Q 2 -1

1. Adding a layer or two might help extract more features, this can be done to an extant for after a certain point this might result in the network overfitting the training data as can be seen with the IMDB cnn after adding another layer.

It seems that adding another layer to the CIFAR cnn results in somewhat of an improved accuracy.

The idea in the end is to try and extract more features in comparison to the basic architecture of the network.

1. The idea of adding 2 layers is the same as adding 1.

We tried to improve accuracy by extracting more features from the network using more layers and more filters, this resulted with very small improvement to the accuracy of the CIFAR10 cnn while the IMDB cnn performed worst with each layer added and seems to have overfit the training data while doing not so well in regard to the test data.

It seems that adding layers increase network complexity and as long as the data set remains the same the increased complexity (more weights) will lead to overfitting.

1. IMDB –

1. Set with 3 layers –each layer has 100 filters of sizes 3, 4, 5.

The idea according the a paper I read is to look for the appearance of tri gram 4 gram and 5 gram in the review thus finding word sequences indicative to the review sentiment.

2. tried the approach above in a reverse order of the kernels sizes 5-4-3, this did not go very well as illustrated in the graphs (see att. ? ).

3. Removed 2 of the layers (1 left) set the filter sizes to 3 with a 100 filters

The idea was to try and reduce network complexity (number of weights) to try and get improved accuracy.

It seems this change made the most impact on accuracy and loss. (Please see att ?)

4. Tried the above approach of reducing complexity. Used 1 layer with 200 filters thus doubling the number of kernels while not adding more depth (layers).

This change seems to have a positive impact in comparison with other changes attempted but still not an improvement to the basic network configuration.

5. used 1 layer with 64 filters of size 3 – basically removing 2 layers from the basic configuration still under the assumption that reduced complexity might have a positive effect on accuracy – this seems to have a positive effect on training accuracy which actually performed better than the basic configuration however validation accuracy was slightly better on the basic configuration (the validation loss was slightly better using this configuration!)

CIFAR-10 –

1. Removed a couple of the conv layers to reduce complexity- removed a section containing 2 conv layers – the one with the 64 filters.

This change damaged the loss factor and reduced nearly 10% of test accuracy.

1. Added another section similar to the basic configuration – 2 layers with 128 filters of size 3\*3 with RELU and max pooling layer-this change resulted in an improved test accuracy/loss also test accuracy/loss showed significant improvement.
2. Removed a section containing 2 conv with layers with 32 filters each – this change resulted in a decreased test lose and accuracy.
3. Changed the number of filters from 64 to 32 in the second section and changed the filter size to 4\*4 – this change resulted in very small decrease to test accuracy/loss.
4. Added another 2 conv layer section in a way that all conv layers have 64 filters of sizes 2 , 3, 4 – this change brought better results on both accuracy and loss it seems the more complex network managed to extract more features (or more meaningful features).

Q2 -2

***CIFAR10***

SGD

1. SGD – this change resulted in somewhat of an improvement in both accuracy and loss.(very small change)
2. SGD with learning rate increased to 0.1 – change resulted in drop in performance (both accuracy and loss).
3. SGD with decreased epochs (25 epochs) – this change resulted in a very small decrease in test accuracy.
4. SGD with an increased number of epochs (100 epochs) - this change seems to have brought the most significant improvement with test accuracy of over 80%.

\*it looks like after about 40 epochs the accuracy stopped improving and the learning process seems to have reached a plateau.

1. SGD with LR increased to 0.05 and epochs decreased to 30 – this resulted in a slight almost insignificant improvement to the accuracy.

SGD with momentum

1. SGD with Momentum set to 0.9 – a slight drop in accuracy (very small) was detected – results are practically the same accuracy wise.
2. SGD with Momentum set to 0.9 and Nesterov set to true – as in (1) a drop in accuracy can be seen( a few percentages drop to accuracy).
3. The same configuration as (2) but with learning rate increased to 0.02 – most likely the worst results seen thus far with accuracy of 10%, the interesting thing is that the network reached accuracy of around 50% along the training process but after a number of epochs a big drop to the performance can be seen the most significant drop occurs around the 40th epochs when performance drops to around 10% very rapidly.
4. The same configuration as (3) but with less epochs (30 epochs) – a clear and meaningful drop in test accuracy in comparison to the original configuration with numbers around 50%.
5. The same configuration as (2) with more epochs (65) – more epochs resulted in decreased accuracy in comparison with the original network.

* Attempted an additional few runs with an increased LR (0.05) but results were catastrophically worst in comparison to the basic network configuration so results were scrubbed along with the graphs associated.
* It seems that while working on this network using SGD with momentum, increasing the learning rate is not the way to go.

ADAM

1. First run I used default ADMA parameters – results are small improvement no the original network with accuracy of just under 80%.
2. Decreased the learning rate to 0.0005 – this proved to have a good effect on performance with accuracy reaching just over 81%.
3. Tried default parameters with 75 epochs – very small improvement of accuracy.
4. Tried the default parameters with 35 epochs – the performance was almost the same in comparison to the original configuration with a very small improvement in accuracy.

* Additional runs were made with increased LR – results were poor to say the least – for example LR of 0.01 and 0.005 resulted in accuracy not surpassing 10%, notebooks and graphs were scrubbed as irrelevant.
* Looks like an increased LR is not the path!

RMSPROP

1. First run with default parameters which are slightly different from the original network (different learning rate and no decay) – this provided results of just over 10% accuracy – the accuracy actually reached almost 70% but started a steady decline around the 10th epoch.
2. Increased the learning rate to 0.005 (around half way between (1) LR and the original network) – this provided with test accuracy of just over 50% - a clear decline in performance can be observed after the 15th epochs approximately.
3. An increased learning rate of 0.003 with only 25 epochs - results were terrible, with accuracy under 50%.
4. Same as (2) but with 10 epochs – best run with this optimizer yet with accuracy of around 73%,

* Another run was made with default parameters and 70 epochs, results and graphs were discarded for it seems that more epochs is not the way to go in context with RMSProp and this network.
* While using this optimizer it is very easy to see that after a certain number of epochs performance starts to drop rapidly and steadily – please see graphs attached.

ADAGrad

1. Used the default Keras parameters for the first attempt – results were slightly better than original net with accuracy just shy of 80%.
2. Decreased learning rate of 0.005 – very small improvement with accuracy just over 78%.
3. Tried training with less epochs (30 epochs) – test accuracy was just shy of 80%.
4. Tried training with more epochs (70 epochs) – this run yielded the best results thus far with accuracy just over 80%

Dropout

1. Increased dropout rate for the first 2 convolutional layers to 0.5 – results are very similar to the default dropout rate with a very small decline in performance.
2. Reduced dropout rate for the first 2 convolutional layers to 0.1 and 0.25 on the third layer – this change resulted in very small improvement to accuracy.
3. Set the dropout uniformly at 0.65 across all layers - resulted in around 10% drop to accuracy.

* another run was attempted with learning rate of 0.05- accuracy seemed to have reach a plateau at around 10% and the run was terminated after 15 epochs results and code were discarded.

\*\*In regards to performance drop after a certain number of epochs – this can be seen very clearly in the graphs associated with SGD with momentum (3) and with almost all the runs using RMSProp as an optimizer.

***IMDB***

SGD

1. Switching to SGD from ADAM resulted in a significant drop on both accuracy and loss – while epoch number and learning rate remains the same.
2. SGD with adding more epochs (15) – while the learning rate remains the same the network did much worse than the basic configuration utilizing ADAM as optimizer.
3. Experimented with an increased learning rate (0.1) and more epochs (15) – the network still performs poorly In comparison with the original configuration. (It seems after the 10th epoch the accuracy started to drop with each epoch).
4. As a conclusion from (3) attempted to lower the epochs to 10 while LR remains 0.1 – this configuration gave the best results while using SGD but far worse than the original network (using ADAM).

SGD with momentum

1. For the first run I chose momentum of 0.5 to try and get a notion of how it will affect the network – the network did not perform very well with accuracy around 50%.
2. Attempted to use momentum value of 0.9 and increasing the epoch number to 10 – the changes to results were insignificant in comparison to (1).
3. Tried the same configuration as (2) with learning rate of 0.1 – the results are the same as (1) and (2) not really worth mentioning.
4. Same configuration as (2) but with Nesterov set to true – this achieved accuracy of 50%.

ADAM

1. The basic network utilizes ADAM to achieve fairly good results with accuracy around 85% - I tried using ADAM with the parameters shown on the Keras doc – this resulted in a very small improvement to the basic network accuracy with a little over 86%.
2. Attempted the configuration of (1) with 10 epochs - more epochs resulted in practically no change to accuracy while test loss took a beating.
3. Attempted the configuration of (1) with 15 epochs - the same as (2) – no change worth mentioning to test accuracy while test loss is higher than the basic 3 epoch configuration.
4. Reverted back to 3 epoch’s with learning rate of 0.1, 0.01 , 0.05 instead the 0.001 proposed in the keras doc – accuracy dropped to around 50% - also tried playing around with batch size but to no avail.

RMSProp

1. First run I was using the parameters suggested in the Keras doc – test accuracy and loss turned out to be the best thus far with accuracy around 88% and loss just under 0.3 (on the first run of this configuration)
2. Tried increasing the epoch’s to 5 - 5 epoch’s gave about the same results as (1).
3. Tried increasing the epoch’s to 10 - the results to test accuracy and loss were not very high and the network actually did worse than the 3 epoch network – while the train accuracy and loss looked much better but this probably was the network overfitting the train data
4. Tried increasing the epochs to 15 - results and observation are the same as (3).

ADAGrad

1. For the first run of ADAGrad I used the parameters from the Keras doc – the results are not reassuring as the accuracy dropped to under 50%.
2. Adding more epochs (10 epochs) did not help the situation with poor accuracy of around 50%.
3. Increased learning rate to 0.1 but to no avail – accuracy remains a joke at 50%.
4. Increased the learning rate to 0.05 – results mirror (3).

Dropout

1. Increased dropout parameter to 0.5 – the results were a slightly improved accuracy and almost 0.1 improvement of the test loss.
2. Tried decreasing dropout to 0.1 – results were quite similar to the basic configuration of 0.2 – still slightly worst.
3. Tried increasing dropout furthermore to 0.8 – this resulted in the worst impact with accuracy around 50% (worst in context to the dropout changes section).
4. Tried dropout of 0.6 – the results were quite similar to (1) with test accuracy just shy of 87%.

* A Drop in performance after a certain number of epochs can be seen clearly in the SGD section part (3) – after the 10th epoch performance started to drop (accuracy).

Please see attached graph in appendix.

Q3

1. GRU – the change from the LSTM layer to the GRU layer resulted in a very small drop in accuracy (just over 82% in comparison with the LSTM which achieved over 83%)

SimpleRNN – the switch from LSTM to SimpleRNN layer resulted in a more significant drop to test accuracy with around 77% - as mentioned above the LSTM configuration achieved around 83% accuracy.

The best results were achieved while using the GRU layer with accuracy surpassing 84% (The original LSTM configuration achieved around 83%)

The networks summary in regards to number of parameters is attached under the Model summary folder.

* The parameters summary of each of the networks can be viewed in the attached documents.

1. Best results were received with the GRU(1) – allowed to run for 5 epochs – increased epochs number resulted in accuracy just shy of 85%.

It seems the best change was adding more epochs…

Reducing embedding out dim provided with very small improvement in accuracy (50 out dim also tried 32 as out dim while setting the unit size the same – accuracy received was around 82%).

We were not able to improve performance with the SimpleRNN layer in comparison to the LSTM configuration, we did manage to improve accuracy using the GRU layer – please see graph attached!

* Attempted several changes to the GRU configuration:

Tried changing max features number, played around with dropout values, also tried different optimizers (RMSProp, AdaGrad) basically attempted all the changes described for the SimpleRNN – The best results received were when more epochs were added or when the unit and embedding dim were modified.

* Attempted several changes to the SimpleRNN configuration:

Tried running for more than 2 epochs with lousy results, attempted to change the embedding dimension, batch size and the max features to 3000 again the results were unkind.

Tried using RELU as activation instead of the tanh but to no avail, performance dropped while trying to engage said changes.

Dropout did not help the situation – tried to set it to 0.2 with very bad results.

Setting the embedding out dim to 64 did not help performance levels as well.

Using a different optimizer was also not the way to victory.