DD2424 - Assignment 3

Marcus Jirwe

March 2020

Introduction

The purpose of the assignment was to train a multi-linear classifier with a variable amount of hidden layers to perform classifications on images from the CIFAR-10 dataset. Additionally the impact of the introduction of batch normalisation is studied and how it can be used to negate the negative impact on performance when training deeper models. Practically, this report relies on code programmed in python and primarily using the packages "numpy" and "matplotlib", with a function to compute gradients numerically being adapted from matlab.

Numerical vs Analytical gradient

More specifically, the relative difference between the gradients is calculated by the equation:

$$diff = \frac{||grad_{analytical} - grad_{numerical}||}{||grad_{analytical}|| + ||grad_{numerical}||}$$

The numerical difference between the different kind of gradients without batch normalisation for a 4-layer network were:

The gradient relative difference for W for layer 1: 1.0396758690215455e-10
The gradient relative difference for W for layer 2: 1.0816098792339167e-06
The gradient relative difference for W for layer 3: 3.119750765529875e-06
The gradient relative difference for W for layer 4: 2.789056626773761e-06
The gradient relative difference for b for layer 1: 1.8321692144096816e-06
The gradient relative difference for b for layer 2: 2.3905082333923803e-06
The gradient relative difference for b for layer 3: 3.113804059695996e-06
The gradient relative difference for b for layer 4: 3.471231731625855e-06

Which seems small enough to assume the calculations to be correct. The results below seem to lend credit to this conclusion. However, the results for the checks grow more problematic when testing the same sized network with batch normalisation:

The gradient relative difference for W for layer 1: 0.45953875826419116
The gradient relative difference for W for layer 2: 0.04671610089458472
The gradient relative difference for W for layer 3: 0.04008675518700235
The gradient relative difference for W for layer 4: 1.5680017551794778e-06
The gradient relative difference for gamma for layer 1: 0.052245723753747704
The gradient relative difference for gamma for layer 2: 0.049345534470887804
The gradient relative difference for gamma for layer 3: 2.4070260694309738e-06
The gradient relative difference for gamma for layer 4: 0.0
The gradient relative difference for beta for layer 1: 0.047071029944393696
The gradient relative difference for beta for layer 2: 0.037945718168486284
The gradient relative difference for beta for layer 3: 2.979537082198368e-06
The gradient relative difference for beta for layer 4: 0.0

The gradients for the biases have been omitted for the batch normalisation check as they

are mentioned to be superfluous when using batch normalisation. The difference becoming greater as the gradients are propagated back through the network are very obvious here. The greatest relative difference is the normal weights for the first layer of the model, where the difference is something like 40%. I do not really understand why the difference is so large when batch normalisation is implemented since I spent a long time looking for errors in the implementation and could not find any. Furthermore, the results when using the implementation seems to coincide with the results presented as an example in the assignment text, which serves as a nice sanity check. This could potentially mean that this method of gradient checking is insufficient for this task.

1 Practical and implementation

The code relies heavily on the use of the "numpy" library with "matplotlib" being used for plotting the results. Care was taken so as to ensure the dimensions of the matrices were the same as the one in the assignment description, which made it easier and more straightforward to follow the instructions. The testing was performed in the code's main function. The assignment necessitated the implementation of the analytical form of the cross-entropy loss and its gradients as opposed to relying on the numerical gradient. To do this, a derivation shown in the lecture notes were followed and the equations were extended to matrix form. To test that the results of the analytical gradient function was correct, it was tested against the values from the more accurate numerical gradient function using the centered difference. The testing was performed by calculating the Frobenius norm of the difference of the two gradients and dividing it with the sum of the individual norms. This gives a measure of a relative difference between the two. It seems that the implementations would be bug-free as the results to follow seem logical. A test was also done to check how sensitive models of different depths were to initial conditions. Normally a He-initialisation was performed for the parameters in the network. But for this experiment, a constant standard deviation was used for each layer with and without batch normalisation. The size of the models were always the same for a given amount of layers. When 3 layers were used, the amount of hidden nodes in each hidden layer was 50. For 9 layers, the amount of hidden nodes from the first layer to the last were: 50, 30, 20, 20, 10, 10, 10, 10. This gave it the same amount of weight parameters as the 3-layer models.

2 Results

Below are the cost, loss and accuracy curves for a 3-layer and 9-layer model, respectively. First are curves for these networks without any batch normalisation and then with batch normalisation, to give a proper oversight on what effect batch normalisation has. Furthermore the three best λ-values for a coarse grid search as well as the three best λ-values in an uniform interval around each of these are shown in two separate tables. This grid search was performed only for a 3-layer network. The networks trained on these sampled λ-values were trained for two cycles of training each to make sure the network performance reflected a properly trained network. For the value of λ for which a model performed the best a network 3-layer and 9-layer network was trained for three cycles of training and evaluated. To ensure that a fair measure of the performance was captured a mean and standard deviation of the performance over 20 runs was calculated. Finally, a study was made as to how sensitive shallow and deeper models were to initial conditions and how this sensitivity could be remedied with the use of batch normalisation.

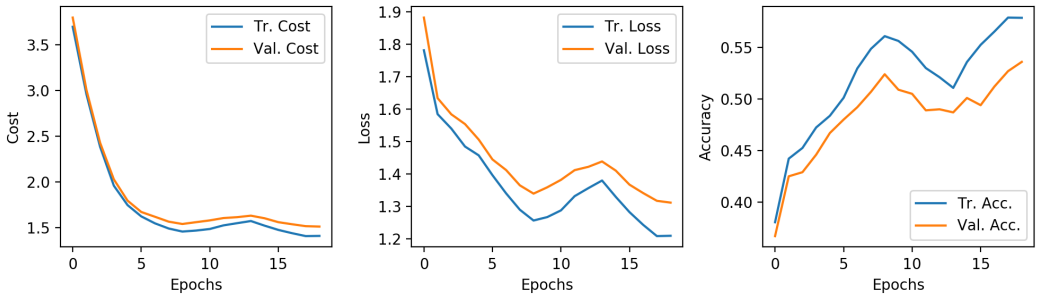

Figure 1: Cost, Loss and Accuracy plots for the hyper-parameter settings: $\eta_{min} = 1e-5, \eta_{max} = 1e-1, \lambda = 0.005, n_s = 2250$. and a batch size of 100 with 19 epochs, without batch normalisation (3 layers).

No Batch normalisation

Accuracy for 3 layers

Training Accuracy: 0.5787
Validation Accuracy: 0.536
Test Accuracy: 0.5213

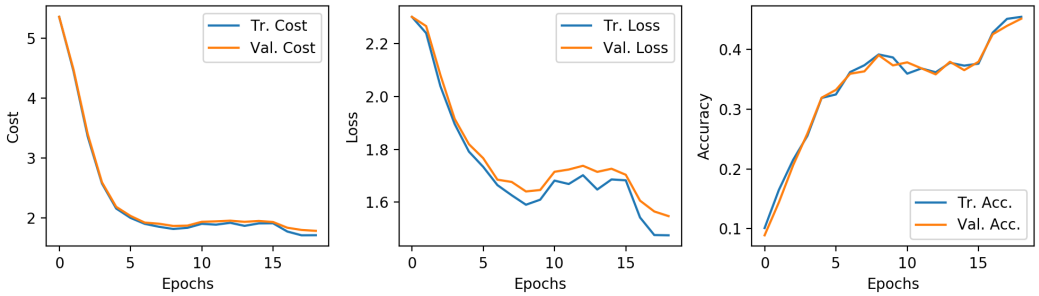

Figure 2: Cost, Loss and Accuracy plots for the hyper-parameter settings: $\eta_{min} = 1e-5, \eta_{max} = 1e-1, \lambda = 0.005, n_s = 2250$. and a batch size of 100 with 19 epochs, without batch normalisation (9 layers).

Accuracy for 9 layers

Training Accuracy: 0.4541
Validation Accuracy: 0.4510
Test Accuracy: 0.4220

The effect that a lack of batch normalisation has is not very clear for the network with three layers as shown in figure 1. However, it is much more clear for a deeper network. The performance of the model with 9 layers is much degraded from its more shallow variant, with a performance of almost ten percentage points worse.

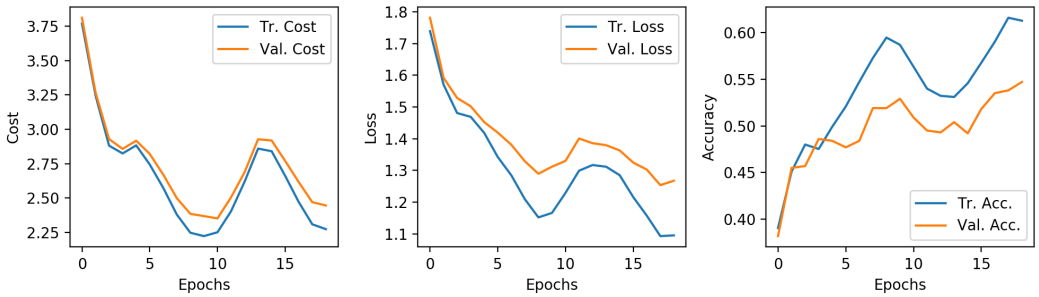

Figure 3: Cost, Loss and Accuracy plots for the hyper-parameter settings: $\eta_{min} = 1e-5, \eta_{max} = 1e-1, \lambda = 0.005, n_s = 2250$. and a batch size of 100 with 19 epochs, with batch normalisation (3 layers).

With Batch Normalisation

Accuracy for 3 layers

Training Accuracy: 0.6126
Validation Accuracy: 0.547
Test Accuracy: 0.5332

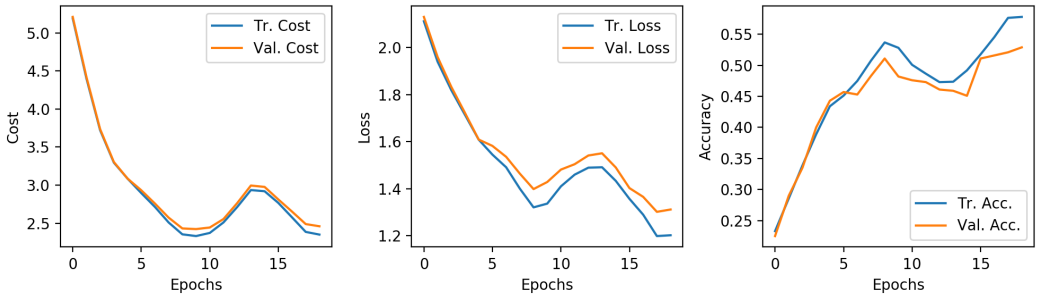

Figure 4: Cost, Loss and Accuracy plots for the hyper-parameter settings: $\eta_{min} = 1e-5, \eta_{max} = 1e-1, \lambda = 0.005, n_s = 2250$. and a batch size of 100 with 19 epochs, with batch normalisation (9 layers).

Accuracy for 9 layers

Training Accuracy: 0.578
Validation Accuracy: 0.529
Test Accuracy: 0.5132

Here we can see the impact that batch normalisation has, even for a good initialisation scheme like He-initialisation. For a more shallow model the improvements are relatively minor and not particularly obvious, with a slight improvement to performance. However, for a deeper model with 9 layers the improvment is dramatic, with an increase in testing performance if almost ten percentage points.

Coarse Grid Search

The coarse grid search was performed on a logarithmic scale. 50 values were sampled in the interval $[-5, -1]$ and were translated into λ-values by being used as exponents. In effect, the

interval for λ was $[10^{-5}, 10^{-1}]$. The three best λ-values and their corresponding accuracies are presented in the table below.

λ	Tr. Accuracy	Val. Accuracy
0.003759250129457098	0.6189	0.5510
0.007624470110464359	0.6057	0.5530
0.004179707368837784	0.6174	0.5630

Fine Grid Search

Since all the best λ-values were of the same magnitude, a fine grid search was performed around each of them by using the values as the center of an uniform distribution with a width of 0.001/4 on each side of the coarse value. In each interval around one of the coarse values 50 values were sampled and the three best of those and their accuracies are shown in the below table. The best out of all these λ-values is bolded.

lambda	Tr. Accuracy	Val. Accuracy
0.0035476760480130176	**0.6187**	**0.5630**
0.0035663241588970115	0.6192	0.5600
0.003886673973420486	0.6186	0.5540
0.004114165701805801	0.6137	0.5500
0.0042667992226196015	0.6181	0.5500
0.0043042609838174276	0.6150	0.5510
0.0076477182451726924	0.5979	0.5580
0.00775190684486371	0.5998	0.5570
0.007789753915408104	0.6056	0.5600

The best model

Using the best λ-value found in the fine grid search a model is trained on all of the available data for three cycles of the cyclic learning rate (28 epochs). Statistics of the performance of the model are calculated from 20 runs of 3 training cycles each.

Accuracy of best 3-layer model

Training accuracy: 0.6342 ± 0.0024
Validation accuracy: 0.5496 ± 0.0082
Test accuracy: 0.5366 ± 0.0029

Sensitivity to initial conditions - $\sigma = 1e-1$

This section deals with the sensitivity to initial conditions. The titular sigma was used as the standard deviation for a zero-mean normal distribution from which all parameter values were initialised by sampling. The models were then trained for 19 epochs each. For all models the same hyper-parameter values were used, being that $/eta_min = 1e-5, \eta_max = 1e-1, \lambda = 0.005, n_s = 2250$ with a batch size of 100. The only difference were the amount of layers and if batch normalisation was used or not.

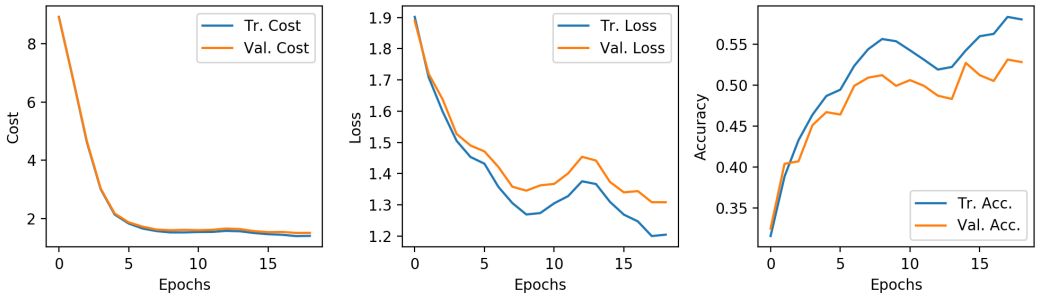

Figure 5: Cost, Loss and Accuracy plots for a 3-layer model with no batch normalisation for $\sigma = 1e-1$.

Accuracy for 3 layers with no BN

Training Accuracy: 0.58
Validation Accuracy: 0.528
Test Accuracy: 0.5245

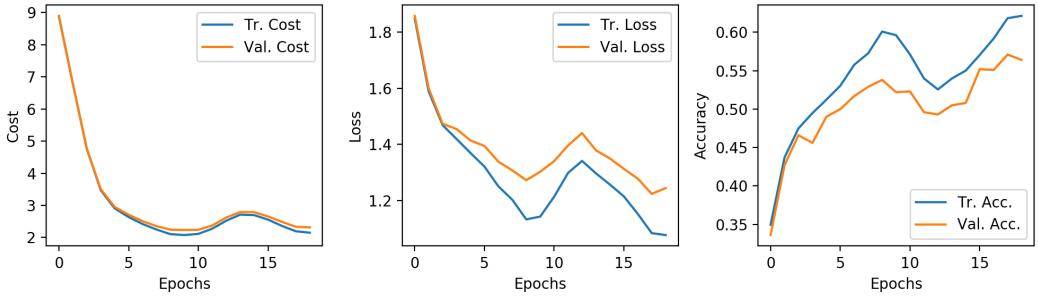

Figure 6: Cost, Loss and Accuracy plots for a 3-layer model with batch normalisation for $\sigma = 1e-1$.

Accuracy for 3 layers with BN

Training Accuracy: 0.6213
Validation Accuracy: 0.564
Test Accuracy: 0.5389

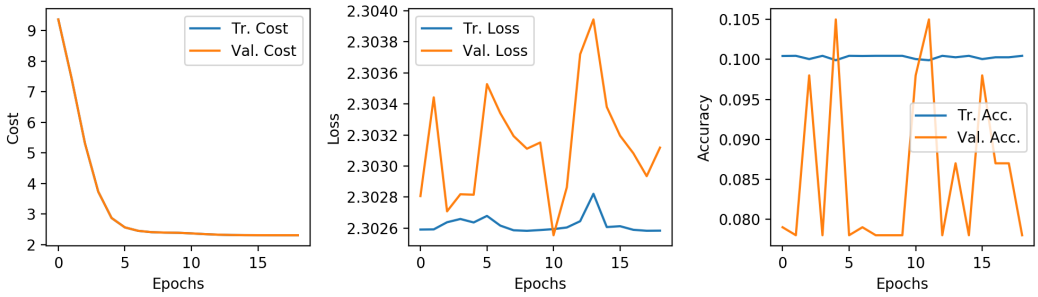

Figure 7: Cost, Loss and Accuracy plots for a 9-layer model with no batch normalisation for $\sigma = 1e-1$.

Accuracy for 9 layers with no BN

Training Accuracy: 0.1004
Validation Accuracy: 0.078
Test Accuracy: 0.1

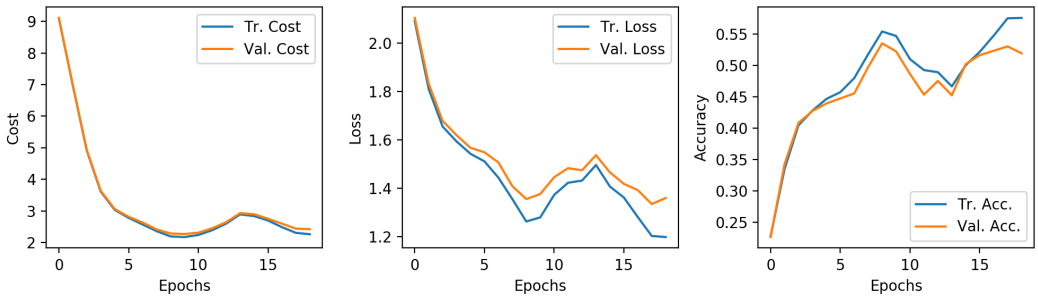

Figure 8: Cost, Loss and Accuracy plots for a 9-layer model with batch normalisation for $\sigma = 1e-1$.

Accuracy for 9 layers with BN

Training Accuracy: 0.5754
Validation Accuracy: 0.519
Test Accuracy: 0.511

Sensitivity to initial conditions - $\sigma = 1e-3$

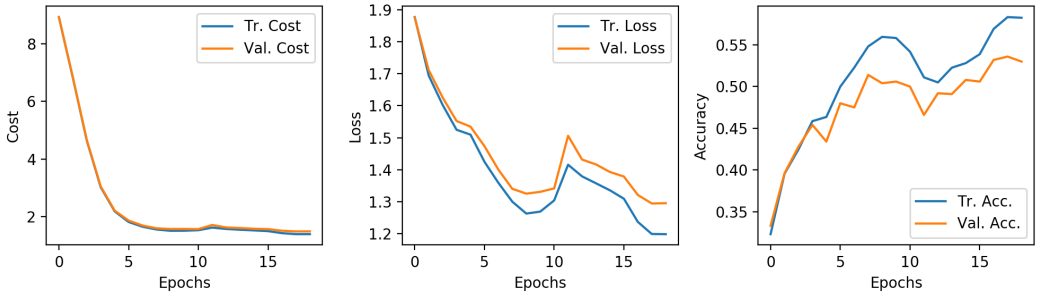

Figure 9: Cost, Loss and Accuracy plots for a 3-layer model with no batch normalisation for $\sigma = 1e-3$.

Accuracy for 3 layers with no BN

Training Accuracy: 0.5826
Validation Accuracy: 0.53
Test Accuracy: 0.5336

Accuracy for 3 layers with BN

Training Accuracy: 0.6153 Validation Accuracy: 0.536 Test Accuracy: 0.5297

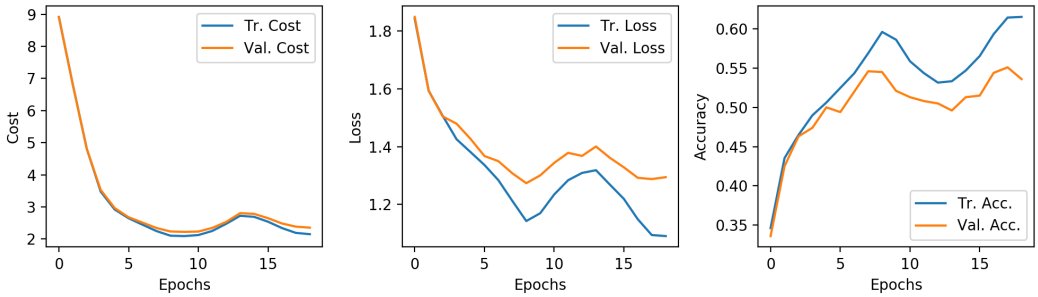

Figure 10: Cost, Loss and Accuracy plots for a 3-layer model with batch normalisation for $\sigma = 1e{-}3$.

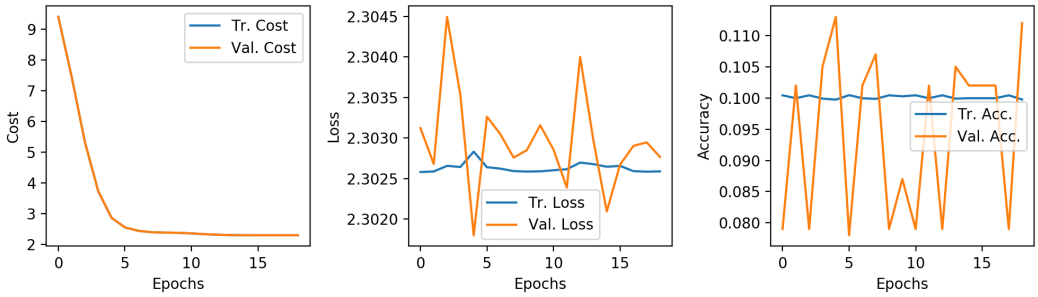

Figure 11: Cost, Loss and Accuracy plots for a 9-layer model with no batch normalisation for $\sigma = 1e-3$.

Accuracy for 9 layers with no BN

Training Accuracy: 0.0998
Validation Accuracy: 0.112
Test Accuracy: 0.1

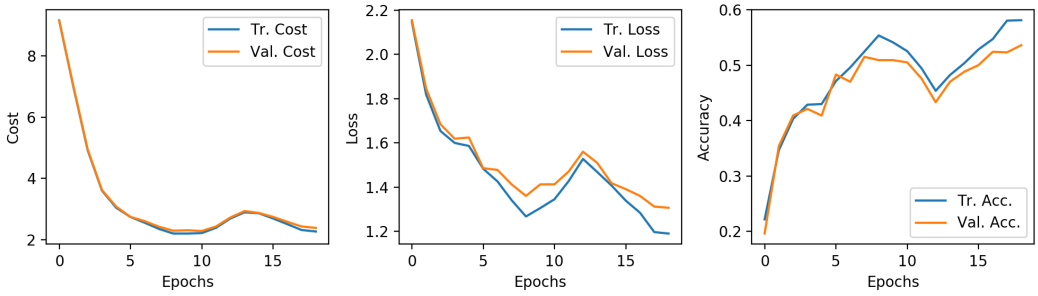

Figure 12: Cost, Loss and Accuracy plots for a 9-layer model with batch normalisation for $\sigma = 1e{-}3$.

Accuracy for 9 layers with BN

Training Accuracy: 0.5811
Validation Accuracy: 0.536
Test Accuracy: 0.5163

Sensitivity to initial conditions - $\sigma = 1e-4$

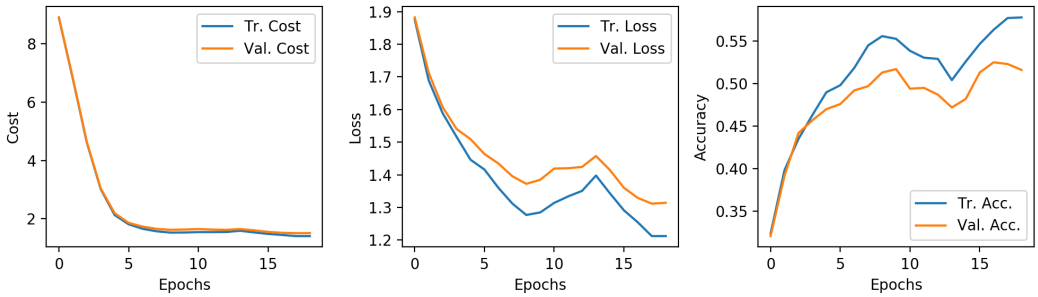

Figure 13: Cost, Loss and Accuracy plots for a 3-layer model with no batch normalisation for $\sigma = 1e-4$.

Accuracy for 3 layers with no BN

Training Accuracy: 0.5777
Validation Accuracy: 0.516
Test Accuracy: 0.5237

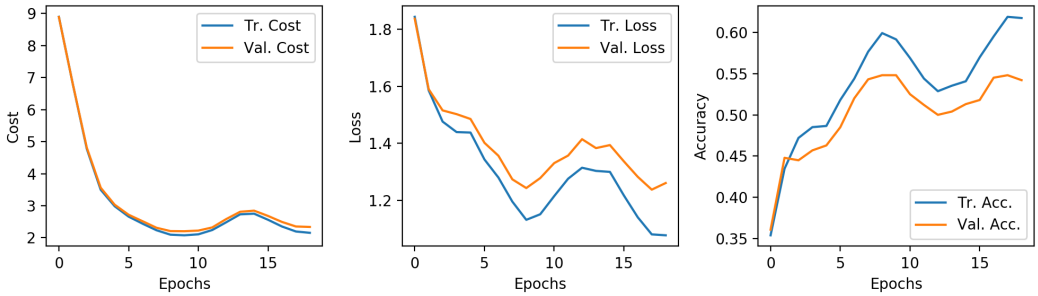

Figure 14: Cost, Loss and Accuracy plots for a 3-layer model with batch normalisation for $\sigma = 1e-4$.

Accuracy for 3 layers with BN

Training Accuracy: 0.6172
Validation Accuracy: 0.542
Test Accuracy: 0.5298

Accuracy for 9 layers with no BN

Training Accuracy: 0.1004
Validation Accuracy: 0.079
Test Accuracy: 0.1

Accuracy for 9 layers with BN

Training Accuracy: 0.575
Validation Accuracy: 0.522

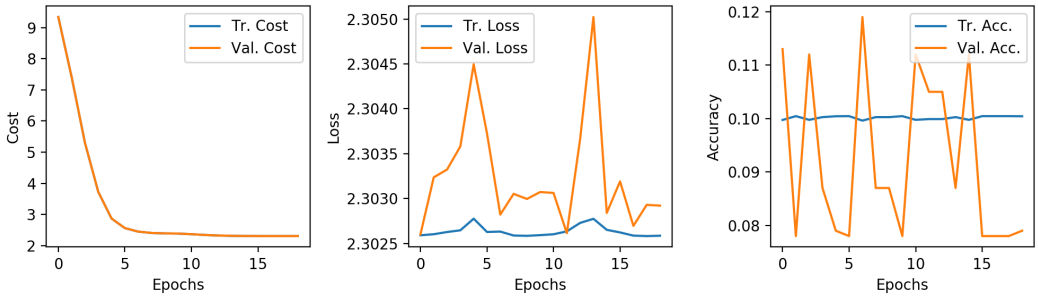

Figure 15: Cost, Loss and Accuracy plots for a 9-layer model with no batch normalisation for $\sigma = 1e-4$.

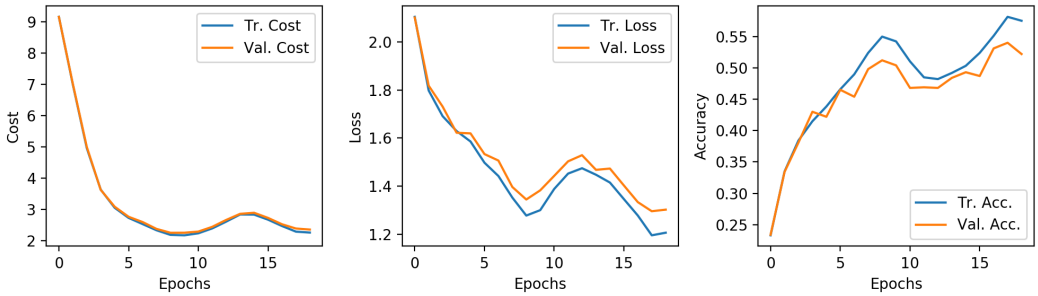

Figure 16: Cost, Loss and Accuracy plots for a 9-layer model with batch normalisation for $\sigma = 1e-4$.

Test Accuracy: 0.5145

We can see that employing batch normalisation does not have a huge impact on more shallow models, since those models do not seem to be as sensitive to initial conditions. However, on deeper models it has a huge impact. Regardless of the value for σ used, the 9-layer model ends up being no better at predicting the class of the images than pure chance without batch normalisation. However, with batch normalisation the models are approximately as good as the 3-layer models. As for why they are not better, it might be that their more advanced structure does not help with this problem, or that they would need more parameters to do so.

3 Conclusion

As a conclusion, the usage of batch normalisation seems to be a requirement for training deeper models. Without it, the models degenerate to being little better at predicting than a model just picking a class at random. Furthermore, batch normalisation does seem to have a slight positive impact on performance even on more shallow models, although the difference is not as obvious. Strangely it seems that having a deeper model does not necessarily mean that higher performance is reached. However, this might be due to the fact that the best lambda value is adapted to a 3-layer model. Furthermore, the 9-layer model is created specifically to have as many parameters as the corresponding 3-layer model, and it might be that having so few parameter in some layers is ineffective. Deeper models seem to be incredibly sensitive to initial conditions and batch normalisation is a nice tool to deal with the issue.