[DD2424] Assignment 3

Victor Hansjons Vegeborn

June 19, 2018

Contents

1 Gradient Check **2**

1.1 1-Layer Network 2

1.2 2-Layer Network 2

1.3 3-Layer Network 3

1.4 4-Layer Network 4

1.5 Overfitting 4-Layer network 4

1.6 Conclusion 5

2 Evolution of loss function **5**

3 Hyper-parameter Search **6**

3.1 First fine-search 6

3.2 Second fine-search 6

3.3 Final Network Results 7

3.4 Discussion 8

4 2-Layer Network and Batch Normalization **8**

4.1 Small Learning Rate 8

4.2 Medium Learning Rate 9

4.3 Large Learning Rate 10

1 Gradient Check

The gradient checks for this assignment was done using an implementation from the provided ComputeGradsNumSlow.m algorithm from the course. The relative error was calculated on each element in the weights and bias terms. The step size was set to h=1e-05 and the denominator term ϵ was also set to 1e-05. All weights was initialized with He initialization and the bias terms was initialized to zero. The batch size was set to 100 for all gradient check experiments.

I choose to extract the maximum relative error from the collection of relative errors from each element in each layer, while also computing the mean (μ) and the variation (σ^2) for the same collections. This was done to get a better statistical overview of the gradient errors.

1.1 1-Layer Network

The architecture of this network:

```
Input -> [FC -> BN -> ReLU]*0 -> FC -> Softmax -> Output
Nodes:  3072 -> 10
```

Note: As batch normalization is only applied on the hidden layers, this network did not utilize any batch normalization.

Gradient	Max	μ	σ^2
b_1	7.217e-10	2.524e-10	5.101e-20
W_1	8.027e-07	6.992e-09	1.420e-15

Table 1: $\lambda = 0$

Gradient	Max	μ	σ^2
b_1	7.217e-10	2.524e-10	5.101e-20
W_1	1.347e-06	1.026e-08	5.053e-15

Table 2: $\lambda = 0.01$

1.2 2-Layer Network

The architecture of this network:

```
Input -> [FC -> BN -> ReLU]*1 -> FC -> Softmax -> Output
Nodes:  3072 -> 50 -> 10
```

Gradient	Max	μ	σ^2
b_1	2.220e-06	1.332e-07	2.781e-13
b_2	6.515e-09	1.414e-09	5.230e-18
W_1	2.588e-03	9.967e-07	2.009e-09
W_2	8.061e-08	1.149e-09	2.128e-17

Table 3: $\lambda = 0$

Gradient	Max	μ	σ^2
b_1	2.220e-06	1.332e-07	2.781e-13
b_2	6.515e-09	1.414e-09	5.230e-18
W_1	2.590e-03	9.983e-07	2.010e-09
W_2	5.416e-08	1.098e-09	1.292e-17

Table 4: $\lambda = 0.01$

1.3 3-Layer Network

The architecture of this network:

```
Input -> [FC -> BN -> ReLU]*2 -> FC -> Softmax -> Output
Nodes:  3072 -> 50 -> 30 -> 10
```

Gradient	Max	μ	σ^2
b_1	2.320e-11	6.776e-12	3.081e-23
b_2	1.459e-12	5.629e-13	1.784e-25
b_3	6.474e-08	6.701e-09	3.743e-16
W_1	7.273e-03	2.151e-06	1.201e-08
W_2	6.082e-07	3.362e-09	4.705e-16
W_3	2.100e-08	1.157e-09	6.252e-18

Table 5: $\lambda = 0$

Gradient	Max	μ	σ^2
b_1	2.320e-11	6.776e-12	3.081e-23
b_2	1.459e-12	5.629e-13	1.784e-25
b_3	6.474e-08	6.701e-09	3.743e-16
W_1	7.243e-03	2.145e-06	1.192e-08
W_2	1.282e-06	3.994e-09	1.373e-15
W_3	2.211e-07	3.311e-09	3.511e-16

Table 6: $\lambda = 0.01$

1.4 4-Layer Network

The architecture of this network:

```
Input -> [FC -> BN -> ReLU]*3 -> FC -> Softmax -> Output
Nodes:  3072 -> 50 -> 30 -> 20 -> 10
```

Gradient	Max	μ	σ^2
b_1	3.464e-11	7.416e-12	5.738e-23
b_2	3.025e-12	6.414e-13	5.633e-25
b_3	2.512e-12	1.027e-12	6.755e-25
b_4	6.639e-10	2.647e-10	4.253e-20
W_1	4.776e-03	1.412e-06	5.172e-09
W_2	2.814e-07	2.717e-09	1.277e-16
W_3	1.425e-07	2.551e-09	1.082e-16
W_4	1.641e-08	8.478e-10	5.098e-18

Table 7: $\lambda = 0$

Gradient	Max	μ	σ^2
b_1	3.464e-11	7.416e-12	5.738e-23
b_2	3.025e-12	6.414e-13	5.633e-25
b_3	2.512e-12	1.027e-12	6.755e-25
b_4	6.697e-10	2.793e-10	4.157e-20
W_1	4.757e-03	1.410e-06	5.151e-09
W_2	4.240e-07	3.087e-09	2.447e-16
W_3	7.474e-08	2.100e-09	3.834e-17
W_4	1.399e-08	1.276e-09	8.331e-18

Table 8: $\lambda = 0.01$

1.5 Overfitting 4-Layer network

Another approach to evaluate the gradient calculations is to overfit a network on some small set of samples. For this experiment the following network architecture was used: `Input -> [FC -> BN -> ReLU]*3 -> FC -> Softmax -> Output` or in terms of nodes: `3072 -> 50 -> 30 -> 20 -> 10`. I trained the network for 100 epochs over a batch of 50 samples. The learning rate was set to $\eta = 0.02$ with a decay rate of 0.99. The regularization term was set to $\lambda = 0$. Batch normalization was used, as well as momentum set at $\rho = 0.8$. Additionally, He initialization was utilized.

These settings gave a 100% accuracy on the training data set of 50 samples. Validation loss and accuracy has been omitted as they are irrelevant when intentionally overfitting networks.

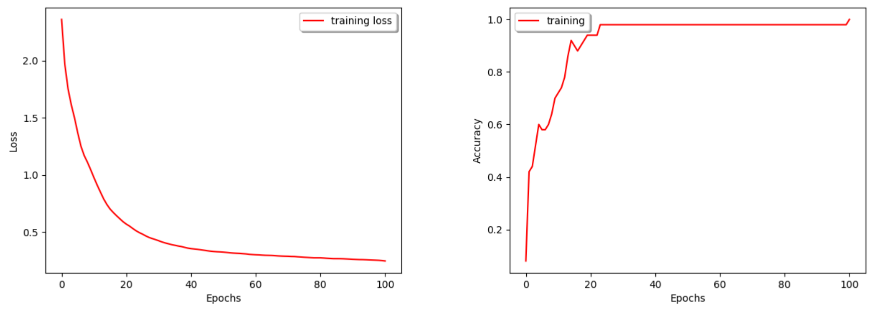


Figure 1: Overfitting a 4-layer network on 50 samples. To the left is the training loss through out 100 epochs. On the right is the accuracy on the training set over the same 100 epochs.

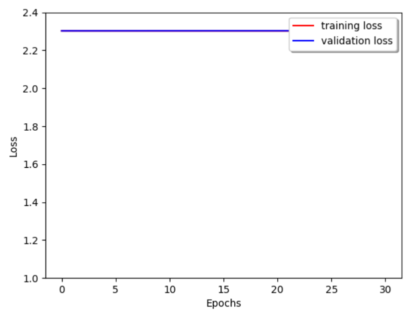
1.6 Conclusion

The max relative error for the weights grow when adding more layers, which is expected as errors are also back propagated. This is also true for the mean and the variance. However, the errors are fairly small. Moreover, being able to overfit a 4-layer network implies that the gradient calculations are correct.

2 Evolution of loss function

The network architecture in this experiment was a 3-layer network: `3072 -> 50 -> 30 -> 10`. During the two executions of the network only batch normalization was toggled on and off. The following were the initialisation of the network: Learning rate $\eta = 0.018$, $\lambda = 1.0e-04$, Momentum $\rho = 0.1$, numer of epochs 30, Initialization of the weights with random normal distribution with a standard deviation set to 0.001.

When training without batch normalization with these hyper-parameters set, the difference in loss from each epoch is so small and cannot be seen in the graphs below. The effect of batch normalization while training the same network is present from the first epoch. See the evolution of the loss function in Figure 2.



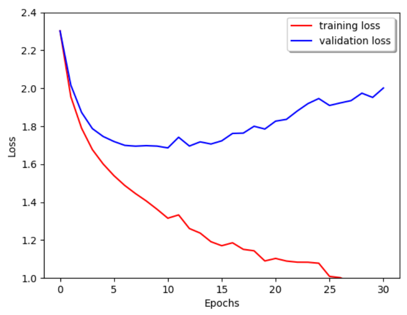


Figure 2: Left: No batch normalization. Right: With batch normalization.

3 Hyper-parameter Search

A coarse-to-fine search for the hyper-parameters η and λ was performed on the same network as in section 2 above. For the other parameters the following was set throughout each search: Momentum $\rho = 0.7$, batch size set to 100, He initialization, and batch normalization. The data consisted of: training: `data_batch_1.mat`, validation: `data_batch_2.mat`, and test: `test_batch.mat`.

An initial coarse search was performed and resulted in fine search ranges for $\eta \in (10^{-3}, 0.8)$ and $\lambda \in (10^{-3}, 10^{-6})$.

3.1 First fine-search

The first fine-search was performed with 300 uniformly sampled pairs in the ranges stated above. Moreover, the number of epochs for each training iteration was set to 9.

Validation accuracy	η	λ
0.4217	0.13718638	1.82552225e-06
0.4192	0.12894386	1.7233738e-05
0.4180	0.13380538	1.08343634e-06

Table 9: Results from first fine-search

3.2 Second fine-search

New ranges was set for this search based on the results in the first search iteration. The new ranges was set to $\eta \in (10^{-1.7}, 10^{0})$ and $\lambda \in (10^{-5.3}, 10^{-3.6})$. For this fine-search another 300 iterations was done per random uniformly sampled pairs of hyper-parameters. However, the number of epochs was extended to 14.

Validation accuracy	η	λ
0.4134	0.05282343	6.62421401e-06
0.4125	0.06745809	9.72563126e-05
0.4122	0.03216179	0.00010803

Table 10: Results from second fine-search

3.3 Final Network Results

The second fine-search did not result in any better performing networks, as can be seen from comparing table 9 and 10. However, as the number of epochs was extended to 14, I decided to utilize the top 3 results from the second fine-search.

For the final training I used 49000 samples for the training data and 1000 samples for the validation data. The final networks was trained over 20 epochs. All other parameters was set as stated in the begining of section 3. From the top 3 resulting hyper-parameters found, the third pair resulted in the best accuracy on the test set. The final accuracy was 49,89%.

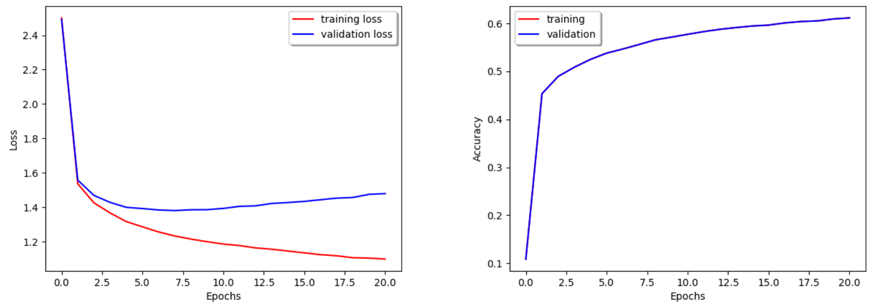


Figure 3: Test set accuracy: 49,22%. $\eta = 0.05282343$, $\lambda = 6.62421401e-06$

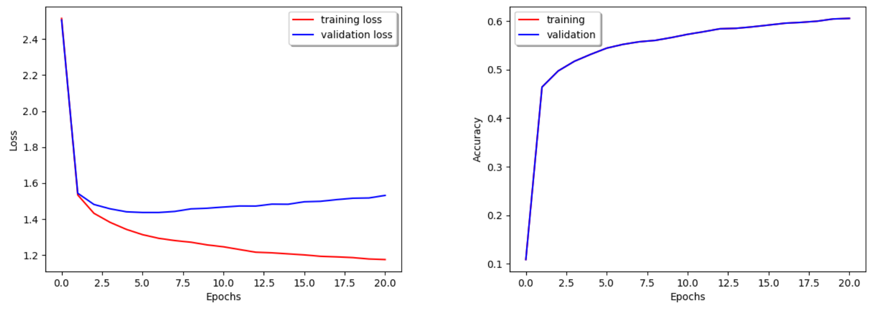
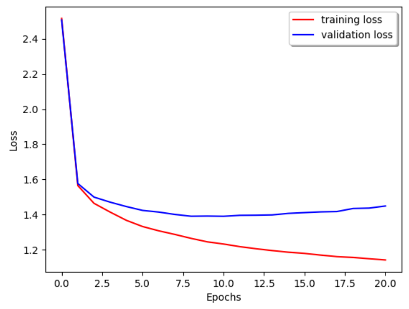


Figure 4: Test set accuracy: 49,38%. $\eta = 0.06745809$, $\lambda = 9.72563126e-05$



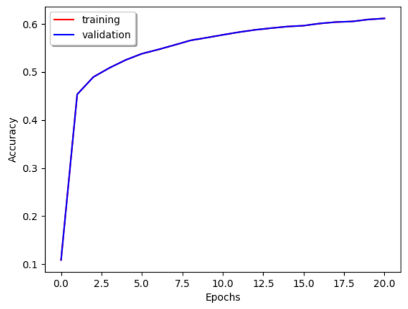


Figure 5: Test set accuracy: 49,89%. $\eta = 0.03216179$, $\lambda = 0.00010803$

3.4 Discussion

There are more overfitting present in the training on the first and second best parameter settings, as can be seen in Figure 3 and 4. The span between the loss curves of the training and validation is far greater than that of the third pair of settings (Figure 5). Perhaps the over-fitted results would benefit from more regularization, which was very much greater in the third and final result.

It should be noted that the loss curve show a initial loss of above 2.4 before any training has occurred. This behaviour was investigated further and was found to be caused by the He initialization as the behaviour was not present during initialization with a standard deviation of 0.001 in the weights matrix.

4 2-Layer Network and Batch Normalization

To further investigate the impact batch normalization has on training and whether it is possible to obtain the same convergence for a network, setup as in assignment 2, but with fewer epochs. The architecture of this network:

```
Input -> [FC -> BN -> ReLU]*1 -> FC -> Softmax -> Output
Nodes:  3072 -> 50 -> 10
```

For each iteration of the experiment the momentum was set to $\rho = 0.8$ and the regularization $\lambda = 0$. The batch size was set to 100. The training data consisted of 49000 samples and the validation data consisted of 1000 samples. The test data was the content of the file `test_batch.mat`. Each training iteration lasted for 10 epochs.

4.1 Small Learning Rate

For the small experiment the learning rate was set to $\eta = 0.0005$.

	Train Accuracy	Test Accuracy
Batch Normalization	44,98%	42,30%
No Batch Normalization	20,51%	21,08%

Table 11: Results

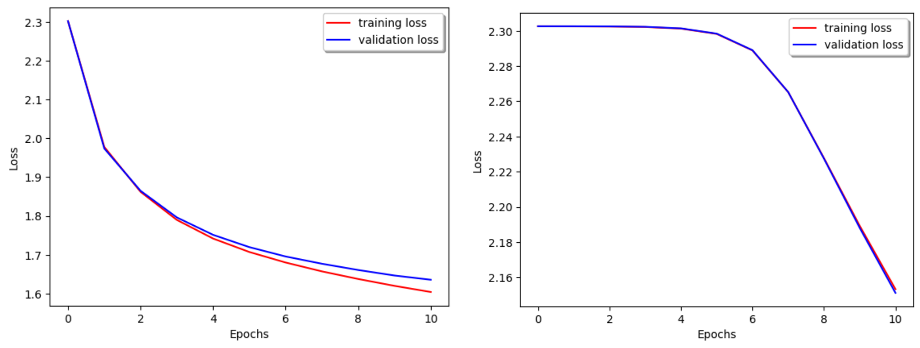


Figure 6: Training and validation loss. Left: With batch normalization. Right: No batch normalization.

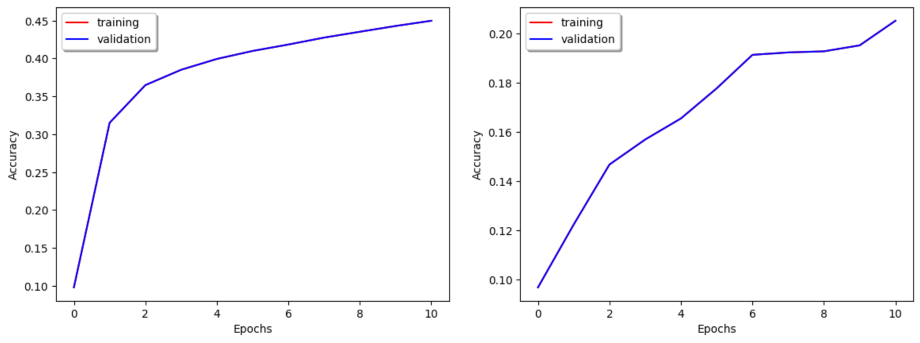


Figure 7: Training and validation accuracy: Left: With batch normalization. Right: No batch normalization

4.2 Medium Learning Rate

For the medium experiment the learning rate was set to $\eta = 0.03$.

	Train Accuracy	Test Accuracy
Batch Normalization	57,13%	49,67%
No Batch Normalization	57,59%	50,45%

Table 12: Results

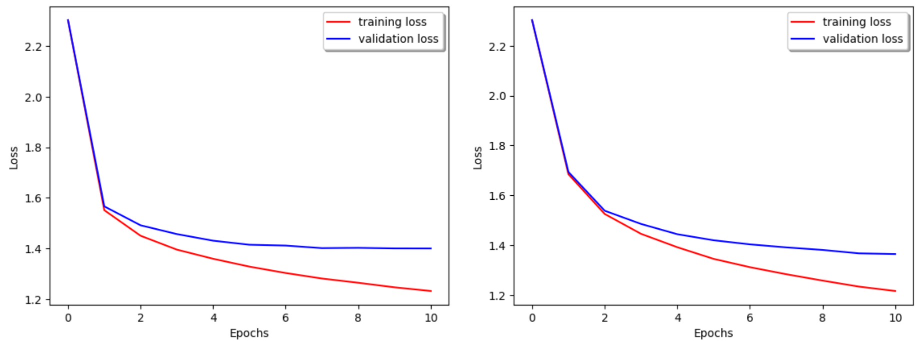


Figure 8: Training and validation loss. Left: With batch normalization. Right: No batch normalization.

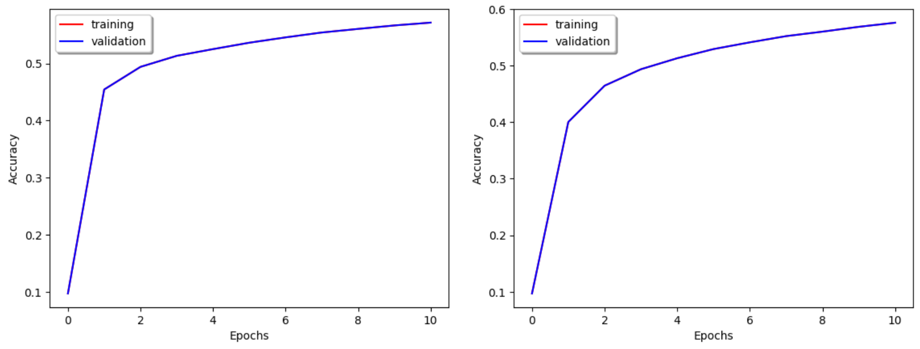


Figure 9: Training and validation accuracy: Left: With batch normalization. Right: No batch normalization

4.3 Large Learning Rate

For the large experiment the learning rate was set to $\eta = 0.8$. Without batch normalization the network could not be trained due to some unknown error in the gradient calculations.

	Train Accuracy	Test Accuracy
Batch Normalization	51,26%	44,92%
No Batch Normalization	N/A	N/A

Table 13: Results

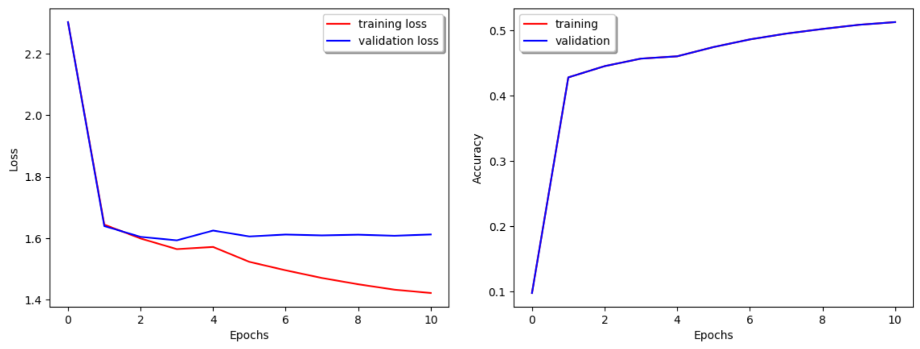


Figure 10: Left: train and validation loss.
Right: train and validation accuracy