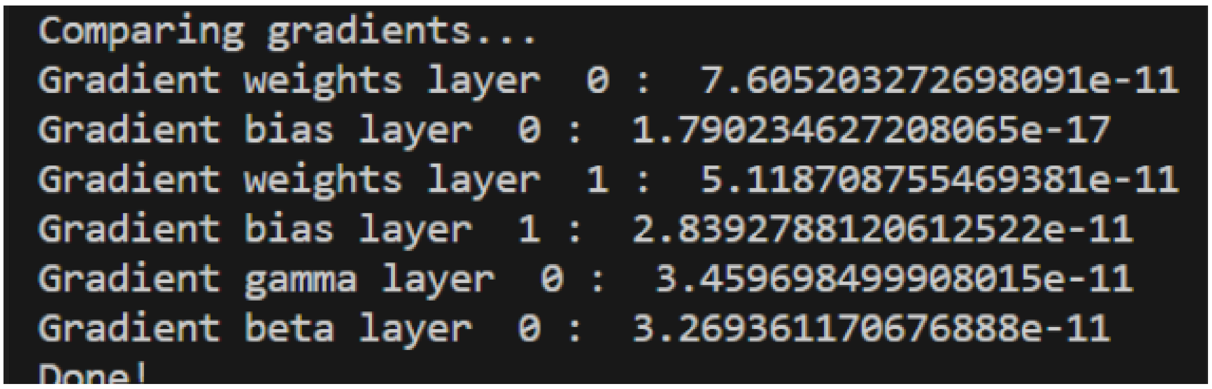
DD2425 - Assignment 3
Rakin Ali

Lab 3 - K layer network

1. Gradient checking

Two functions that calculate the gradients were created. The first one was called *'back_pass'* and it estimated the gradients analytically. The other one *'compute_gradients_slow'* did it numerically. To check the gradients, it's sufficient to take the max value of the absolute value of the gradient vectors from both of the functions and compare them. If the max is above 10^{-10} then the 'back_pass' function is not close enough.

To speed things up, it's also sufficient to take the first 1000 points of the data and use the [input, 10,10] layer. Although this is not the most accurate way of checking the gradients, it's sufficient enough. Regularisation was set to 0.05. Describing the initialization settings for the network is irrelevant as the two gradient calculation functions are supposed to get similar answers either way. Batch normalization was applied. These are the results:



```
Comparing gradients...
Gradient weights layer  0 :  7.605203272698091e-11
Gradient bias layer  0 :  1.790234627208065e-17
Gradient weights layer  1 :  5.118708755469381e-11
Gradient bias layer  1 :  2.8392788120612522e-11
Gradient gamma layer  0 :  3.459698499908015e-11
Gradient beta layer  0 :  3.269361170676888e-11
Done!
```

Table 1: Passed means it's smaller than 10^{-10}

Layer	Gradient beta	Gradient gamma	Gradient weights	Gradient bias
0	none		Passed	Passed
1	Passed	Passed	Passed	Passed

A more thorough checking would be needed but since Batchnormalisation significantly improved the results and the network behaved as expected, it was deemed unnecessary.

2. Evolution of loss function - 3 layers

Parameter settings:

Batch size = 100
He initialization
Eta values ∈ $[10^{-1}, 10^{-5}]$ cyclic learning rate
Step size = 2250 (5 * 4’500/ Batch size)
Lambda = 0.05
Alpha = 0.9
Cycles = 2
Layers [Input layer,50, 50,10]

The only difference between these two graphs below is whether batch normalisation is implemented or not. The first one is with the batch norm and the other is without batch normalisation

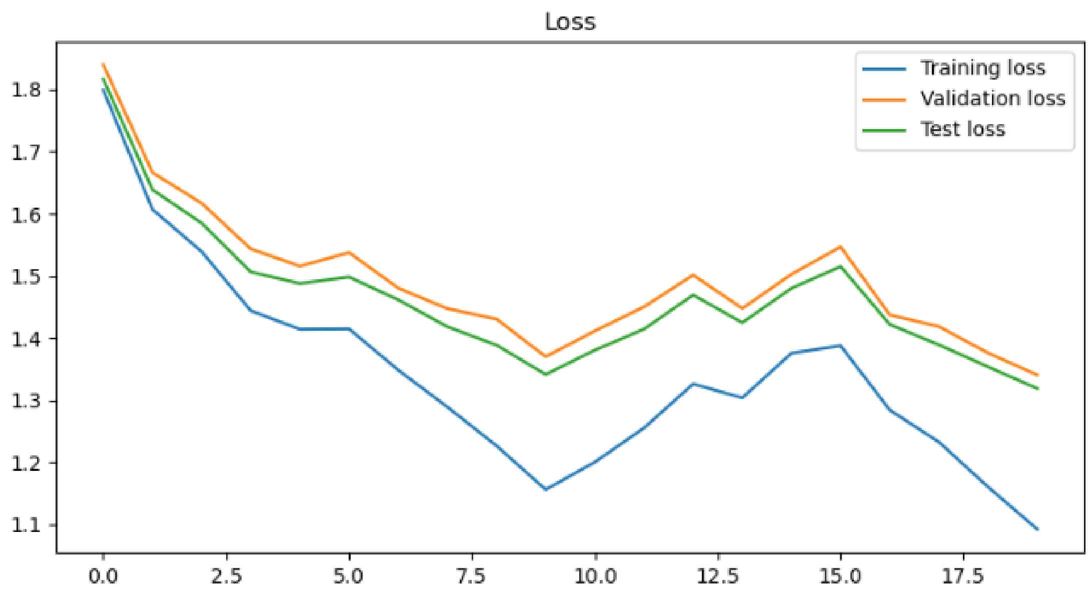


Figure1: Batch normalisation

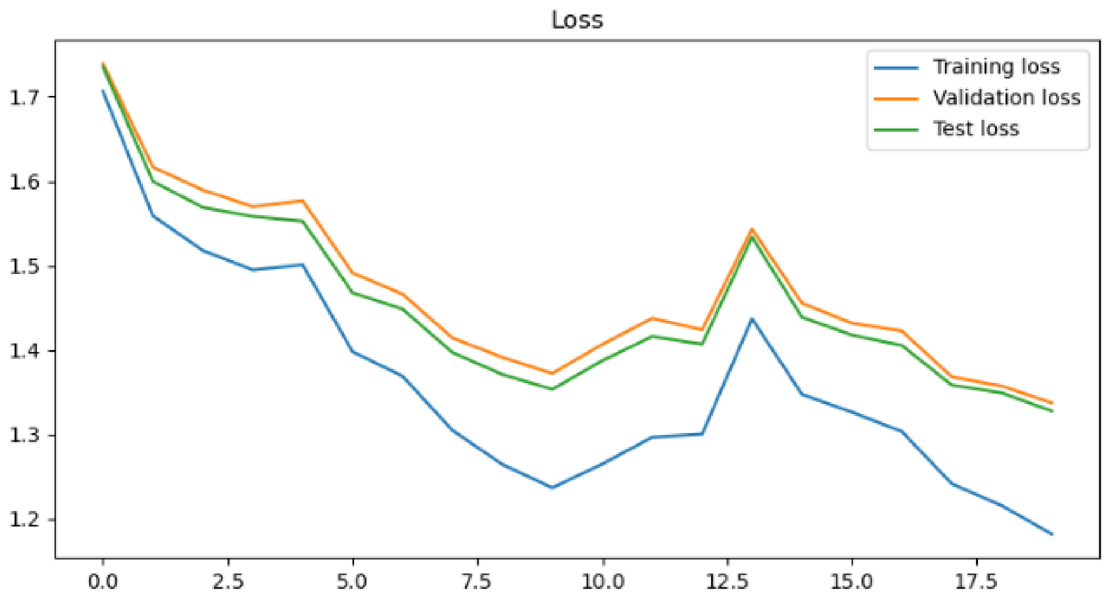


Figure 2: No batch normalisation

Comments: The loss curve of the batch normalisation is smoother and does not spike as much as the curve without batch normalisation. They look similar here however when the lambda was increased by 10, that is lambda = 0.05 instead of 0.005 then the difference was a lot more evident. Figures 1 & 2 graph just, barely, proves that Batch normalisation improves loss/cost functions.

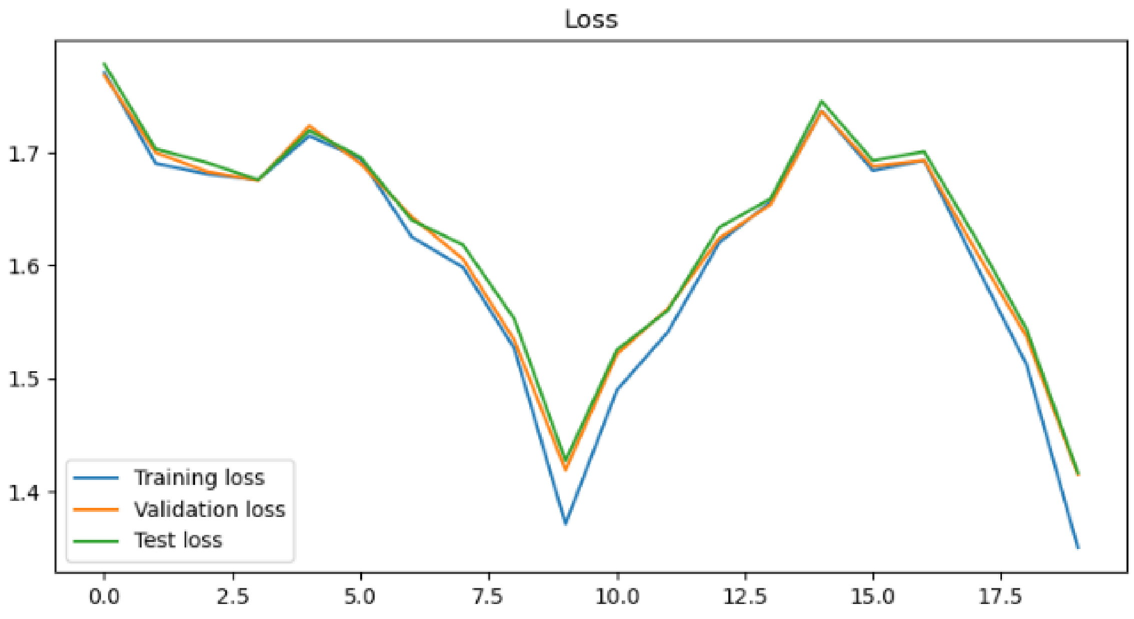


Figure 3: Batchnormalisation with increased regularisation

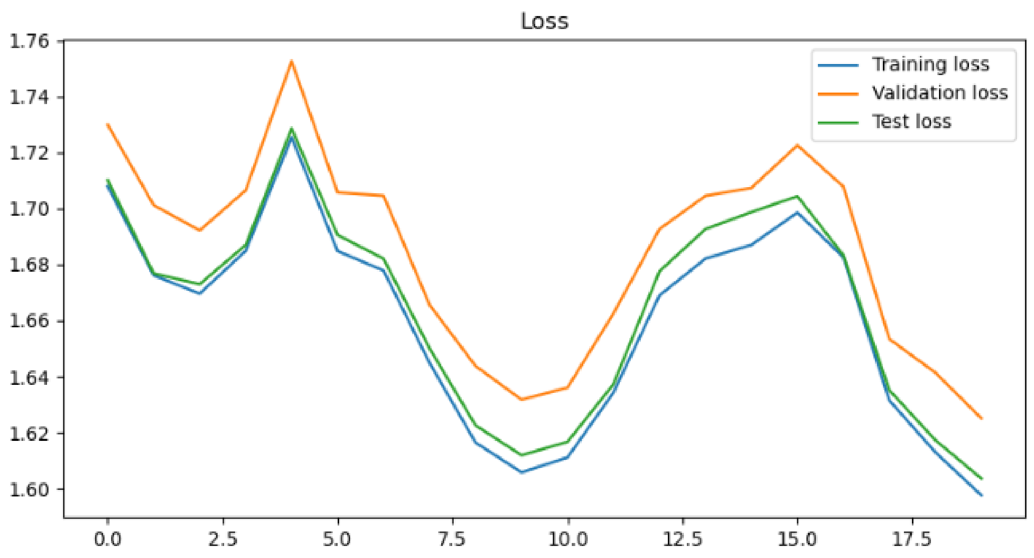


Figure 4 No batch normalisation with increased regularisation

Extra comments: Here is where we start to see the effects of batch normalisation when regularisation is increased.

3. Evolution of loss function - 9 layers

Parameter settings:

Batch size = 100
He initialization
Eta values ∈ [10^{-1}, 10^{-5}] cyclic learning rate
Step size = 2250 (5 * 4’500/ Batch size)
Lambda = 0.005
Alpha = 0.9
Cycles = 2
Layers [Input layer,50, 50,10]

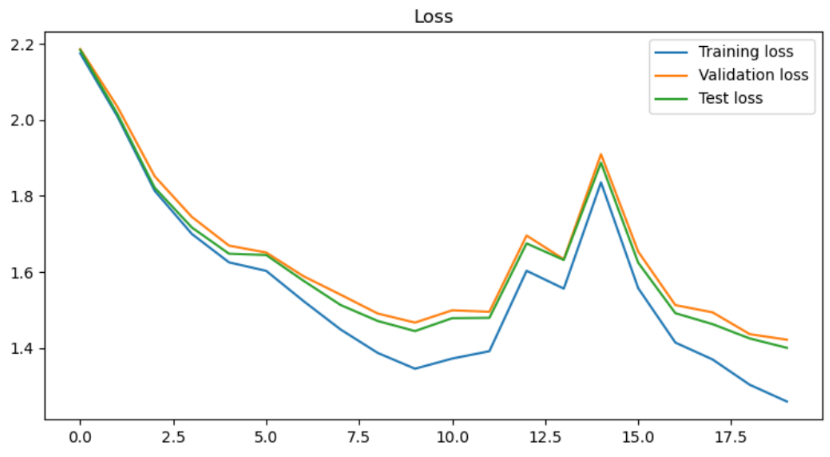


Figure 5: No batch normalisation

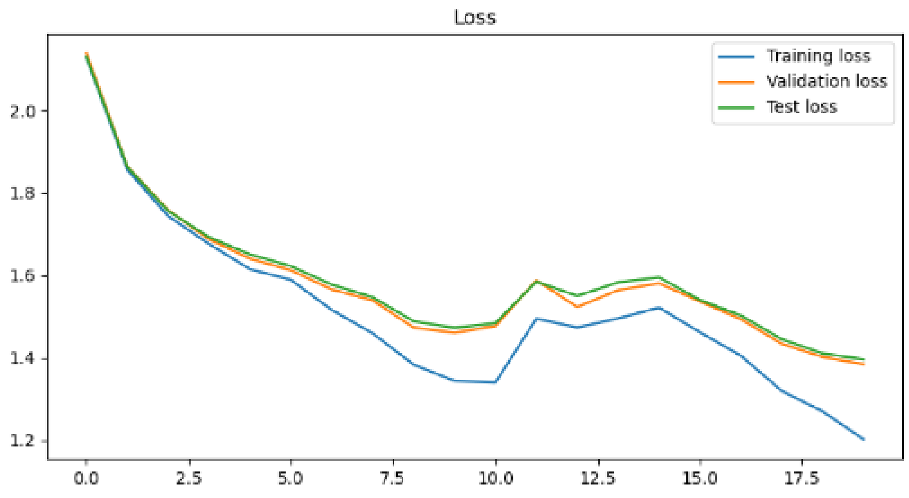


Figure 6: Batch normalisation

Comment: Here it definitely became more evident that batch normalisation is good when working with deeper neural networks.

4. Searching for good lambda values ← 3 layers

Large lambda search: To make things simpler. The first five random numbers uniformly distributed between [-5,-1] were taken thereafter $10^{\text{random Numbers}}$ became the lambda. Their corresponding final accuracies on the test data were stored.

Narrow lambda search: Afterwards, a narrow lambda search was conducted. The narrow lambda search took 3 numbers uniformly at random between the first and second best lambda values. The lambda with the highest accuracy was returned.

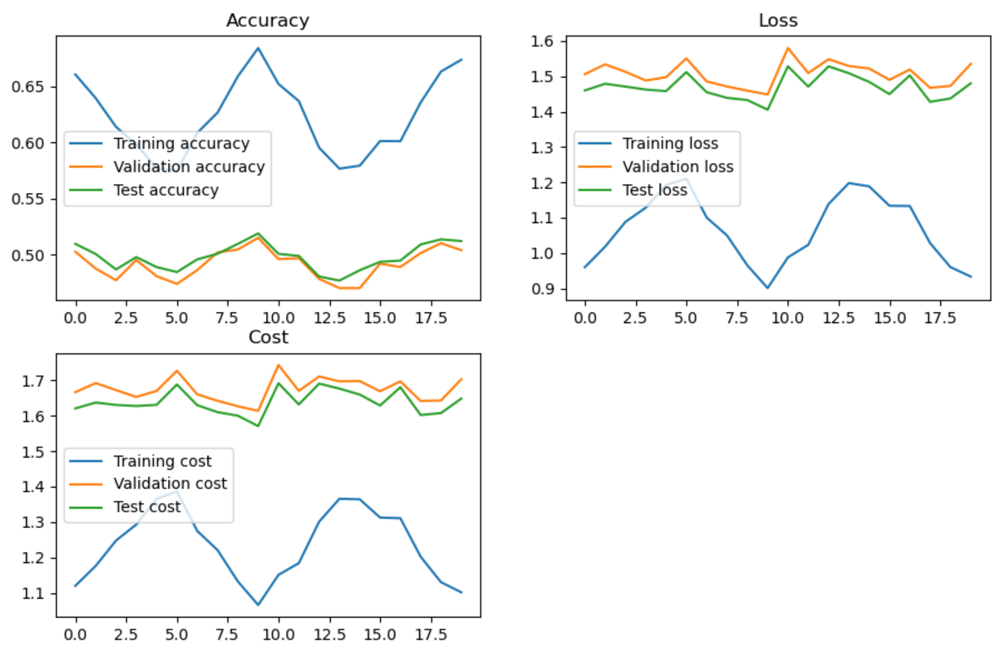
Parameter 1: Large lambda search:

Batch size = 512
Cycles = 1
Stepsize = 2250
Start learning rate = 0.001
do_Batchnorm = True

Parameter 2: Narrow lambda search

Batch size = 100
Cycles = 2
Stepsize = 2250
Start learning rate = 0.001
do_Batchnorm = True

Best lambda returned: 0.0010312147716679328

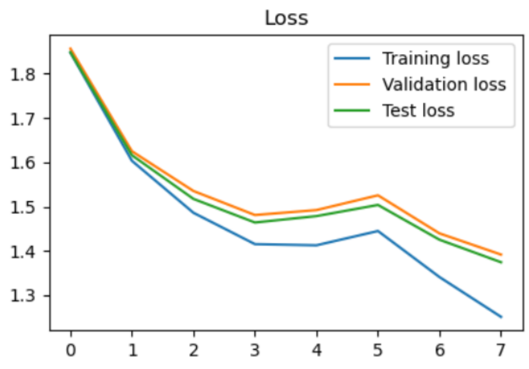


Comment: Although a bit of overfitting, still got a good result.

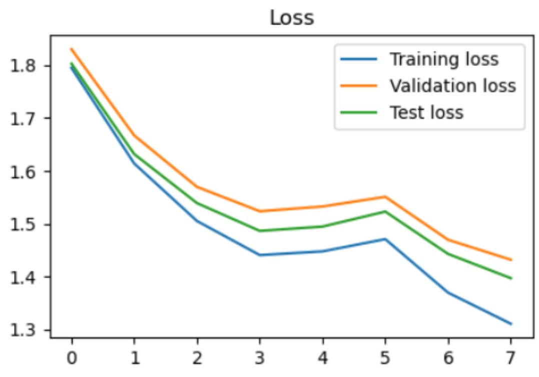
5. Sensitivity to Initialization:

Parameters: Narrow lambda search

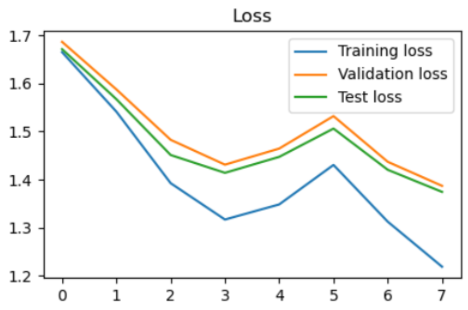
Batch size = 100
Cycles = 2
Stepsize = 900
Start learning rate = 0.001
do_Batchnorm = True
lambda = 0.005



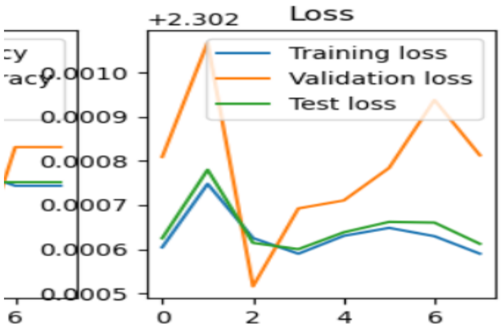
Sigma = 1e-1 Batchnormalisation



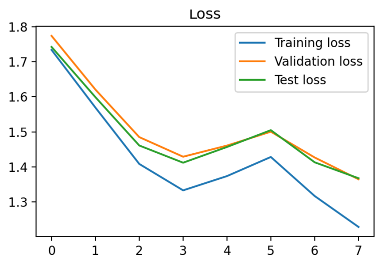
Sigma = 1e-1 Batchnormalisation



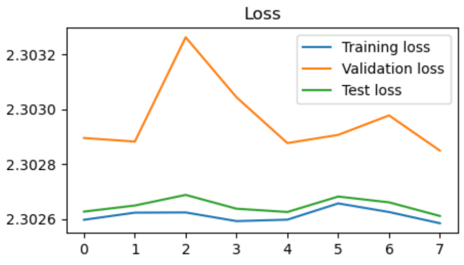
Sigma = 1e-3 Batchnormalisation



Sigma = 1e-1 Batchnormalisation



Sigma = 1e-4 Batchnormalisation



Sigma = 1e-1 Batchnormalisation

Comments: Sigma values seems to have a substantial effect when there is no batch normalisation conducted. When batch normalisation is conducted, the loss function still more or less looks the same. When sigma is 1e-4 then almost no learning should be done however with batch normalisation learning is still conducted while without it any learning is done. This shows that batch normalisation is a lot less sensitive to initialisation