Deep Learning (DD2424) Assignment 3

Badi Mirzai

July 2021

1 Intro

In this assignment, we generalize to train a k-layer neural network on the CIFAR-10 data set. In addition to cyclic learning rates and grid search for finding the optimal lambda regularization, we also implement batch normalization and investigate its impact on the neural network's performance.

The input layer is 3072 pixeles values from each image with softmax as activation function for the output layer and arbitrarily nodes on the hidden layer, while the output layer is 10 nodes. The classification of the image is done by choosing the output with the highest predicted probability. The network is trained with gradient descent through back propagation of the cross entropy loss. The trained network achieved an average accuracy of 62.36% train dataset and a test average accuracy of 53.98%. The best test accuracy after training was 54.68% with the optimal lamda regularization and using batch normalization.

2 Gradient checks

The gradients were checked in a similar fashion to previous assignments. The first 10 images were used with the first 30 dimensions to test the gradient and to avoid computational wait.

2.1 Without batch normalization

First, the gradients were checked without batch normalization on a neural network with 3 layers (each hidden layer having 50 nodes). The results of the maximum absolute -and relative errors were the following:

Two layers [50] (no batch normalization):

1. For layer 1, W1: absolute error $1.8568 * 10^{-8}$ and $5.1147 * 10^{-8}$ in relative error.

2. For layer 1, b1: absolute error $1.8704 * 10^{-8}$ and $1.0274 * 10^{-7}$ in relative error.

3. For layer 2, W2: absolute error $3.0849 * 10^{-7}$ and $2.5763 * 10^{-7}$ in relative error.

4. For layer 2, b2: absolute error $6.1297 * 10^{-8}$ and $2.2939 * 10^{-7}$ in relative error.

Three layers [50, 50] (no batch normalization):

1. For layer 1, W1: absolute error $1.8835 * 10^{-8}$ and $5.2067 * 10^{-8}$ in relative error.

2. For layer 1, b1: absolute error $1.7858 * 10^{-8}$ and $5.9468 * 10^{-8}$ in relative error.

3. For layer 2, W2: absolute error $8.3394 * 10^{-8}$ and $7.7759 * 10^{-8}$ in relative error.

4. For layer 2, b2: absolute error $1.7240 * 10^{-8}$ and $7.1289 * 10^{-8}$ in relative error.

5. For layer 3, W3: absolute error $3.1924 * 10^{-7}$ and $2.5041 * 10^{-7}$ in relative error.

6. For layer 3, b3: absolute error $7.8099 * 10^{-8}$ and $1.2514 * 10^{-7}$ in relative error.

For 4 layers [50, 50, 50], we got the following errors (without batch normalization):

1. For layer 1, W1: absolute error $2.2804 * 10^{-8}$ and $9.6227 * 10^{-8}$ in relative error.

2. For layer 1, b1: absolute error $1.6704 * 10^{-8}$ and $1.1093 * 10^{-7}$ in relative error.

3. For layer 2, W2: absolute error $1.4600 * 10^{-7}$ and $2.1113 * 10^{-7}$ in relative error.

4. For layer 2, b2: absolute error $2.8382 * 10^{-8}$ and $1.3050 * 10^{-7}$ in relative error.

5. For layer 3, W3: absolute error $1.7933 * 10^{-7}$ and $2.2295 * 10^{-7}$ in relative error.

6. For layer 3, b3: absolute error $3.6952 * 10^{-8}$ and $2.3494 * 10^{-7}$ in relative error.

7. For layer 4, W4: absolute error $2.3685 * 10^{-7}$ and $3.7842 * 10^{-7}$ in relative error.

8. For layer 4, b4: absolute error $5.5615 * 10^{-8}$ and $3.8615 * 10^{-7}$ in relative error.

2.2 With batch normalization

Two layers [50] With batch normalization:

1. For layer 1, W1: absolute error $7.5788 * 10^{-7}$ and $7.6205 * 10^{-7}$ in relative error.
2. For layer 1, b1: absolute error $7.7716 * 10^{-17}$ and $7.7716 * 10^{-11}$ in relative error.
3. Gamma: absolute error $1.8210 * 10^{-8}$ and $6.53243558 * 10^{-8}$ in relative error.
4. Beta: absolute error $1.8668 * 10^{-8}$ and $8.0802 * 10^{-8}$ in relative error.
5. For layer 2, W2: absolute error $6.5885 * 10^{-8}$ and $1.6294 * 10^{-7}$ in relative error.
6. For layer 2, b2: absolute error $6.2724 * 10^{-8}$ and $2.4670 * 10^{-7}$ in relative error.

For 3 layer network [50, 50] with batch normalization we get the following errors:

1. For layer 1, W1: absolute error $4.6410 * 10^{-7}$ and $1.2548 * 10^{-6}$ in relative error.
2. For layer 1, b1: absolute error $8.8818 * 10^{-10}$ and $8.8818 * 10^{-4}$ in relative error.
3. For layer 1, Gamma: absolute error $3.9641 * 10^{-8}$ and $1.6758 * 10^{-7}$ in relative error.
4. For layer 1, Beta: absolute error $2.6652 * 10^{-8}$ and $1.3622 * 10^{-7}$ in relative error.
5. For layer 2, W2: absolute error $2.1959 * 10^{-7}$ and $3.9898 * 10^{-7}$ in relative error.
6. For layer 2, b2: absolute error $8.8818 * 10^{-10}$ and $8.8818 * 10^{-4}$ in relative error.
7. For layer 2, Gamma: absolute error $2.8518 * 10^{-8}$ and $7.7923 * 10^{-8}$ in relative error.
8. For layer 2, Beta: absolute error $2.7053 * 10^{-8}$ and $9.7440 * 10^{-8}$ in relative error.
9. For layer 3, W3: absolute error $8.8234 * 10^{-8}$ and $1.82701276 * 10^{-7}$ in relative error.

10. For layer 3, b3: absolute error $6.2084 * 10^{-8}$ and $1.7888 * 10^{-7}$ in relative error.

I am quite certain than the gradients were correctly calculated, based on the tests and results obtained all the absolute and relative errors seemed to be small enough. The gradients also seemes to be sufficiently correct, based on the later results in this report.

3 Evolution of loss function

3.1 3 Layers

Figure 1 and 2 shows the evolution of the loss fucntions and accuracies for the 3 layer network with 50 nodes on the two hidden layers. The network (both with and without batch normalization) was trained with the default settings $\lambda = 0.005$, $n_{epochs} = 20$ (2 cycles), $n_{batch} = 100$, $\eta_{min} = 1 * 10^{-5}$, $\eta_{max} = 0.1$, $n_s = 2250$.

3.2 9 Layers

Figure 3 and 4 shows the evolution of the loss functions and accuracy for the 9 layer network with [50, 30, 20, 20, 10, 10, 10, 10] nodes on each hidden layer. The network (both with and without batch normalization) was trained with the default setting $\lambda = 0.005$, $n_{epochs} = 20$ (2 cycles), $n_{batch} = 100$, $\eta_{min} = 1 * 10^{-5}$, $\eta_{max} = 0.1$, $n_s = 2250$.

4 Lambda search

The models was trained on the CIFAR-10 data-set, where mini-batches of size 100 was used. The training data was shuffled for each epoch, where the loss and accuracy was calculated each epoch for the training and validation data sets. The input data was pre-processed by first normalized to one (division by 255). The mean and variance of the training data was calculated and used as a way to center the training, test and validation input data for increasing the performance.

For the course random search, I uniformlly sampled different lambda values between 10^{-5} and 10^{-1} (see figure 5) and the corresponding best values is in Table 1. Here (for both the course search and fine search), we trained on 45000 images and used 5000 images for validation. We ran for 2 cycles, with $n_{epoch} = 10$, $\eta_{min} = 10^{-5}$, $\eta_{max} = 10^{-1}$, $n_{batch} = 100$, $n_s = 900$. After the course search, we repeated the lambda search on a narrower range of lambda values, see figure 6. The three best lambda values can be seen in Table 2. Finally, we trained the model on the best λ for 3 cycles, see Figure 8. The results showed that the validation accuracy reached up to 53%. Finally, a second fine search was done in order to find a better lambda, see Figure 7 and Table 3.

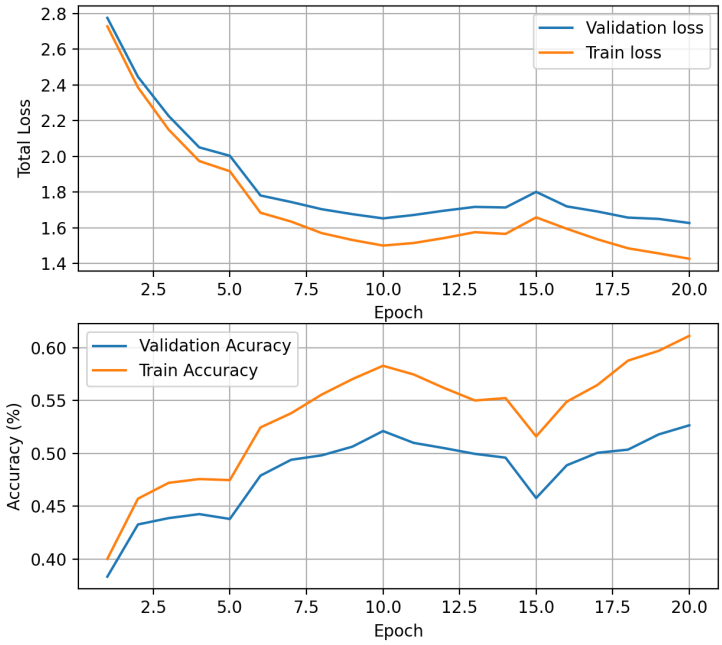


Figure 1: Loss functions and accuracy for cyclical learning rates on a 3 layer network without batch normalization. After training, the accuracy was 61.13% for the train dataset and 52.66% for the validation dataset.

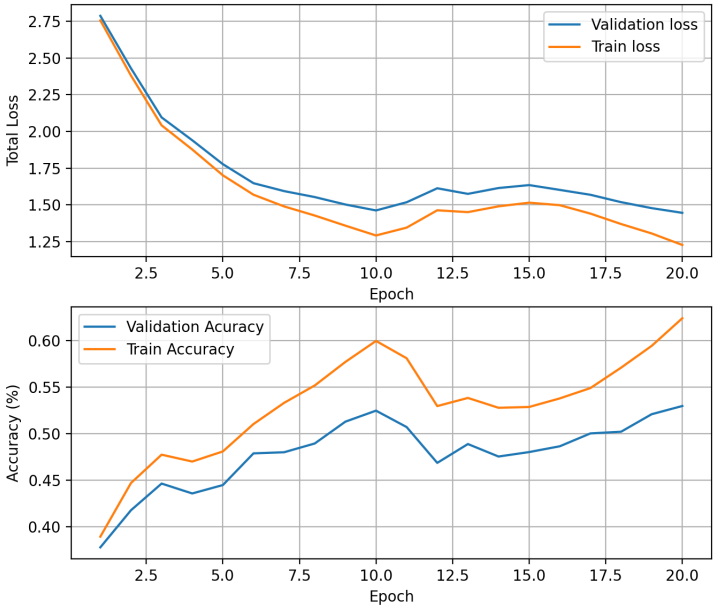


Figure 2: Loss functions and accuracy for cyclical learning rates on a 3 layer network using batch normalization. After training, the accuracy was 62.37% for the train dataset and 53.09% for the validation dataset.

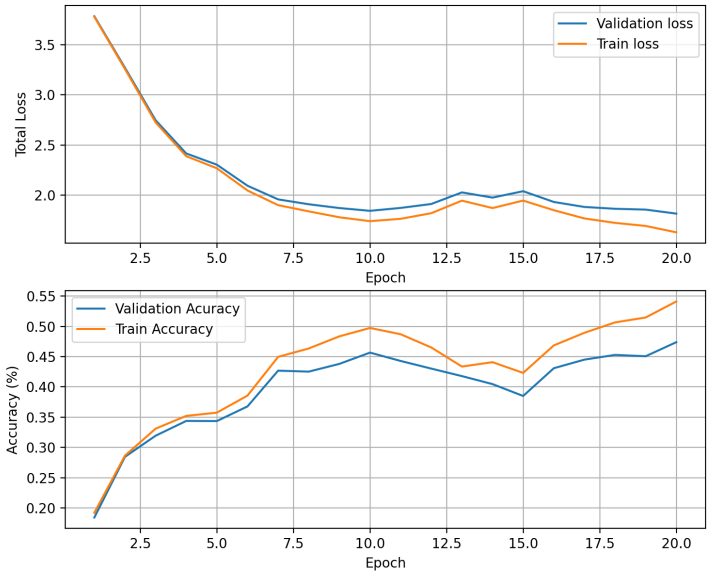


Figure 3: Loss functions and accuracy for cyclical learning rates on a 9 layer network without batch normalization. After training, the accuracy was 54.10% for the train dataset and 47.38% for the validation dataset.

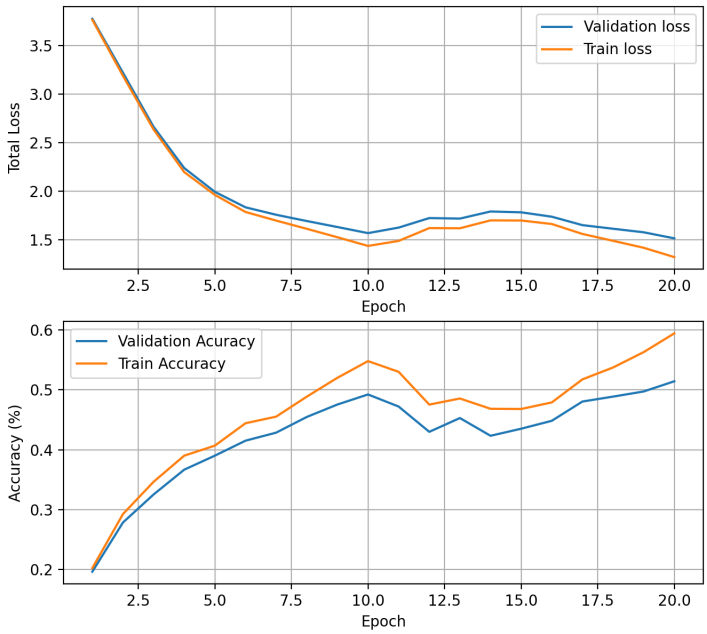


Figure 4: Loss functions and accuracy for cyclical learning rates on a 9 layer network using batch normalization. After training, the accuracy was 59.44% for the train dataset and 51.65% for the validation dataset.

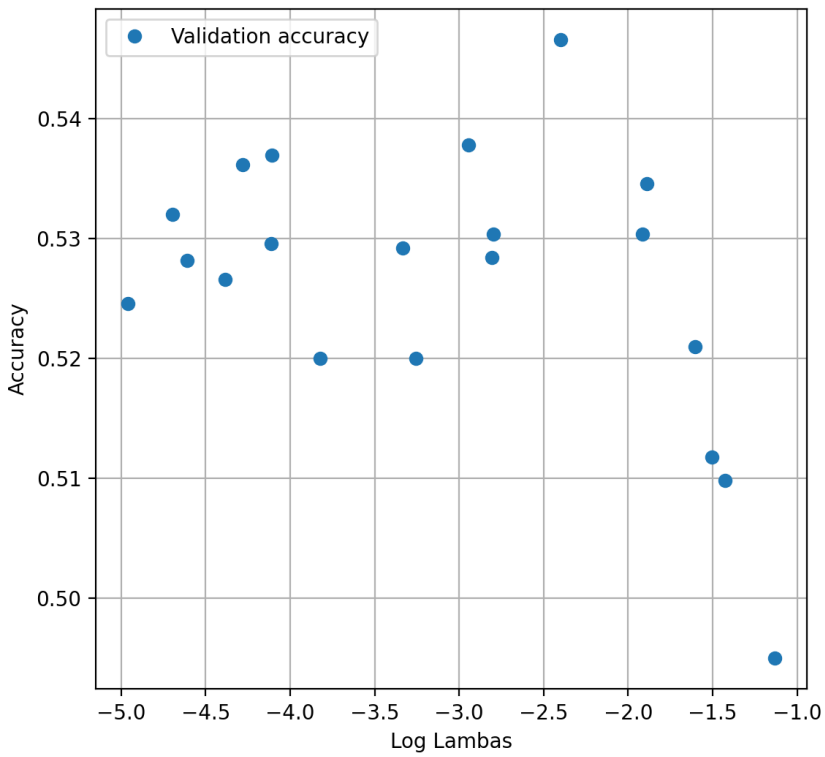


Figure 5: Validtion Accuracy over different values of the lambda regularization in the course search. Network was trained on all 5 datasets, with 5000 images as validation dataset.

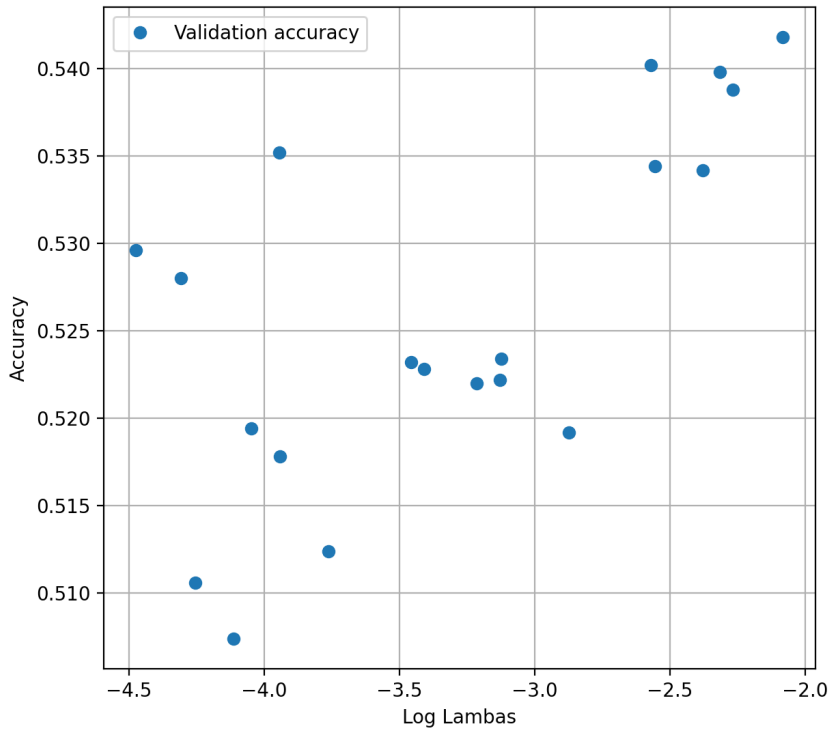


Figure 6: Validtion Accuracy over different values of the lambda regularization in the fine search. Network was trained on all 5 datasets, with 5000 images as validation dataset.

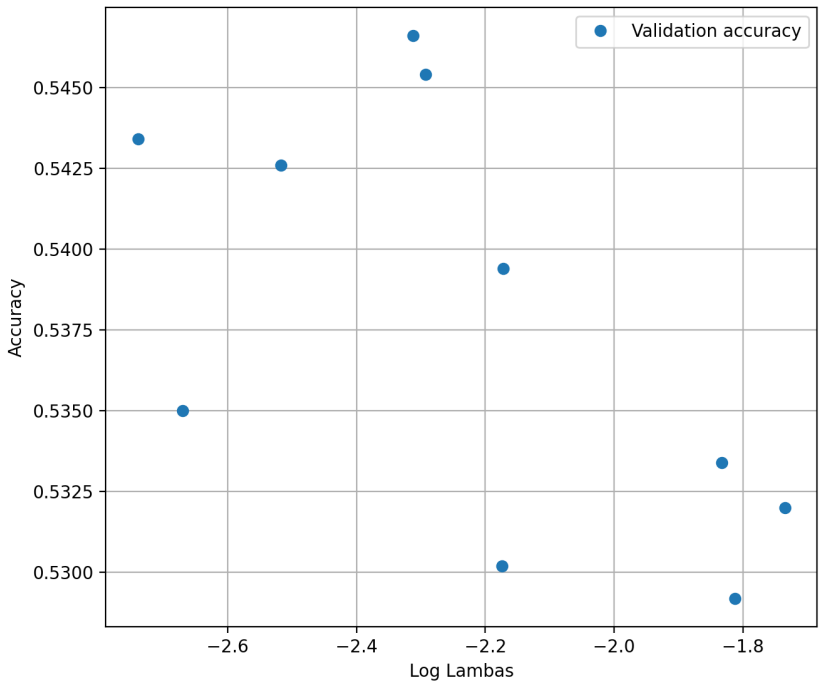


Figure 7: Validtion Accuracy over different values of the lambda regularization in the second fine search with a narrower search range (10 sampled points between -1.7 and -2.75). Network was trained on all 5 datasets, with 5000 images as validation dataset.

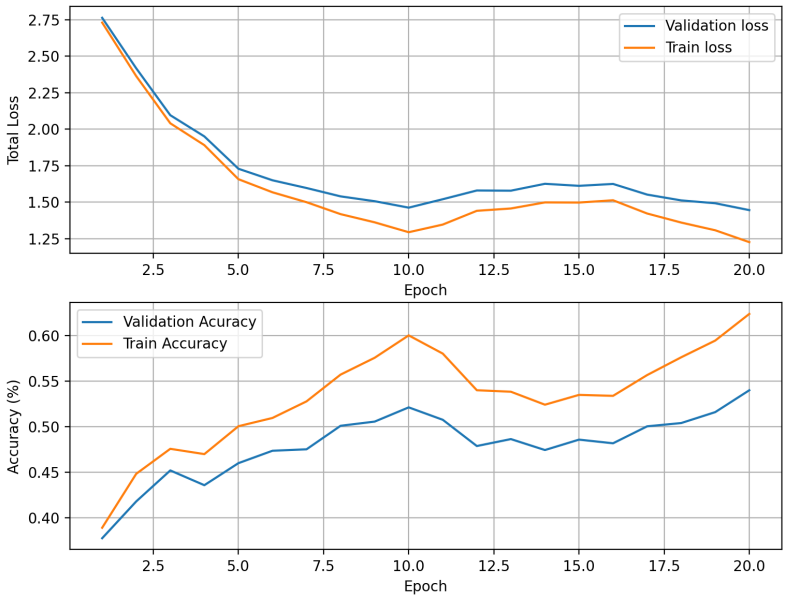


Figure 8: Final performace for the best $\lambda = 4.8739 * 10^{-3}$ with cyclical learning rates. Here, $n_{epochs} = 20$, $n_{batch} = 100$, $\eta_{min} = 1 * 10^{-5}$, $\eta_{max} = 0.1$, $n_s = 2250$. Test average accuracy = 53.98% and train average accuracy =62.36%.

Table 1: 3 best lambdas for the course search.

$\lambda(log)$	λ	accuracy (%)
-2.3985	$3.9948*10^{-3}$	54.66
-2.9442	$1.1370*10^{-3}$	53.78
-4.1068	$7.8192*\ 10^{-5}$	53.70

Table 2: 3 best lambdas for the fine search.

$\lambda(log)$	λ	accuracy (%)
-2.0824	$8.2713*10^{-3}$	54.18
-2.5717	$2.6810*10^{-3}$	54.02
-2.3164	$4.8258*\ 10^{-3}$	53.98

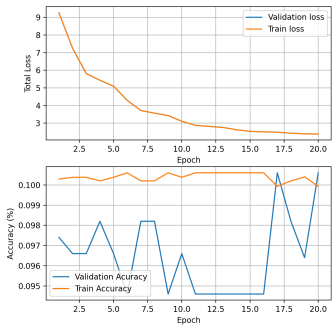
Table 3: 3 best lambdas for the second fine search (log lambda between -1.7 and -2.75).

$\lambda(log)$	λ	accuracy (%)
-2.3121	$4.8739*10^{-3}$	54.68
-2.2922	$5.1026*10^{-3}$	54.54
-2.7391	$1.8234*\ 10^{-3}$	54.34

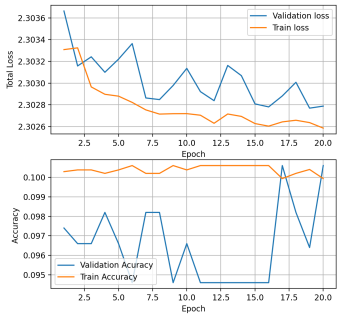
5 Weight sensitivity

Instead of using the He weight initialization, we initialize the 9-layer neural network from a normal distribution with different variances $\sigma = [10^{-1}, 10^{-3}, 10^{-4}]$. The performace of the 9-layer neural network was checked and compared to when using batch normalization versus not using it. See figure 9 and 10. The two different figures shows how important batch normalization is and that it can give good performace even with "bad" initialization of the networks weights. For speeding up the simulations, I set $n_s = 2 * \frac{4500}{n_{batch}} = 2 * \frac{45000}{100} = 900$ which resulted in less iterations but still served as a proof of concept for the batch normalization.

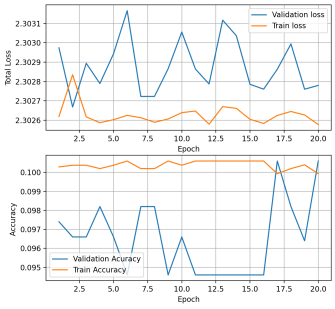
In conclusion, we can say that Neural Networks are less sensetive to weight initialization when using batch normalization. The plots in Figure 9 showed a much lower accuracy and a much flatter loss function, compared to when using the batch normalization in 10.



a) $\sigma = 10^{-1}$. Train accuracy 10.00%, test accuracy 10.06%

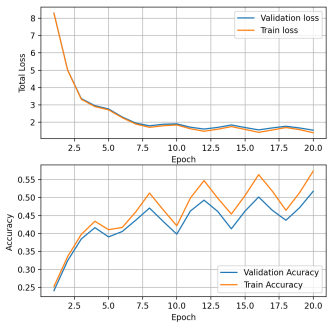


b) $\sigma = 10^{-3}$. Train accuracy 10.00%, test accuracy 10.06%

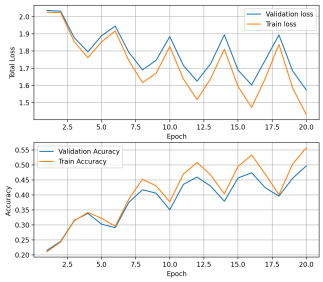


c) $\sigma = 10^{-4}$. Train accuracy 10.00%, test accuracy 10.06%

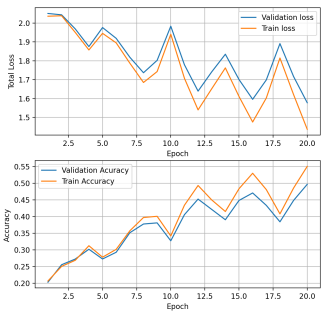
Figure 9: Different sigma initializations without batch normalization.



a) $\sigma = 10^{-1}$. Train accuracy 57.40%, test accuracy 51.78%



b) $\sigma = 10^{-3}$. Train accuracy 55.71%, test accuracy 49.66%



c) $\sigma = 10^{-4}$. Train accuracy 55.05%, test accuracy 49.66%

Figure 10: Different sigma initializations using batch normalization.