Assignment Three DD2424

For this assignment the goal is to build, train and test a k-layer neural network which should be able to classify images. The dataset used for training and testing is the CIFAR-10 image dataset.

Gradient Computations for a K-layer network

In order to determine the state and performance of my gradient computations, I have compared them with that of a more accurate, numerical method of calculating the gradients. The evaluation is done using a relative error which can be seen in equation 1.

$$\frac{|g_a - g_n|}{max(eps, |g_a| + |g_n|)} \tag{1}$$

The results of the calculated differences between computing the gradients numerically and analytically can be seen in figure 1. Since the differences are sufficiently low, we can conclude that the computations this far are bug free.

```
grad_W difference in layer  1  =  3.2234163189356575e-10
grad_W difference in layer  2  =  2.977828361688659e-10
grad_W difference in layer  3  =  3.110894519009824e-10
grad_W difference in layer  4  =  1.5194166686578661e-09

 grad_W mean difference =  8.1687686287376e-10

grad_b difference in layer  1  =  1.2098505344348281e-09
grad_b difference in layer  2  =  4.385252796376693e-09
grad_b difference in layer  3  =  1.609237765771987e-08
grad_b difference in layer  4  =  4.4997783276157223e-07

 grad_b mean difference =  1.5722177125003454e-07
```

Figure 1: Gradient computations comparison in a 4-layer network

Evolution of the Loss

3-Layer Network

For the next assignment I trained a 3-layer network with 50 nodes in the first two hidden layers. The cyclic learning rate shifted between 1e-1 and 1e-5. λ was set to 0.005. Training took part during two cycles of training with a batch size set to 100. Calculating the loss and accuracy of the network without using batch normalization gave a prediction rate on the test set of 52.38%. The plot for the loss computation of the 3-layer network can be found in figure 2.

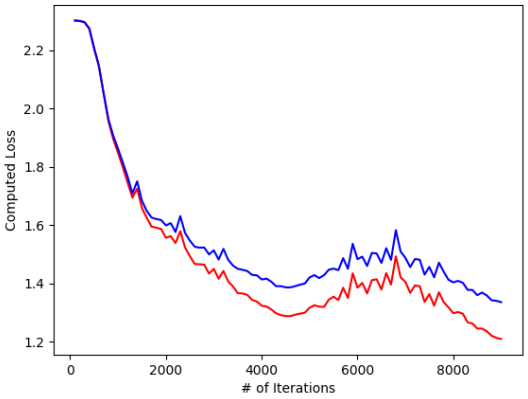

Figure 2: Computed loss for a 3 layer network without Batch Normalization

Using the exact same hyper parameters I also trained a 3-layer network with the same number of hidden nodes but this time using batch normalization. This resulted in a prediction rate of 51.13% which was slightly lower than without using batch normalization. The plot for the network can be seen in figure 3.

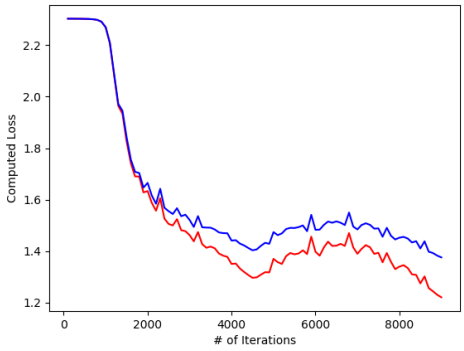

Figure 3: Computed loss for a 3 layer network with Batch Normalization

9-Layer Network

The evolution of loss for a 9-Layer network using the same hyper-parameters as previously and without batch normalization can be seen in figure 4. The prediction accuracy of the model when more layers were introduced reduced

drastically to 40.4%.

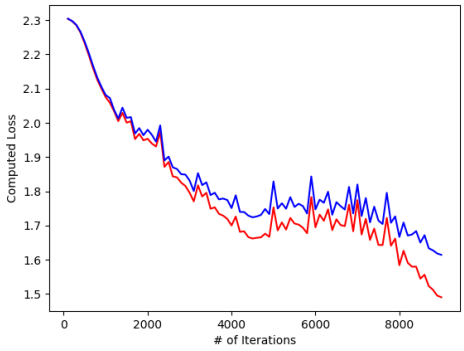

Figure 4: Computed loss for a 9 layer network without Batch Normalization

In figure 5 we can see a 9-Layer network with batch normalization introduced. This model had an accuracy of 47.81% which is a significant improvement to that of the previously mentioned model in figure 4. The advantages of Batch Normalization improves training when using more layers in the models, the evolution of the loss is a lot more stable and results in a lower final loss. What can also be seen in figure 5 by studying and comparing the plots is that the training becomes more stable which is stated in the assignment as one of the big pros of batch normalization.

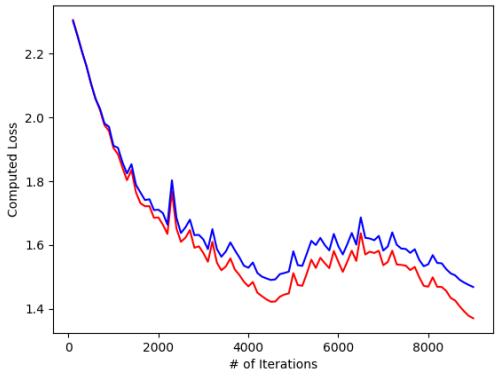

Figure 5: Computed loss for a 9 layer network with Batch Normalization

Lambda Search

To search for a good value for lambda i initially started with a broad test to be able to narrow my lambda search even further. The results of and values initially tested can be seen in table 1.

λ	Accuracy (%)
1e-1	41.63%
1e-2	51.21%
1e-3	50.77%
1e-4	49.95%
1e-5	48.47%
1e-6	48.95%

Table 1: Initial Lambda testing values and results

Looking at table 1 it shows that a good value for lambda is somewhere around 1e-2 and 1e-3. To further explore a good value i centered my search around these roughs initial estimates. I did this by choosing 10 random values between 1e-2 and 1e-3. The value which performed best were 8.88e-3, it managed to reach a prediction rate of 52.53% on the test dataset.

Sensitive to Initialization

For this assignment I trained a 3-Layer network with 50 Nodes in the first two hidden layers. The hyper parameters are the same as in previous assignments with the exception for λ for which i used my findings in the previous assignment, 8.88e-3. When σ was set to 1e-1 the loss which can be seen in figure 6 and 7 is quiet similar, as were their accuracy scores.

When I set σ to 1e-4 however, then the effects initialization has on training a model with batch normalization vs without becomes a lot more clear. In figure 8 and 9 we can see the results of the training process with vs without batch normalization. The model seen in 9 balances out a lot faster, the other model in figure 8 can be seen standing still for almost 6000 iterations and only had a 43.98% accuracy on the test set whereas the model in figure 9 at least managed to reach a 48.81% accuracy. Therefore I believe it is fair to conclude that a model which is trained using batch normalization is not as sensitive to the weight initialization process itself.

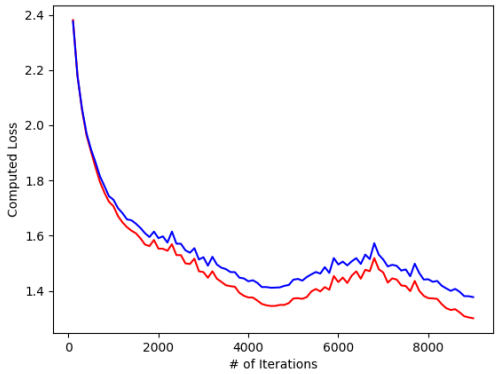

Figure 6: 3-Layer network loss computation with σ set to 1e-1 and without batch normalization

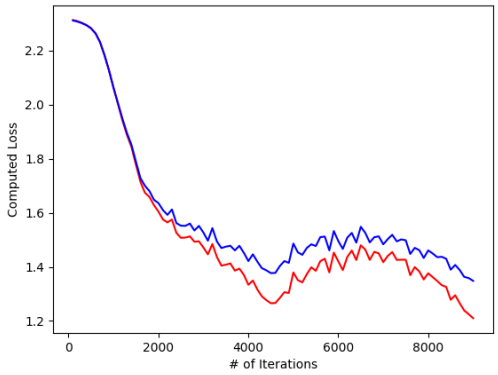

Figure 7: 3-Layer network loss computation with σ set to 1e-1 and with batch normalization

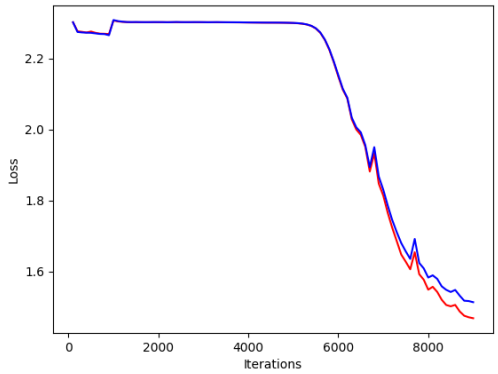

Figure 8: 3-Layer network loss computation with σ set to 1e-4 and without batch normalization

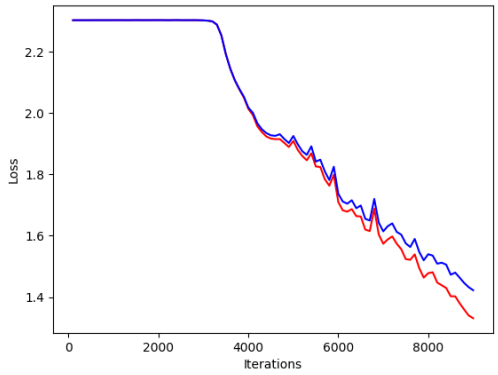

Figure 9: 3-Layer network loss computation with σ set to 1e-4 and with batch normalization