company, schools, etc. The input data is in text, which should be converted to character/word IDs to feed to the networks by using 'tf.contrib.learn.preprocessing'. Restrict the maximum length of the characters/word inputs to 100. Use the Adam optimizer for training with a batch size = 128 and learning rate = 0.01. Assume there are 256 different characters.

The character and word CNN is defined as in Listing 3.4, while the char and word RNN is defined as in Listing 3.5, with the loss and accuracy functions defined in Listing 3.6.

```python
def cnn_model(x, model_type, dropout=False):
    if model_type == 'char':
        input_layer = tf.reshape(
            tf.one_hot(x, 256), [-1, MAX_DOCUMENT_LENGTH, 256, 1])

        # if dropout:
        #     input_layer = tf.layers.dropout(input_layer, 0.1)

        with tf.variable_scope('Char_CNN_Layer1'):
            conv1 = tf.layers.conv2d(
                input_layer,
                filters=N_FILTERS,
                kernel_size=FILTER_SHAPE1,
                padding='VALID',
                activation=tf.nn.relu)
            pool1 = tf.layers.max_pooling2d(
                conv1,
                pool_size=POOLING_WINDOW,
                strides=POOLING_STRIDE,
                padding='SAME')
            if dropout:
                pool1 = tf.layers.dropout(pool1, 0.25)
        with tf.variable_scope('Char_CNN_Layer2'):
            conv2 = tf.layers.conv2d(
                pool1,
                filters=N_FILTERS,
                kernel_size=FILTER_SHAPE2,
                padding='VALID',
                activation=tf.nn.relu)
            pool2 = tf.layers.max_pooling2d(
                conv2,
                pool_size=POOLING_WINDOW,
                strides=POOLING_STRIDE,
                padding='SAME')
            if dropout:
                pool2 = tf.layers.dropout(pool2, 0.25)

        pool2 = tf.squeeze(tf.reduce_max(pool2, 1), squeeze_dims=[1])

        logits = tf.layers.dense(pool2, MAX_LABEL, activation=None)

        # if dropout:
        #     logits = tf.layers.dropout(logits, 0.5)

        return input_layer, logits

    elif model_type == 'word':
        word_vectors = tf.contrib.layers.embed_sequence(
        x, vocab_size=n_words, embed_dim=EMBEDDING_SIZE)

        word_list = tf.unstack(word_vectors, axis=1)

        input_layer = tf.reshape(
            word_vectors, [-1, MAX_DOCUMENT_LENGTH, HIDDEN_SIZE, 1])
        # if dropout:
        #     input_layer = tf.layers.dropout(input_layer, 0.1)

        with tf.variable_scope('Word_CNN_Layer1'):
            conv1 = tf.layers.conv2d(
                input_layer,
                filters=N_FILTERS,
                kernel_size=[20, 20],
                padding='VALID',
                activation=tf.nn.relu)
            pool1 = tf.layers.max_pooling2d(
                conv1,
                pool_size=POOLING_WINDOW,
                strides=POOLING_STRIDE,
                padding='SAME')
            if dropout:
                pool1 = tf.layers.dropout(pool1, 0.25)
        with tf.variable_scope('Word_CNN_Layer2'):
            conv2 = tf.layers.conv2d(
                pool1,
                filters=N_FILTERS,
                kernel_size=FILTER_SHAPE2,
                padding='VALID',
                activation=tf.nn.relu)
            pool2 = tf.layers.max_pooling2d(
                conv2,
                pool_size=POOLING_WINDOW,
                strides=POOLING_STRIDE,
                padding='SAME')
            if dropout:
                pool2 = tf.layers.dropout(pool2, 0.25)

        pool2 = tf.squeeze(tf.reduce_max(pool2, 1), squeeze_dims=[1])

        logits = tf.layers.dense(pool2, MAX_LABEL, activation=None)
```

```python

        # if dropout:
        #     logits = tf.layers.dropout(logits, 0.5)

        return input_layer, logits
    else:
        raise Exception(f'No model type named {model_type}')
```
Listing 3.4: Character and word convolutional neural network (layers controlled by layers parameter)

```python
def rnn_model(x, model_type, cell_type='GRU', num_layers=1, dropout=False):
    if model_type == 'char':
        input_layer = tf.one_hot(x, 256)
        input_layer = tf.unstack(input_layer, axis=1)

        cells_ = []
        for i in range(num_layers):
            # Define cell type
            if cell_type == 'GRU':
                cell = tf.nn.rnn_cell.GRUCell(HIDDEN_SIZE, reuse=tf.get_variable_scope().reuse)
            elif cell_type == 'VANILLA':
                cell = tf.nn.rnn_cell.BasicRNNCell(HIDDEN_SIZE, reuse=tf.get_variable_scope().reuse)
            elif cell_type == 'LSTM':
                cell = tf.nn.rnn_cell.LSTMCell(HIDDEN_SIZE, reuse=tf.get_variable_scope().reuse)
            else:
                raise Exception(f'No cell type matches {cell_type}')

            # Define dropout
            if dropout:
                cell = tf.nn.rnn_cell.DropoutWrapper(cell, output_keep_prob=0.5)

            cells_.append(cell)

        # Multi-layer rnn cells
        cells = tf.nn.rnn_cell.MultiRNNCell(cells_)
        _, encoding = tf.nn.static_rnn(cells, input_layer, dtype=tf.float32)

        encoding = np.array(encoding).flatten()

        # Dense layer
        logits = tf.layers.dense(encoding[-1], MAX_LABEL, activation=None)
        # if dropout:
        #     logits = tf.layers.dropout(logits, 0.5)

        return input_layer, logits

    elif model_type == 'word':
        word_vectors = tf.contrib.layers.embed_sequence(
            x, vocab_size=n_words, embed_dim=EMBEDDING_SIZE)

        word_list = tf.unstack(word_vectors, axis=1)

        cells_ = []
        for i in range(num_layers):
            # Define cell type
            if cell_type == 'GRU':
                cell = tf.nn.rnn_cell.GRUCell(HIDDEN_SIZE, reuse=tf.get_variable_scope().reuse)
            elif cell_type == 'VANILLA':
                cell = tf.nn.rnn_cell.BasicRNNCell(HIDDEN_SIZE, reuse=tf.get_variable_scope().reuse)
            elif cell_type == 'LSTM':
                cell = tf.nn.rnn_cell.LSTMCell(HIDDEN_SIZE, reuse=tf.get_variable_scope().reuse)
            else:
                raise Exception(f'No cell type matches {cell_type}')

            # Define dropout
            if dropout:
                cell = tf.nn.rnn_cell.DropoutWrapper(cell, output_keep_prob=0.5)

            cells_.append(cell)

        # Multi-layer rnn cells
        cells = tf.nn.rnn_cell.MultiRNNCell(cells_)
        _, encoding = tf.nn.static_rnn(cells, word_list, dtype=tf.float32)

        encoding = np.array(encoding).flatten()

        # Dense layer
        logits = tf.layers.dense(encoding[-1], MAX_LABEL, activation=None)

        # if dropout:
        #     logits = tf.layers.dropout(logits, 0.5)

        return word_list, logits
    else:
        raise Exception(f'No model type named {model_type}')
```

Listing 3.5: Character and word recurrent neural network neural network (layers controlled by layers parameter)

```python
def create_model(feature_size, neuron_size, weight_decay_beta, learning_rate, layers=3, dropout=False):
# Create the model
x = tf.placeholder(tf.int64, [None, MAX_DOCUMENT_LENGTH])
y_ = tf.placeholder(tf.int64)

inputs, logits = rnn_model(x, model_type='char', cell_type=item['cell_type'], num_layers=item['num_layers'])

# Optimizer
entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=tf.one_hot(y_, MAX_LABEL), logits=logits))

# Gradient clipping
if item['grad_clip']:
    minimizer = tf.train.AdamOptimizer()
    grads_and_vars = minimizer.compute_gradients(entropy)
    grad_clipping = tf.constant(2.0, name="grad_clipping")
    clipped_grads_and_vars = []
    for grad, var in grads_and_vars:
        clipped_grad = tf.clip_by_value(grad, -grad_clipping, grad_clipping)
        clipped_grads_and_vars.append((clipped_grad, var))
    train_op = minimizer.apply_gradients(clipped_grads_and_vars)
else:
    train_op = tf.train.AdamOptimizer(lr).minimize(entropy)

correct_prediction = tf.equal(tf.argmax(logits, 1), y_)
correct_prediction = tf.cast(correct_prediction, tf.float32)
accuracy = tf.reduce_mean(correct_prediction)
```

Listing 3.6: loss and accuracy functions

Mini-batch gradient descend is done through Listing 3.7.

```
for start, end in zip(range(0, len(train_X), batch_size), range(batch_size,
   len(train_X), batch_size)):
    _, batch_cost = sess.run([train_step, loss], {x: train_X[start:end], y_:
    train_Y[start:end]})
    train_cost_.append(batch_cost)
```

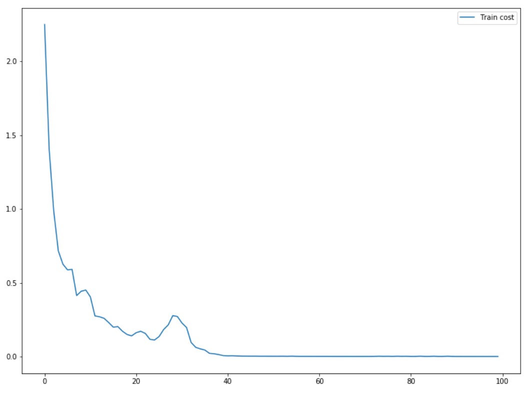
Listing 3.7: Mini-batch setup

3.2.1 Question 1

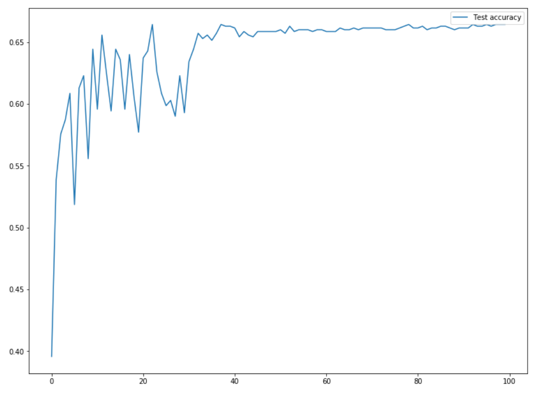
Design a Character CNN Classifier that receives character ids and classifies the input. The CNN has two convolution and pooling layers:

- A convolution layer C_1 of 10 filters of window size 20x256, VALID padding, and ReLU neurons. A max pooling layer S_1 with a pooling window of size 4x4, with stride = 2, and padding = 'SAME'.
- A convolution layer C_2 of 10 filters of window size 20x1, VALID padding, and ReLU neurons. A max pooling layer S_2 with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'.

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



(a) Training cost against epochs for CNN



(b) Test accuracy against epochs for CNN

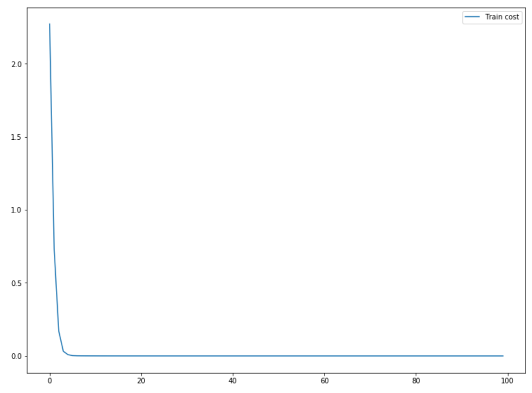
Figure 3.6: Plot of training cost and testing accuracy against epochs for character CNN

3.2.2 Question 2

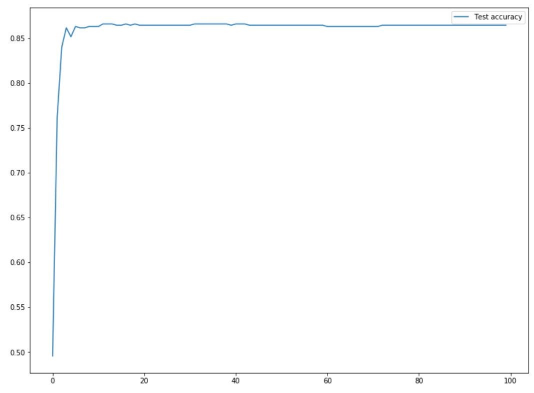
Design a Word CNN Classifier that receives word ids and classifies the input. Pass the inputs through an embedding layer of size 20 before feeding to the CNN. The CNN has two convolution and pooling layers with the following characteristics:

- A convolution layer C_1 of 10 filters of window size 20x20, VALID padding, and ReLU neurons. A max pooling layer S_1 with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'.
- A convolution layer C_2 of 10 filters of window size 20x1, , VALID padding, and ReLU neurons. A max pooling layer S_2 with a pooling window of size 4x4, with stride = 2 and padding = 'SAME'.

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



(a) Training cost against epochs for CNN

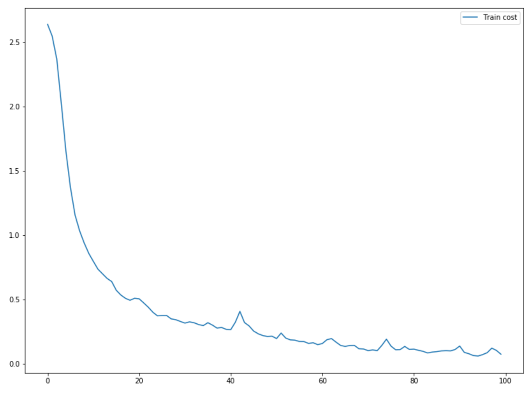


(b) Test accuracy against epochs for CNN

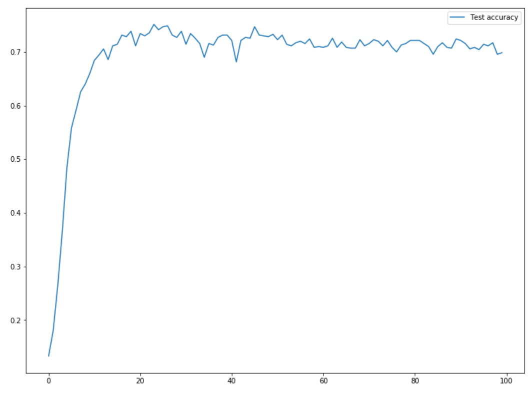
Figure 3.7: Plot of training cost and testing accuracy against epochs for word CNN

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



(a) Training cost against epochs for RNN

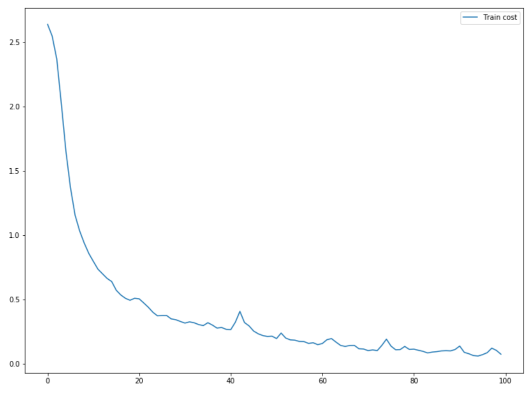


(b) Test accuracy against epochs for RNN

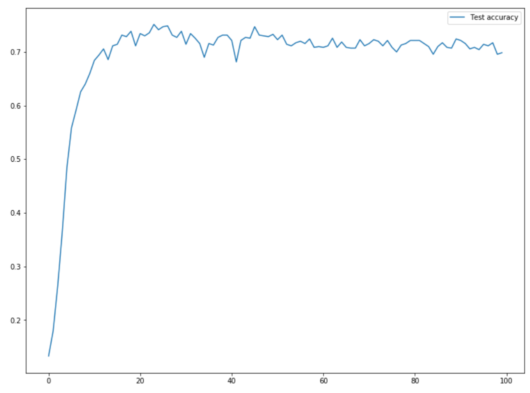
Figure 3.8: Plot of training cost and testing accuracy against epochs for Character RNN

3.2.4 Question 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. Plot the entropy on training data and the accuracy on testing data versus training epochs.



(a) Training cost against epochs for RNN

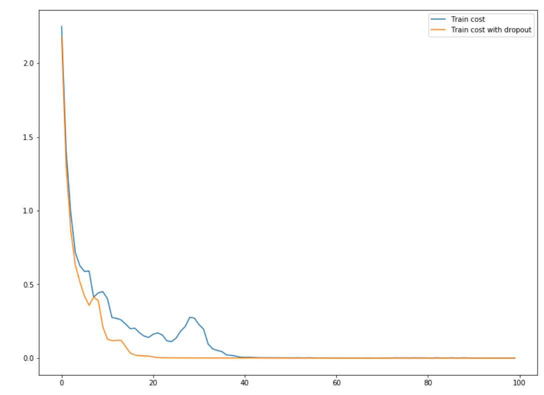


(b) Test accuracy against epochs for RNN

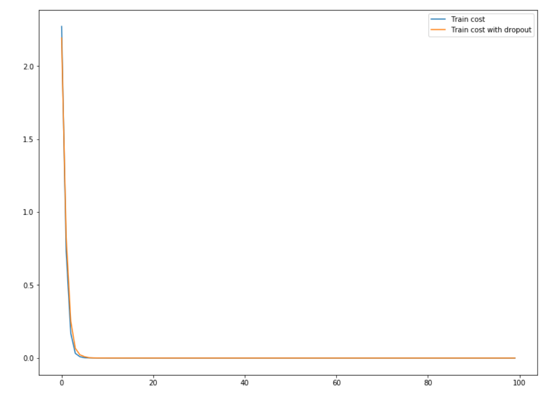
Figure 3.9: Plot of training cost and testing accuracy against epochs for Word RNN

3.2.5 Question 5

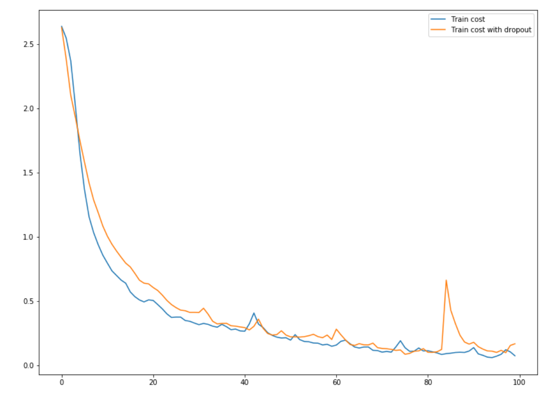
Compare the test accuracies and the running times of the networks implemented in parts (1) – (4). Experiment with adding dropout to the layers of networks in parts (1) – (4), and report the test accuracies. Compare and comment on the accuracies of the networks with/without dropout.



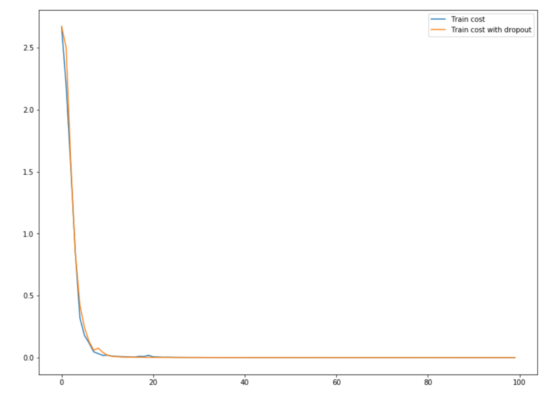
(a) Training cost against epochs for Character CNN



(b) Training cost against epochs for Word CNN

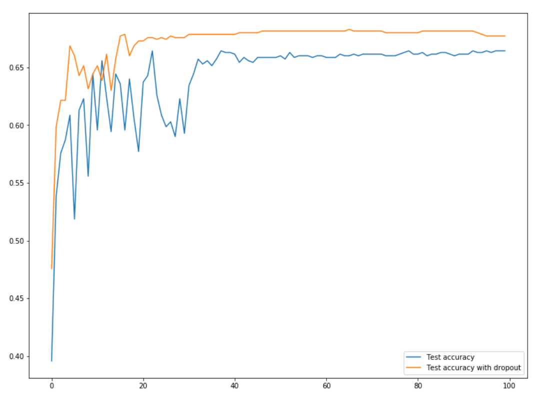


(c) Training cost against epochs for Character RNN

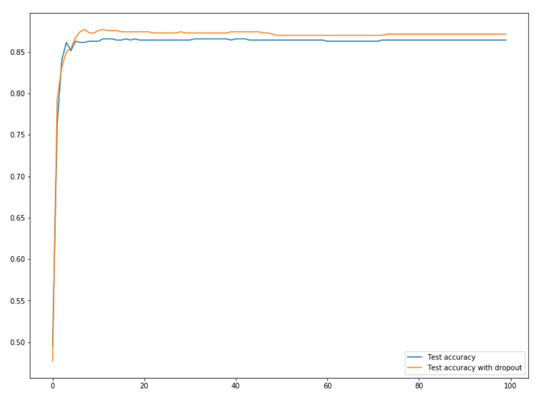


(d) Training cost against epochs for Word RNN

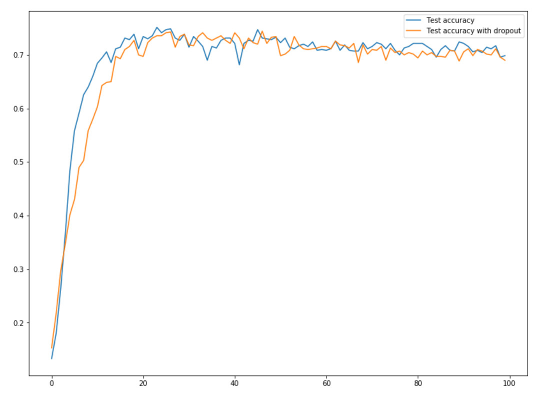
Figure 3.10: Dropout training cost comparisons for Charcter and Word classification for CNN and RNN



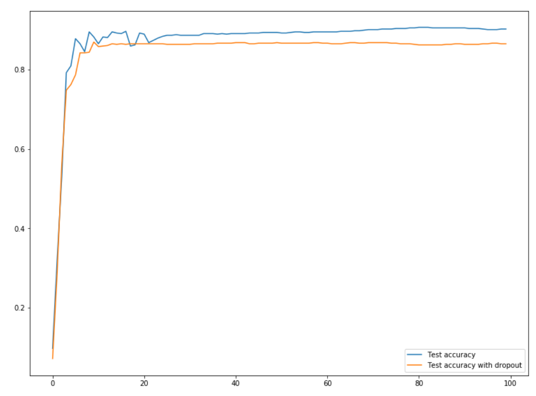
(a) Test accuracy against epochs for Character CNN



(b) Test accuracy against epochs for Word CNN



(c) Test accuracy against epochs for Character RNN



(d) Test accuracy against epochs for Word RNN

Figure 3.11: Dropout test accuracy comparisons for Charcter and Word classification for CNN and RNN

We initially experimented with a dropout rate of (0.1, 0.25, 0.25, 0.5), similar to the dropouts used in [2] but achieved accuracy lesser with dropout. This is due to the fact that placing drop out at the start reduces the chances of epochs being activated, cutting down the number of test epochs. Placing drop out at the end, we are not able to identify and correct errors caused by the dropout that was implemented before, hence lowering accuracy. So we removed the input and output dropouts, retaining only a dropout rate of (0.25, 0.25) for CNN layers, which allowed us to get better results. While applying dropout to CNN, it is also said that for CNN dropout ”still helps because it provides noisy inputs for the higher fully connected layers which prevents them from overfitting. ” in [2]. This finding is consistent with our experiment in Figure 3.11.

char CNN	word CNN	char RNN	word RNN
45.56	25.04	202.97	265.86

Table 3.1: Network running times

The run times for the different networks can be found in Table 3.1. It can be seen that for char classification, the CNN network is around 5 times faster than the RNN network, while for the word

CNN network, it is more than 10x faster than the word RNN network. However, the accuracy of the word CNN network is about 10% more than the accuracy of the word RNN network as well, while the accuracy of the char CNN network is similar to the accuracy of the char RNN network.

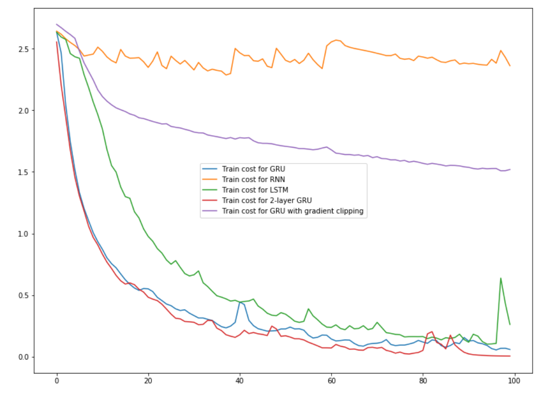
3.2.6 Question 6

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:

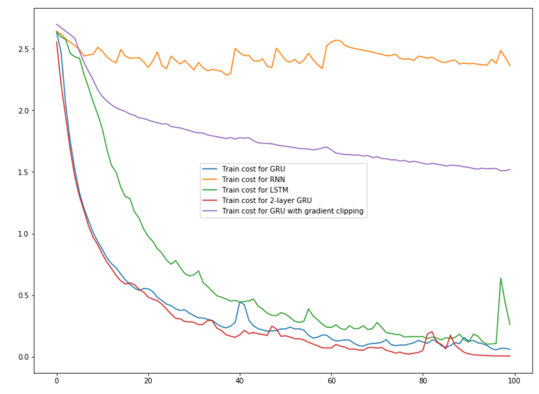
a. Replace the GRU layer with (i) a vanilla RNN layer and (ii) a LSTM layer

b. Increase the number of RNN layers to 2 layers

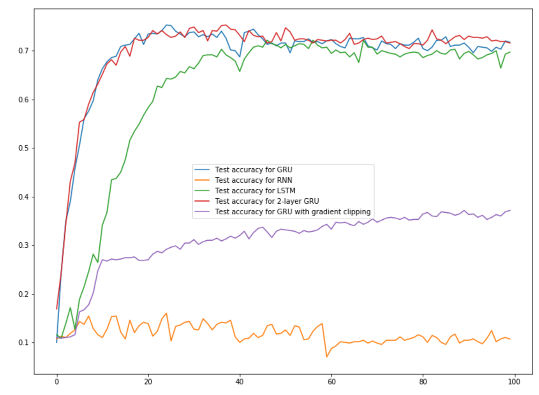
c. Add gradient clipping to RNN training with clipping threshold = 2.



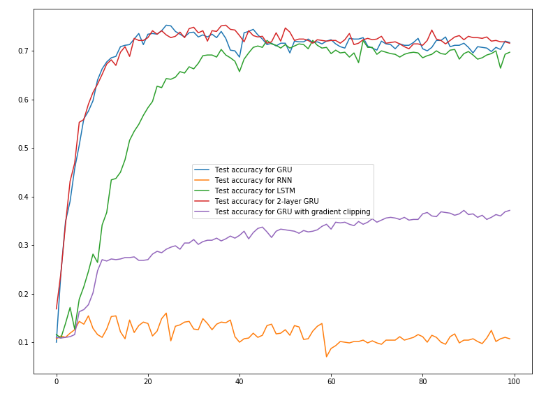
(a) Training cost for Character RNN



(b) Test accuracy cost for Character RNN



(c) Training cost for Word RNN



(d) Test accuracy cost for Word RNN

Figure 3.12: Plot of training cost and testing accuracy against epochs for different layer type, layers and gradient clipping

For this experiment, we are using the original GRU layer (blue in Figure 3.12) as a benchmark/ control for result comparison.

(a) (i) Having a vanilla RNN layer cause an extreme drop in accuracy and performance. This is because with each step, the content is overwritten. This makes it harder for the RNN to predict future outcomes based on past memory which will result in wrong predictions, thus having the worst performance.

   (ii) The LSTM layer has similar performance as the original GRU layer. This is due to the fact that having a LSTM layer allows the ability to retain or forget information, creating a control flow within the network. It allows for detection of an important input feature and carry it over long distances, thus it is capable of learning long-term dependencies. This reduces the chance of exploding gradient problems which is why it has similar performances as a GRU layer.

(b) Having the number of GRU layers increased to 2 resulted in better performance of the network as compared to using a single layer because having more layers allows for training of more complex arbitrary functions, thus doing a better job at predicting future outcomes resulting in better performance. This, as can be seen in Figure 3.12, it had the best performance out of all the other setups.

(c) Gradient clipping forces the gradient values to specific maximum and minimum values if the gradient exceeds the given range. This method addresses the numerical stability of training deep neural network models and does not offer any general improvement in performance. The network converges slower but it can reach the accuracy of the other networks when given enough epoches. Since the gradient is limited within a range of 2.0, the results will also be limited thus it may not be as ideal as compared to the other networks, hence a drop in performance. However, on running a much higher number of epoches, the accuracy still increases to about the same accuracy as the other setups, which means that the introduction of gradient clipping actually greatly reduced the rate of convergence for our experiment.

Therefore from the experiment, the vanilla RNN layer has the worst accuracy as compared to the others. Whereas, having 2 GRU layers resulted in the best performance

3.3 Conclusion

In conclusion, we have discussed two problems in this report and experienced the differences in utilizing different optimizers and techniques for a RNN in handling these 2 different problems.

Bibliography

[1] T. Carneiro, R. V. M. Da Nóbrega, T. Nepomuceno, G.-B. Bian, V. H. C. De Albuquerque, and P. P. Reboucas Filho, “Performance analysis of google colaboratory as a tool for accelerating deep learning applications”, *IEEE Access*, vol. 6, pp. 61 677–61 685, 2018.

[2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting”, *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.