NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ4042: Neural Networks and Deep Learning

Assignment 2:

Object Recognition

Text Classification

Lin HuangJiayin

U1721208D

Introduction:

This paper consists of 2 section. The first focuses on object classification, while the second focuses on text classification.

For the first section, the CIFAR-10 dataset will be used. It consists of 32x32 colour images, with 10 class labels. Training is done on 10,000 samples, and testing is done on 2,000 test samples. A 6-layer CNN will be built for object recognition. The performance of the network will be discussed as the number of channels, learning rate and dropout gets introduced.

For the second section, the Graduate Admissions Prediction dataset is being used. It contains 8 input parameters, and 1 predicted parameter. This paper will make use of 7 input parameter, and the predicted parameter. Both CNN and RNN will be built for text recognition. The performance of the network will be discussed as embedded layers, LSTM layers and GRU layers are introduced

Contents

Methods:

Data Pre-processing

Part A

The inputs are RGB colour images of size 32x32. The pixel values are scaled to 0-1 from 0-255.

```
np.array(data, dtype=np.float32) / 255

labels = np.array(labels, dtype=np.int32)
```

Part B

Word: The inputs rows are split into 1 section and the alphabets are converted to lower case. Each unique word is then built into a dictionary, and converted to numpy array

```
def preprocess(contents, word_dict, document_max_len):
    x = list(map(lambda d: word_tokenize(clean_str(d)),contents))
    x = list(map(lambda d: list(map(lambda w: word_dict.get(w, 
word_dict["<unk>"]),d)),x))
    x = list(map(lambda d: d + [word_dict["<eos>"]],x))
    x = list(map(lambda d: d[:document_max_len],x))
    x = list(map(lambda d: d + (document_max_len – len(d)) * 
[word_dict["<pad>"]],x))
```

Character: The inputs rows are split into 1 section and the alphabets are converted to lower case. Each character is mapped to an integer and converted to numpy array

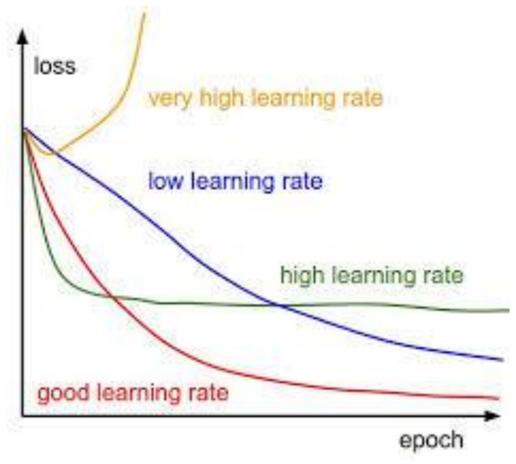
```
def preprocess(strings, char_to_ix, MAX_LENGTH ):
    data_chars = [list(d.lower()) for _, d in enumerate(strings)]
    for i, d in enumerate(data_chars):
        if len(d)>MAX_LENGTH:
            d = d[:MAX_LENGTH]
        elif len(d) < MAX_LENGTH:
            d += [' '] * (MAX_LENGTH - len(d))

    data_ids = np.zeros([len(data_chars), MAX_LENGTH], dtype=np.int64)
    for i in range(len(data_chars)):
        for j in range(MAX_LENGTH):
            data_ids[i, j] = char_to_ix[data_chars[i][j]]

    return np.array(data_ids)
```

Learning rate

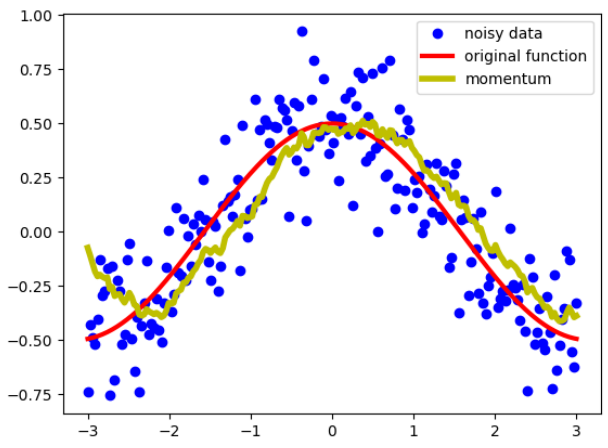
The learning rate indicates the pace at which the weights get updated, which controls the speed of the model adoption to the problem. A smaller learning rate requires more training epochs, vice versa.



Stochastic Gradient Descent

In stochastic gradient descent, a few random samples are selected instead of the whole data set for each iteration.

SGD with Momentum



The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continue to move in their direction. From the graph, the momentum function provides a smoother line to the original function, as compared to the one with noise.

Adaptive Learning rates

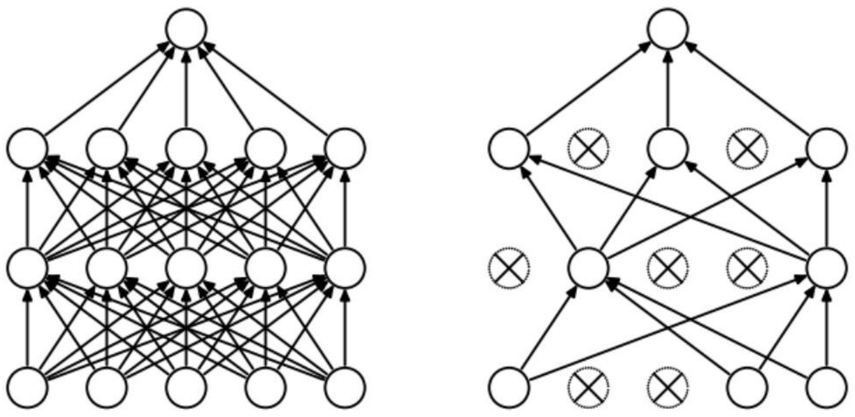
An optimal learning rate will provide the best approximate of the function. However, it is difficult to analytically calculate the optimal learning rate. An adaptive learning rates will be able to decrease the learning rate when the model plateaus and increase the learning rate if performances does not increase. This paper will adopt 2 of the commonly used adaptive learning rates: RMSProp and Adam.

RMSProp

RMSProp optimizer is similar to GD with momentum. However, for RMSProp, the oscillations are restricted in the vertical direction, allowing for faster converges in the horizontal direction. Furthermore, RMSProp can be said as an improvement of AdaGrad, as it discard of history, allowing for rapid converges.

Dropout

Dropouts creates a "thinned network" of neurons that did not get dropped. By reducing the number of neurons, overfitting can be avoided

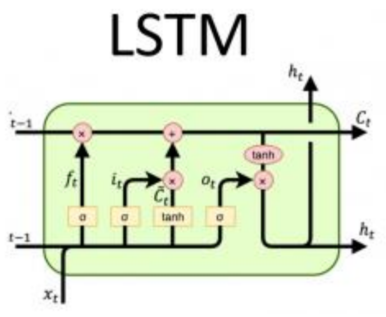

(a) Standard Neural Net (b) After applying dropout.

Word Embedding

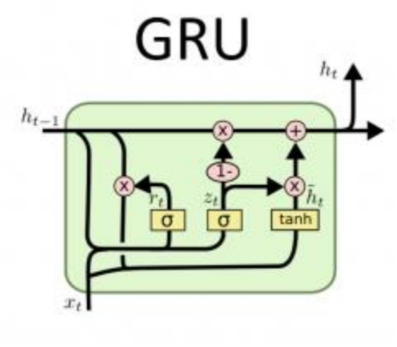
A word embedding is a class of approaches for representing documents and words using a dense vector representation. It is the improvement from the traditional bag-of-word model encoding schemes. For this paper, the embedded layer will be part of the learning model

LSTM

The LSTM layer helps with the issue of short-term memory in RNN, due to the vanishing gradient problem in vanilla RNN. A LSTM consists of forget gate, input gate, cell state, and output gate. The forget gate decides the information to be kept/ thrown. The input gate passes information to update the cell state. The cell state calculates the cell state. The output gate decides that the next hidden state is.



GRU layer

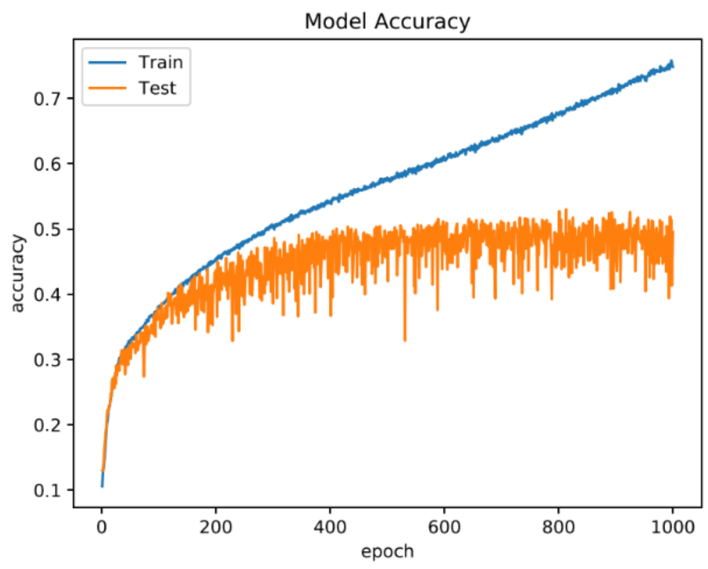


The GRU layer also helps with the issue of short-term memory in RNN, due to the vanishing gradient problem in vanilla RNN. A GRU layer consists of an update gate, and a reset gate. The update gates work similar to the forget and input gate of an LSTM. The reset gate decides how much information to forget.

Essentially, the GRU is like LSTM, but GRU does not use a memory unit.

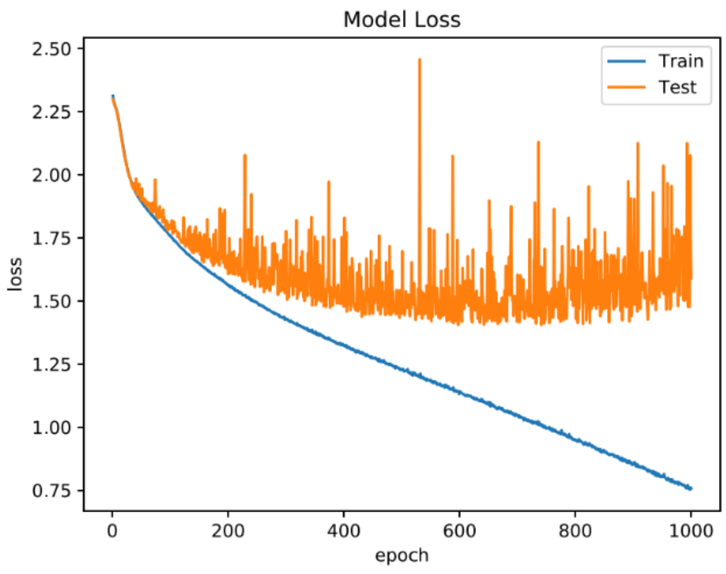
Part A: Object Recognition

1.1. Plot the training cost, test cost, training accuracy and test accuracy against learning epochs



Train accuracy: 74.8%, Test accuracy: 49.3%

The testing accuracy converges around 400 epochs while training accuracy continued to increase in a seemingly linear way.

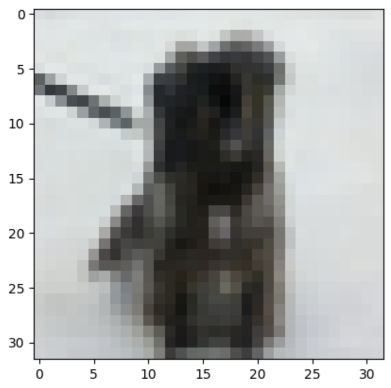


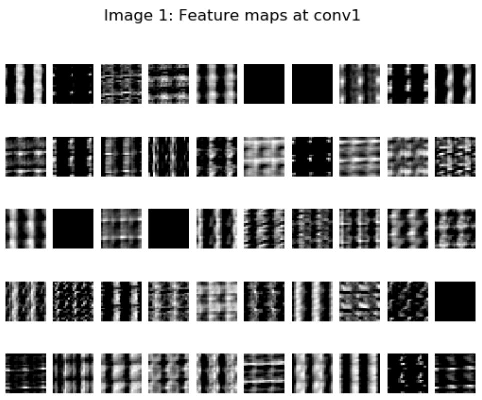
Train loss: 0.756, Test loss: 1.592

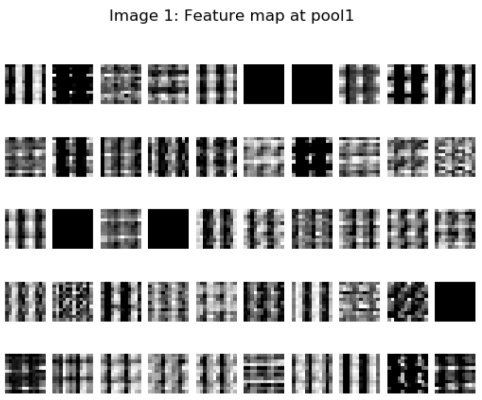
The testing loss increases with the epochs, suggesting that the model has started to overfit.

1.2. Plot the feature maps at both convolution layers and pooling layers along with the test images

1.2.1. First image







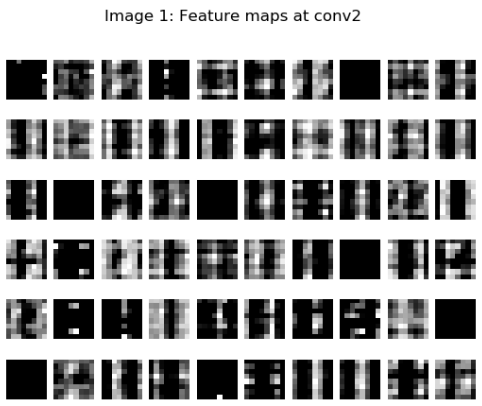


Image 1: Feature map at pool2

1.2.2. Second image

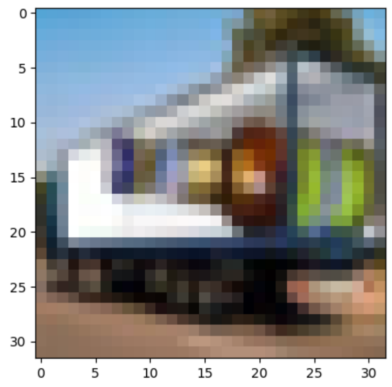


Image 2: Feature maps at conv1

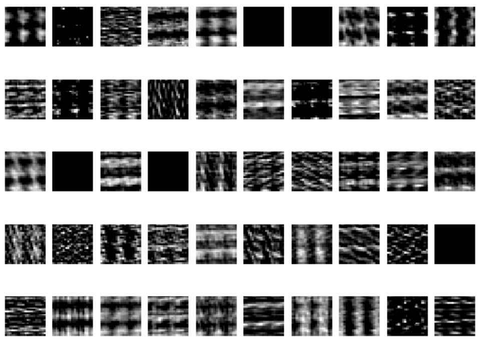


Image 2: Feature map at pool1

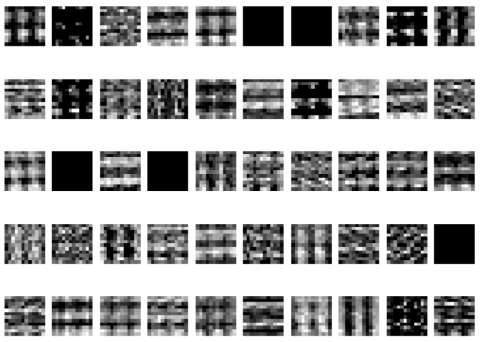


Image 2: Feature maps at conv2

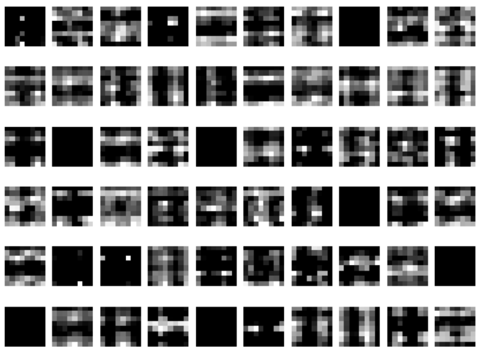
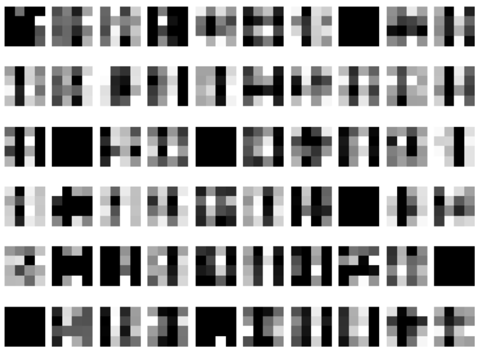


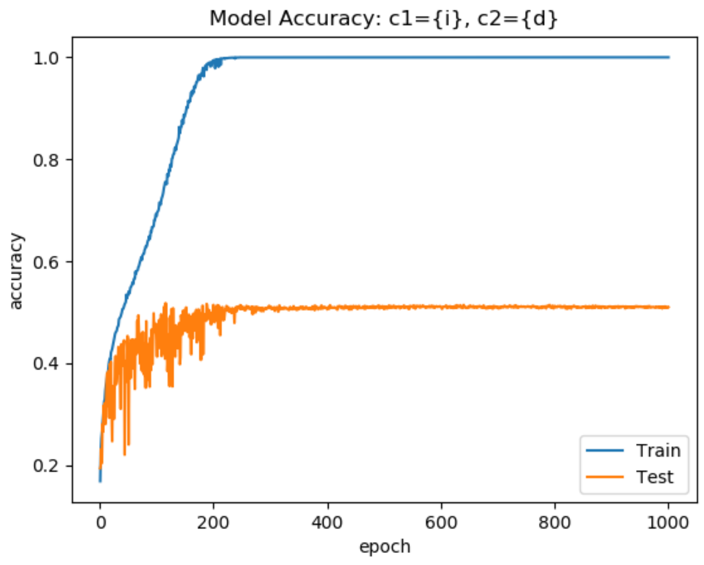
Image 2: Feature map at pool2



2. Find the optimal combinations of the numbers of channels at the convolution layers

num_ch_c1	**num_ch_c2**	**test_acc (%)**
10	20	38.5
10	40	43.3
10	60	42.6
10	80	44.7
10	100	46.4
30	20	45.4
30	40	46.8
30	60	48.4
30	80	48.5
30	100	48.9
50	20	45.6
50	40	47.2
50	60	49.3
50	80	50.0
50	100	50.8
70	20	46.5
70	40	50.5
70	60	50.0
70	80	51.2
70	100	50.8
90	20	47.3
90	40	47.3
90	60	49.3
90	80	51.1
90	100	50.4

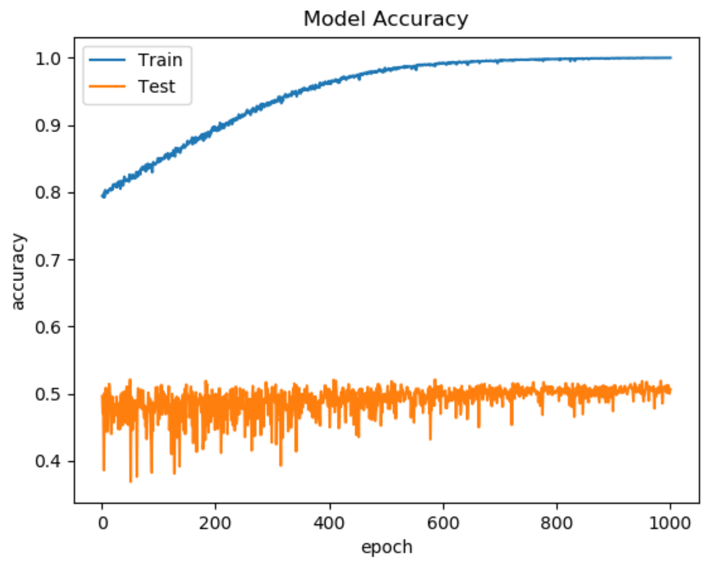
Optimal combination: **c1 with 70 channels, c2 with 80 channels**



Model Accuracy: c1={i}, c2={d}
1.0
0.8
0.6
0.4
0.2
accuracy
0
200
400
600
800
1000
epoch
Train
Test

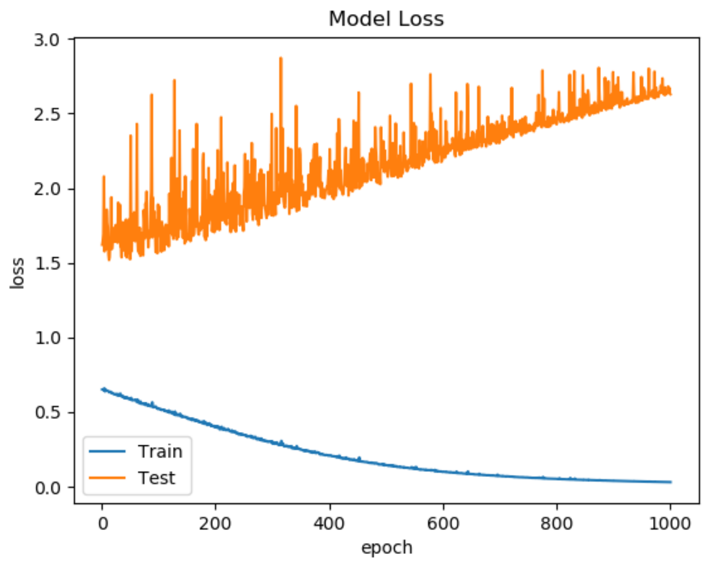
3. Train the network by

3.1. Adding the momentum term with momentum 0.1



Train accuracy: 100% , Test accuracy: 50.6%

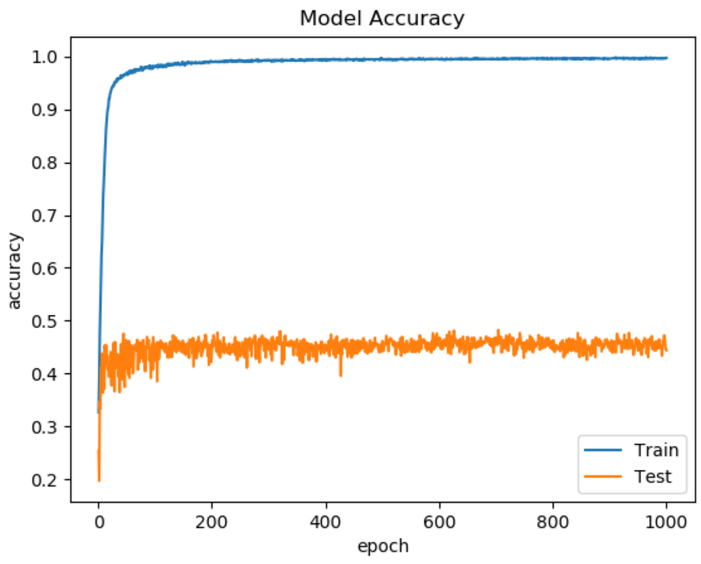
The testing accuracy does not change much since the start, while the training accuracy increases and converge around epoch 700.



Train loss: 0.0311 Test loss: 2.6290

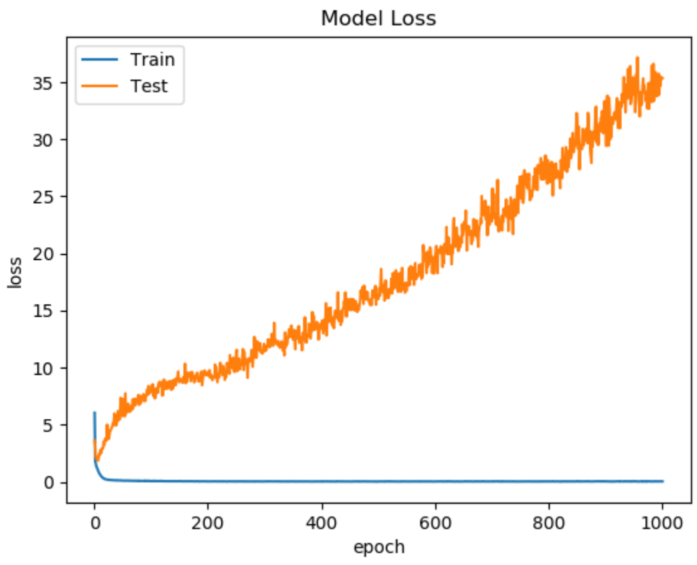
The testing loss increases linearly with the epochs, showing that the model has overfit

3.2. Using RMSProp



Train accuracy: 99.7% , Test accuracy: 44.6%

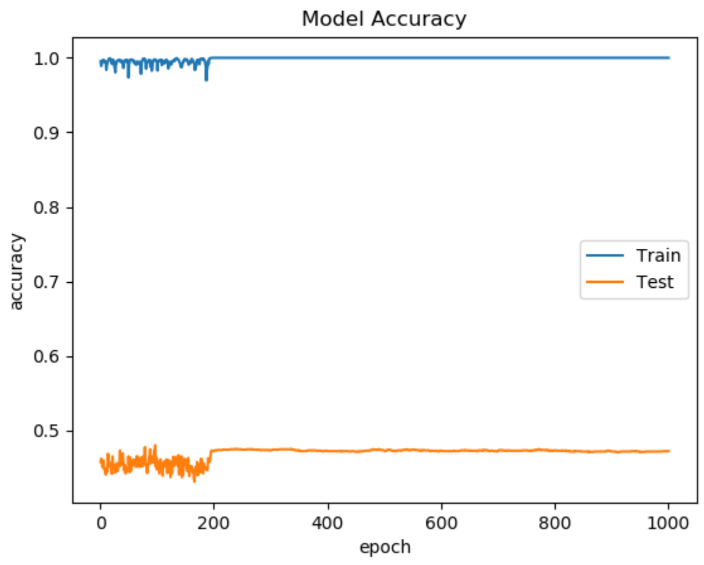
The testing accuracy does not change much since the start, while the training accuracy increases and converge around epoch 100.



Train loss: 0.035 , Test loss: 35.265

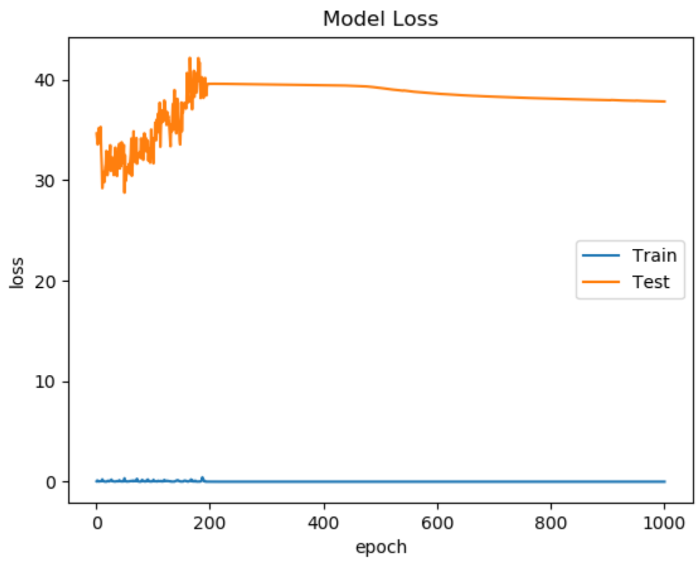
The testing loss increases linearly with the epochs and is way too high. The model has clearly overfit and something have gone wrong

3.3. Adam optimizer



Train accuracy: 100%, Test accuracy: 47.2%

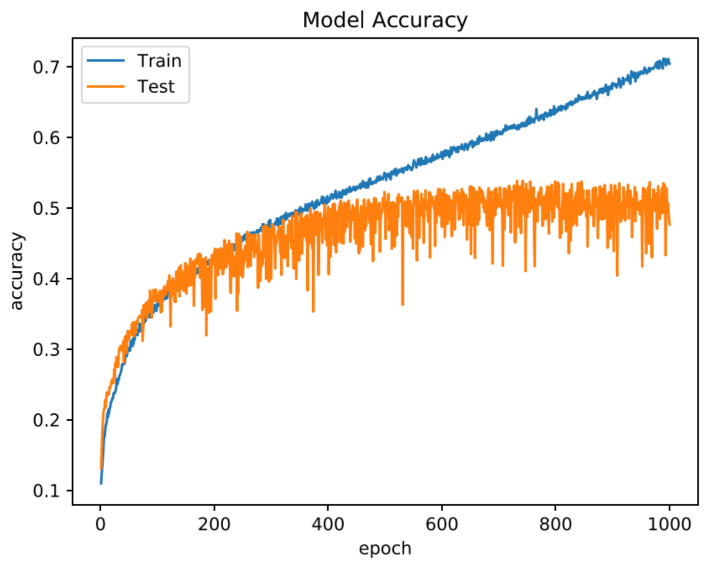
Both testing and training converges around 300 epochs



Train loss: 0.000, Test loss: 37.825

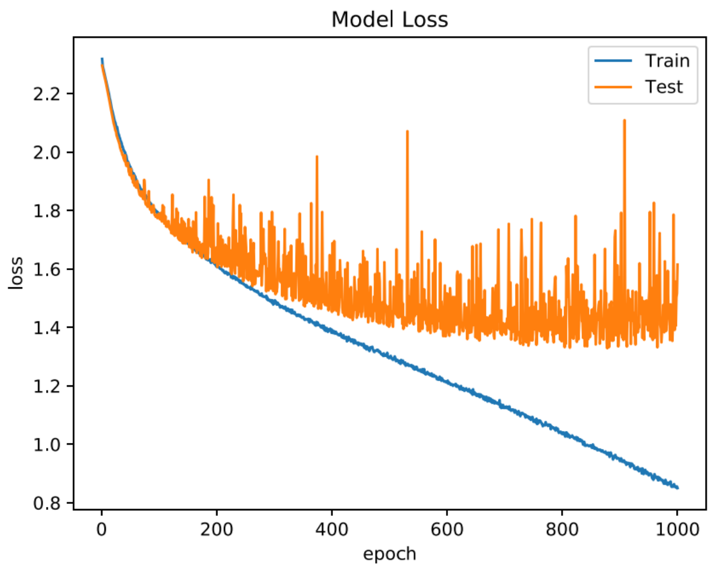
Train loss has been around 0 since the start, while testing loss slowly decreases. Something went really wrong and I am not sure why? Maybe the model overfit really bad, or the optimiser is not suitable, or maybe I did something wrong

3.4. Dropout to the two fully connected layers



Train accuracy: 70.4%, Test accuracy: 47.7%

Training accuracy increases linearly while test accuracy converges around epoch 600

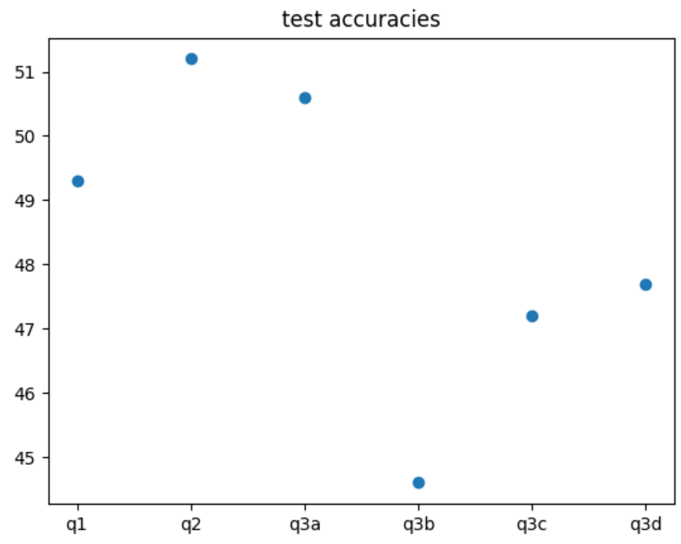


Train loss: 0.850, Test loss: 1.615

Training loss decreases linearly while testing loss decreases gradually. The testing loss starts to increase at the end, indicating that overfitting has occurred.

4. Compare the accuracies of all the models from parts(1) – (3)

Model/ question	1 Initial model	2 Optimal channels	3a Momentum	3b RMSProp	3c Adam	3d dropout
train accuracy	0.748		1.000	0.997	1.000	0.704,
test accuracy	0.493	0.512	0.506	0.446	0.472	0.477
train loss	0.756		0.031	0.035	0.000	0.850
test loss	1.592		2.629	35.27	37.83	1.615

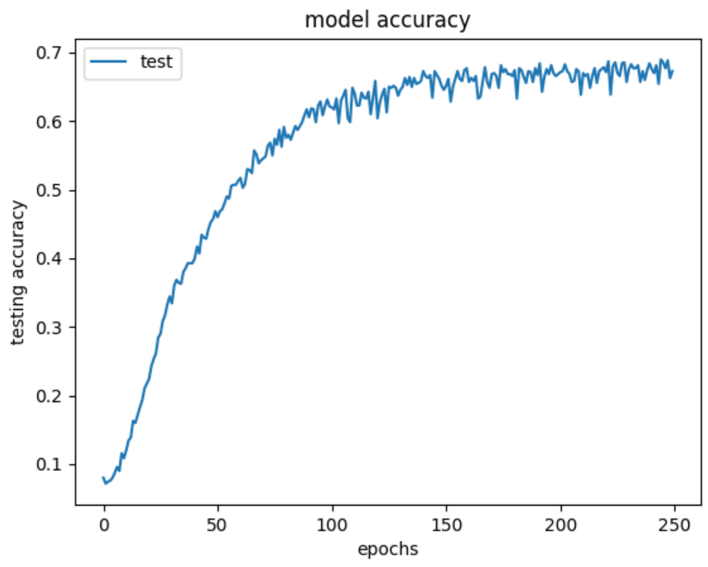


q2 provides the highest testing accuracy, at 0.512, while using SGD by itself. The testing accuracy goes by: q2 > q3a > q1 > q3d > q3c > q3b.

A channel of 70 and 80 with SGD outperforms other optimizers.

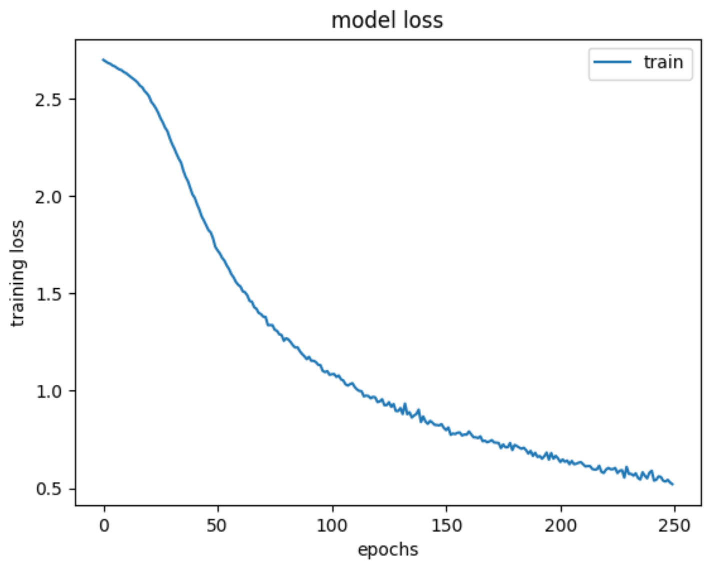
Part B: Text Classification

1. Design a Character CNN



Test accuracy = 67.3%

Testing accuracy increases drastically at the start, and the rate reduced around epoch 100, and result in a gradual convergence

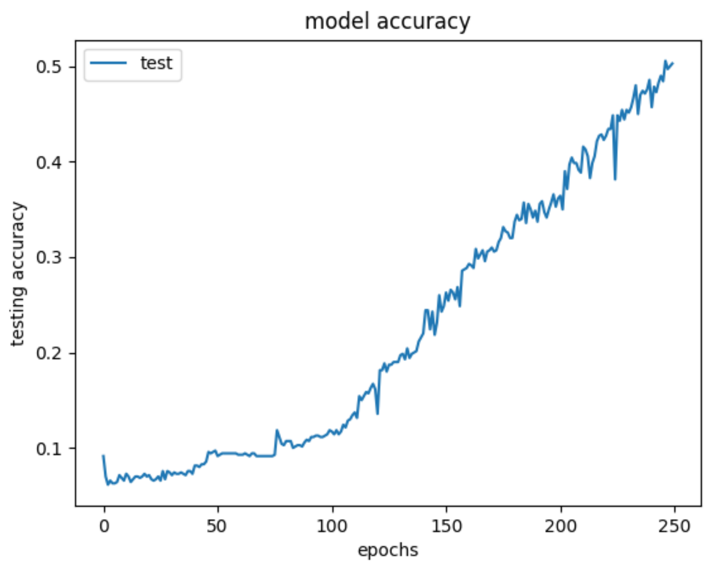


Train loss = 0.520

Training loss decreases drastically at the start, and the rate reduced around epoch 100, result in a almost linear decrease in loss.

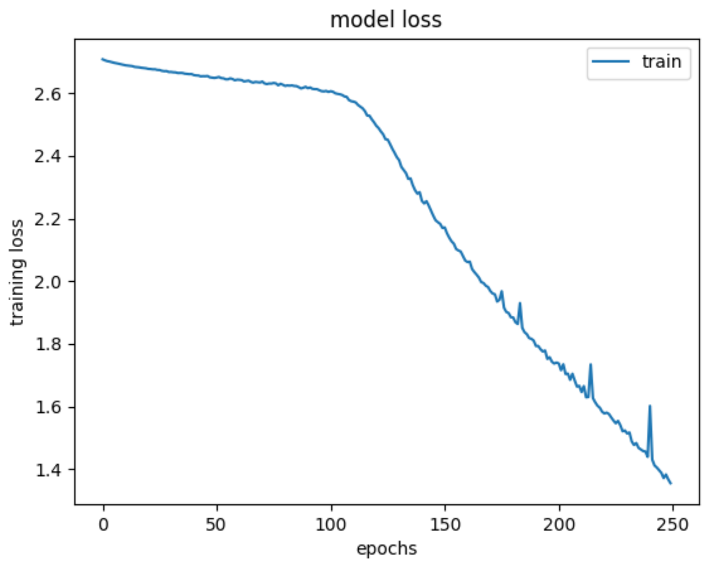
Both test and loss behave similarly, however, as the training loss has yet to converge, it will be good to train for more epochs.

2.Design a Word CNN



Test accuracy = 50.3%

Testing accuracy increases at a greater rate around epoch 100, almost linearly

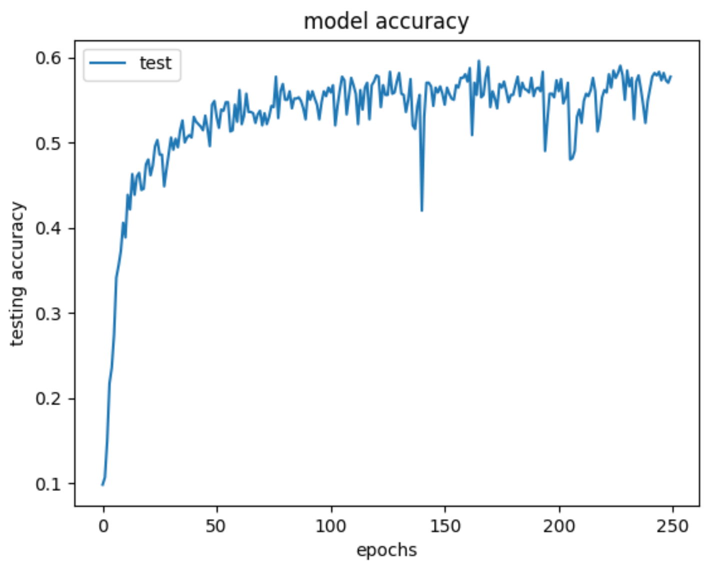


Train loss = 1.353

Training loss decreased gradually till epoch 100, started to decrease at a much higher rate.

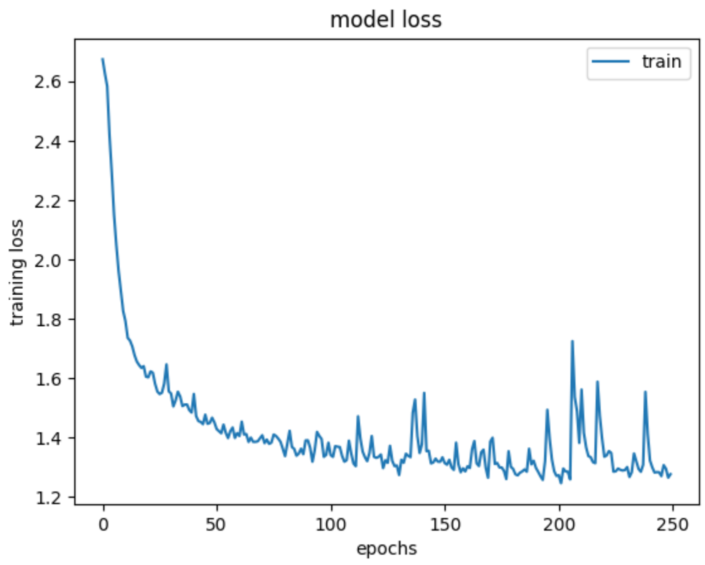
More epochs are required for training

3. Design a character RNN classifier



Test accuracy = 57.7%

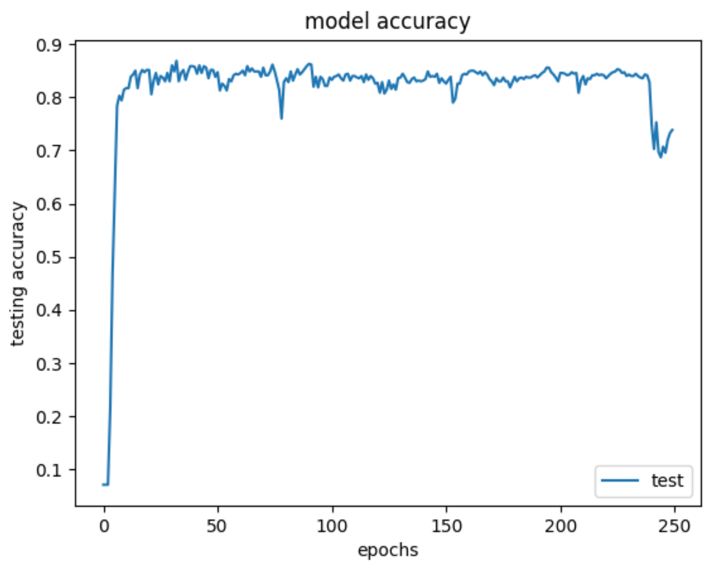
Testing accuracy converges around epoch 100



Train loss = 1.28

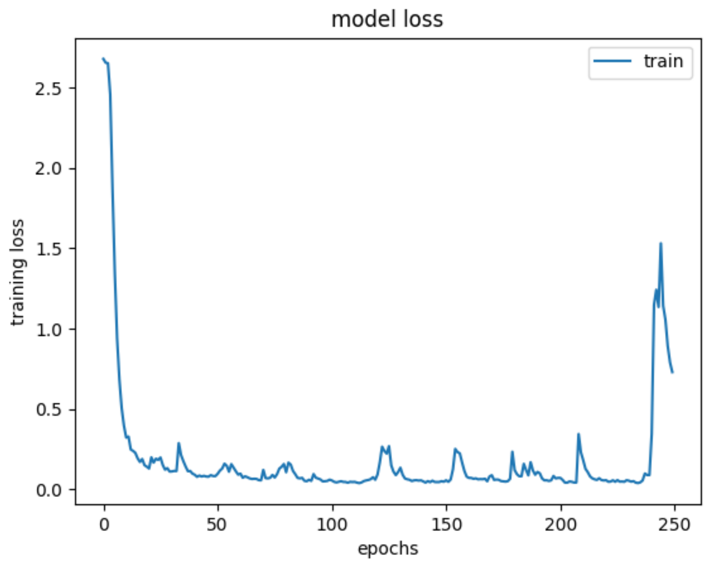
Training loss continues to decrease gradually

4. Word RNN classifier



Test accuracy = 73.9%

Testing accuracy converges since epoch 25, however, had a drastic drop in accuracy around epoch 230



Train loss = 0.729

Likewise, the training loss had a drastic increase around epoch 230

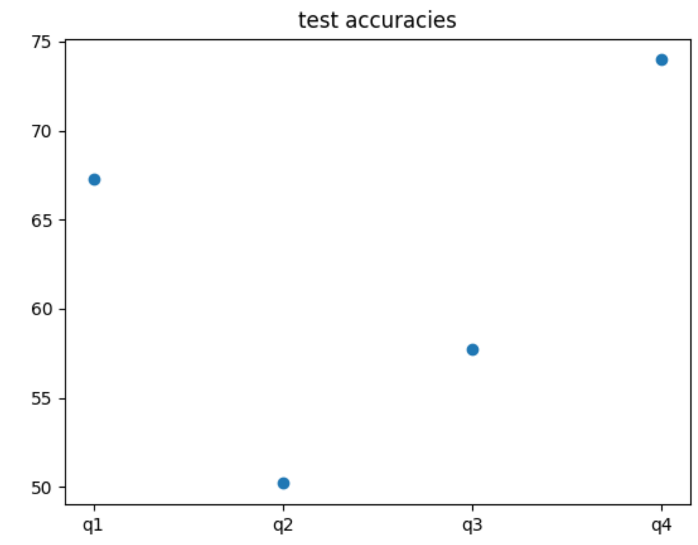
The change might be due to a few reasons:

1. The learning rate is too high
2. vanishing gradient/ dead neurons
3. NaN values

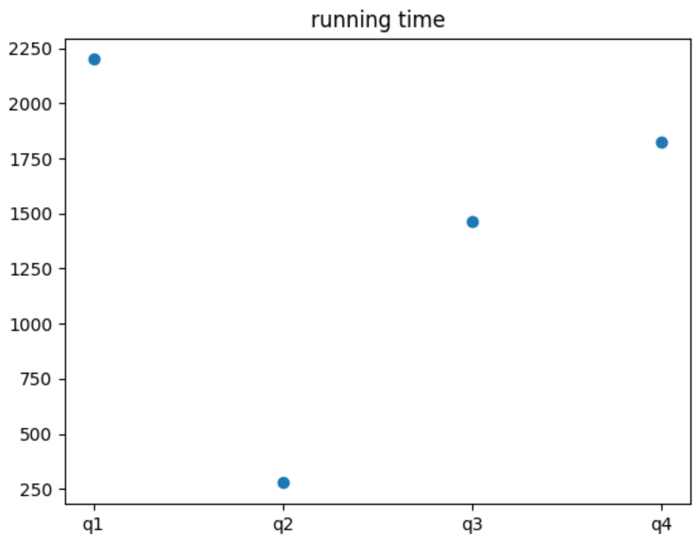
5. Compare the test accuracies and the running time

5.1. With dropout

Question/ Model	q1 Char CNN	q2 Word CNN embedded	q3 Char RNN	q4 Word RNN embedded
Test accuracies(%)	67.3	50.2	57.7	73.9
running time	2200.1104850769043	277.6241948604584	1462.3785090446472	1826.6572318077087



The word RNN with embedded layer(q4) provides the highest testing accuracy, at 73.9. The testing accuracy goes by: q4 > q1 > q3 > q2



The word CNN with embedded layer(q2) provides the lowest running time, 278 seconds. The running time goes by: q2 < q3 < q4 < q5

The testing accuracies is fairly constant with the running time, such that the longer the training time, the higher the accuracies.

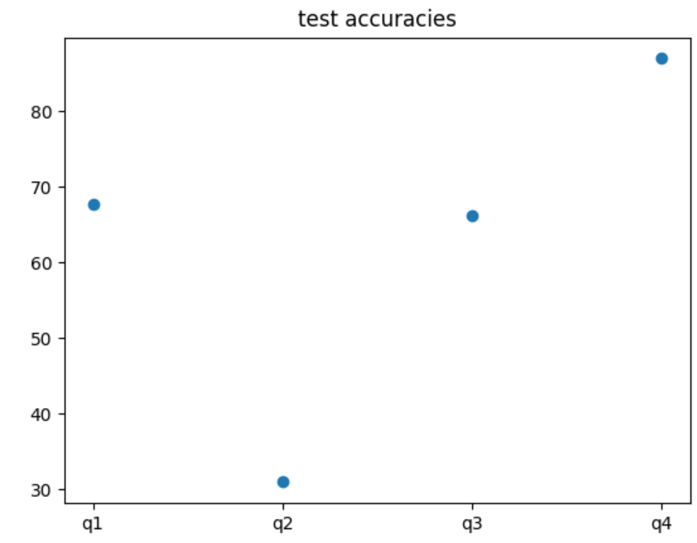
A character CNN works better than a character RNN.

A word RNN works better than a word CNN.

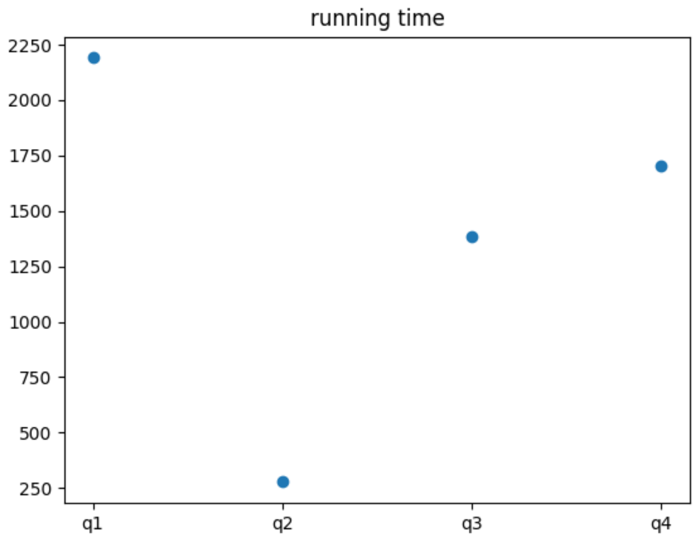
5.2. Without dropout

The drop out layer have been removed from both CNN and RNN

Question/ Model	q1 Char CNN	q2 Word CNN embedded	q3 Char RNN	q4 Word RNN embedded
Test accuracies(%)	67.7	31.0	66.3	87.0
running time	2191.3902111053467	277.96397590637207	1384.8000147342682	1705.319620847702



The word RNN with embedded layer(q4) provides the highest testing accuracy, at 87.0. The testing accuracy goes by: q4 > q1 > q3 > q2



The word CNN with embedded layer(q2) provides the lowest running time, 278 seconds. The running time goes by: q2 < q3 < q4 < q5

5.3 Comparison of with/without dropout

	Question/ Model	q1 Char CNN	q2 Word CNN embedded	q3 Char RNN	q4 Word RNN embedded
Dropout	Test accuracies (%)	67.3	50.2	57.7	73.9
	running time	2200.1104850769043	277.6241948604584	1462.3785090446472	1826.6572318077087
No dropout	Test accuracies (%)	67.7	31.0	66.3	87.0
	running time	2191.3902111053467	277.96397590637207	1384.8000147342682	1705.319620847702

The relationship between the different model reminds constant when there is dropout and when there is no dropout. Such that for test accuracies, the dropout experiment has : q4 > q1 > q3 > q2 , and non-drop out experiment also has q4 > q1 > q3 > q2 .

3 models have their accuracies reduced when there is dropout, except for model in question2, word CNN with embedded layer.

The running time reminds relatively constant, with a maximum difference of 120 seconds and a minimum of less than a second.

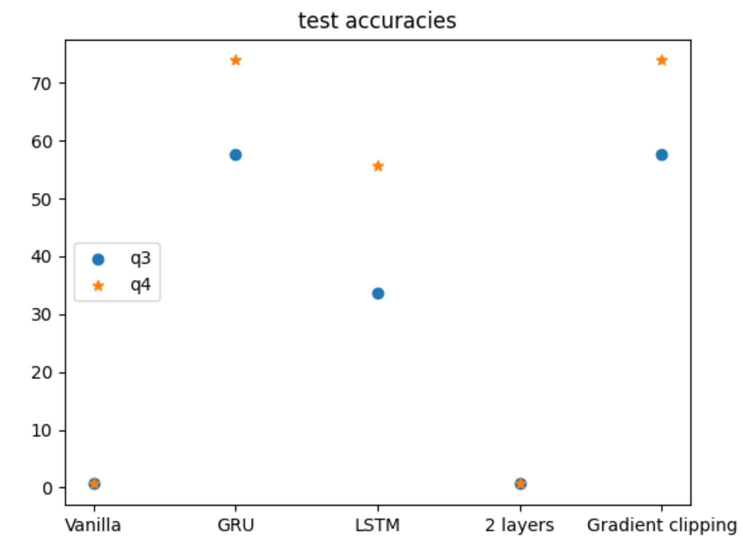
6. For RNN networks implemented in (3) and (4), perform the following experiments with the aim of improving performances

Question/ Model		q3 Char RNN	q4 Word RNN embedded
Vanilla layer	Test accuracies(%)	0.82	0.71
	running time	647.7193531990051	837.0420229434967
GRU layer	Test accuracies(%)	57.7	73.9
	running time	1462.3785090446472	1826.6572318077087
LSTM layer	Test accuracies(%)	33.7	55.6
	running time	1159.1800620555878	1377.6565296649933
2 layers RNN + GRU layer	Test accuracies(%)	0.66	0.71
	running time	2364.2441234588623	2730.2069561481476
Gradient clipping + GRU layer	Test accuracies(%)	57.7	73.9
	running time	1404.0322885513306	1831.9784710407257

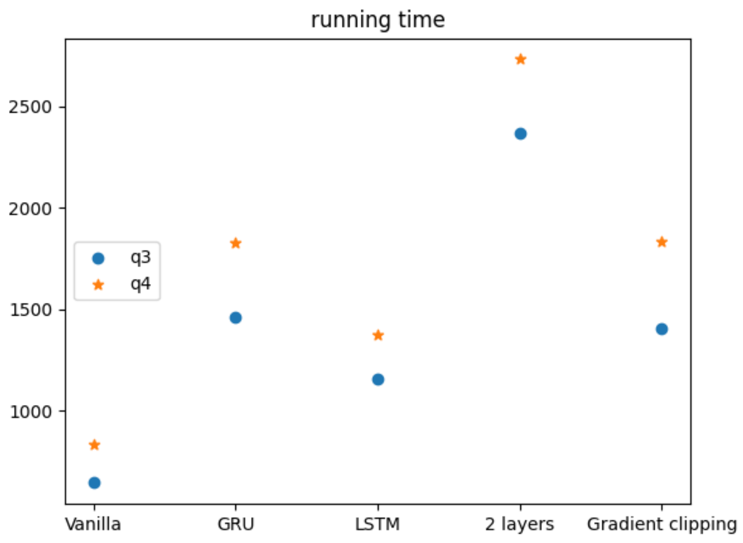
The test accuracies for the vanilla RNN layer is unbelievably low. The test accuracies for the LSTM layer is also unexpectedly low, as it is supposedly to be better than a vanilla RNN layer. However, indeed that the test accuracies of LSTM are greater than that of a vanilla RNN. It is fair to say that LSTM has managed to solve the accuracy issue of the vanilla layer.

The test accuracies for the 2 layers RNN is also unbelievably low as it is built on top of the GRU model. However, its accuracies is lower than that of a single vanilla RNN model. Essentially, it means that the input pattern is not complex enough for the complexity of a stacked RNN model.

The test accuracies for the gradient clipping RNN is the same as the layer without gradient clipping. This means that the original (GRU layer) faces minimal exploding gradient issue.



The difference between the accuracies are generally constant, with the vanilla and 2 layers at an almost identical level.



The running time ratio between a character RNN and a word RNN is even more equal than their test accuracies.

Conclusion

In this paper it has been showed that not all methods can increase accuracy/ prevent overfitting, even though the methods are supposed to aid with those components. It is essential to experiment around for an optimal model