1. Introduction

In this assignment, the task was to create a k-layered neural network with and without batch normalization. To compute the gradients, the approach of mini batching was selected. The idea of mini batching is, for each epoch, to iteratively select random subsets (batches) of the training data, make predictions on the subsets, and ultimately update the model's parameters before the next epoch. Additionally, the approach of cyclical learning rates was used in order to enhance the network's accuracy. This means that the learning rate was increased and decreased periodically during training. Furthermore, a random and fine grid search for the best lambda value was done in order to regularize the network optimally.

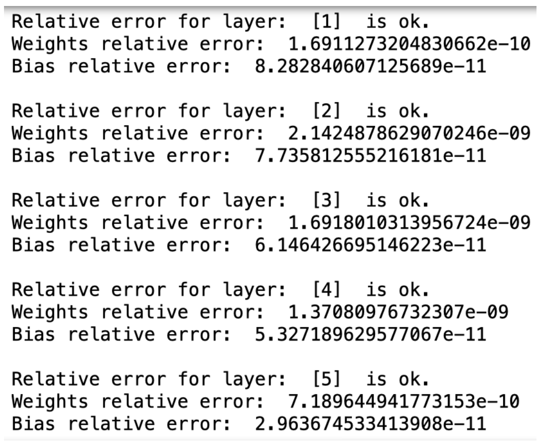
For forward pass, the softmax function was used to determine the probabilities of an image belonging to each class. For the back propagation in this model, cross-entropy was used as a loss function. By combining the cross-entropy loss and a regularization term, the full cost function was defined.

Batch normalization is the process of re-centering and re-scaling the output of a layer n, which will serve as the input for the next layer n+1. This occurs for each batch during training. Doing so, the neural network's performance and stability increases, especially for deeper networks. When applying batch norm to each layer, we introduce gamma (an estimated mean for the unnormalized scores for each layer) and beta (estimated variance for each dimension of those scores).

2. Analytic gradient and numerical gradient tests

The model used an analytical approach in order to compute the gradients of each mini batch. In order to make sure that the analytical gradient was successfully computed, it was compared to a function that rather calculated the numerical gradient. The tests were done using various (non-random) dimensions of the first training data instance. To pass the test, the absolute relative error for the model's weights (W) and bias (b) should not exceed 1e-5. Below, the test results (in figure 1) are depicted. The implementation of testing k-layers without batch norm differs from testing k-layers with batch norm. Testing with batch norm includes checking the relative error of beta and gamma values as well.

Conclusively, the testing of a small training batch example for k-layers with and without batch norm passed. The functions below were called with a threshold of 1e-5 for 2, 3 and 4 hidden layers respectively, and a reduced dimensionality of the input vectors equivalent to the assignment instructions. For batch norm, gamma and beta were also compared.



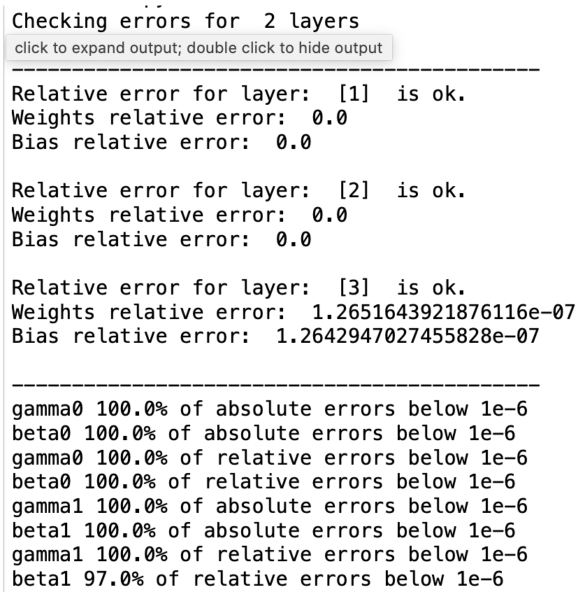
```
Relative error for layer:  [1]  is ok.
Weights relative error:  1.6911273204830662e-10
Bias relative error:  8.282840607125689e-11

Relative error for layer:  [2]  is ok.
Weights relative error:  2.1424878629070246e-09
Bias relative error:  7.735812555216181e-11

Relative error for layer:  [3]  is ok.
Weights relative error:  1.6918010313956724e-09
Bias relative error:  6.146426695146223e-11

Relative error for layer:  [4]  is ok.
Weights relative error:  1.37080976732307e-09
Bias relative error:  5.327189629577067e-11

Relative error for layer:  [5]  is ok.
Weights relative error:  7.189644941773153e-10
Bias relative error:  2.963674533413908e-11
```



```
Checking errors for  2 layers
--------------------------------------------
Relative error for layer:  [1]  is ok.
Weights relative error:  0.0
Bias relative error:  0.0

Relative error for layer:  [2]  is ok.
Weights relative error:  0.0
Bias relative error:  0.0

Relative error for layer:  [3]  is ok.
Weights relative error:  1.2651643921876116e-07
Bias relative error:  1.2642947027455828e-07

--------------------------------------------
gamma0 100.0% of absolute errors below 1e-6
beta0 100.0% of absolute errors below 1e-6
gamma0 100.0% of relative errors below 1e-6
beta0 100.0% of relative errors below 1e-6
gamma1 100.0% of absolute errors below 1e-6
beta1 100.0% of absolute errors below 1e-6
gamma1 100.0% of relative errors below 1e-6
beta1 97.0% of relative errors below 1e-6
```

Figure : Test results of the analytic gradient testing. To the left: without batch norm, 4 hidden layers,
To the right: with batch norm, 2 hidden layers.

3. Results

3.1 Default settings: loss plots with vs without batch norm, 3 layers

The plots show the loss plots of the model using batch norm (to the right) vs. no batch norm (to the left). This was done using the default setting given by the instructions. It is clear that the convergence curve is much smoother when using batch normalization.

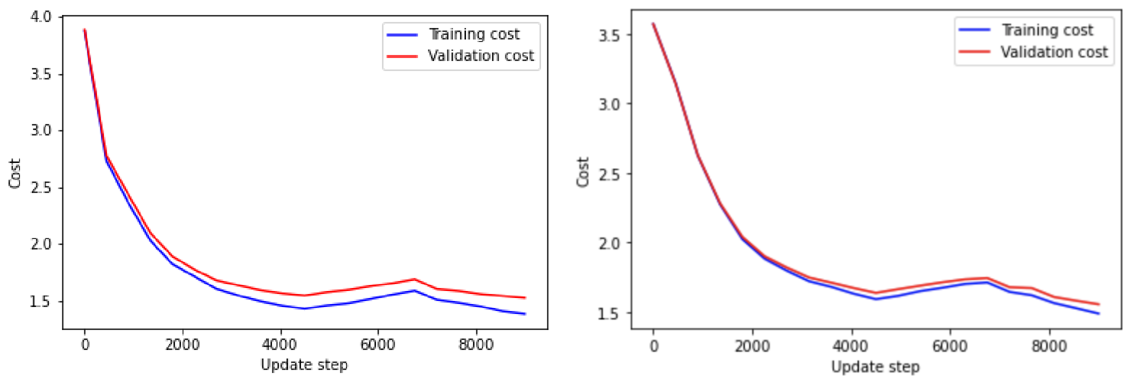


Figure 3: First test, comparing losses of 3-layered nerual networks with default parameter setting. Lambda = 0.005, eta_min = 1e-5, eta_max=1e-1, step size = 5 * 45000 / n_batch, n_batch = 100, cycles = 2.

3.2 Default settings: loss plots with vs without batch norm, 9 layers

The plots show the loss plots of the model using batch norm (to the right) vs. no batch norm (to the left). This was done using the default setting given by the instructions. The accuracy was increased with ~6% usin batch norm, but the convergence did not differ as much as in the previous experiment.

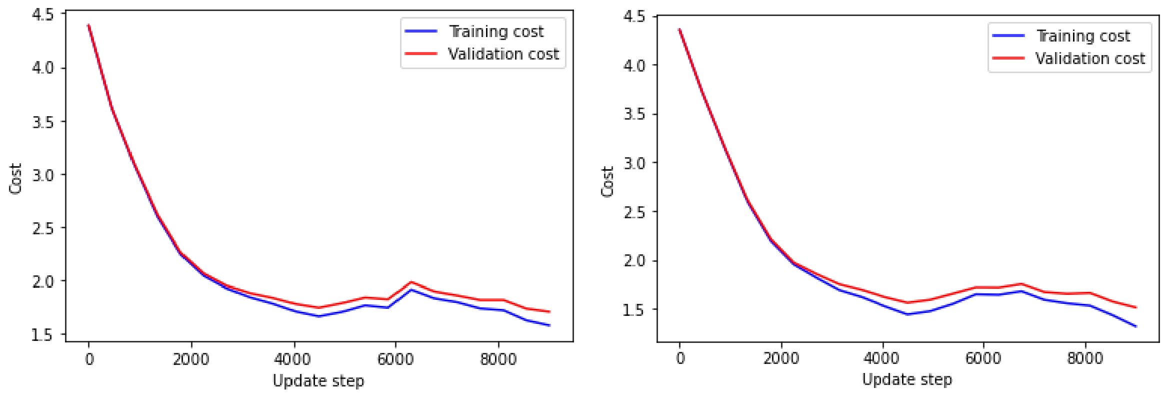


Figure 4: First test, comparing losses of 9-layered nerual networks with default parameter settings. Lambda = 0.005, eta_min = 1e-5, eta_max=1e-1, step size = 5 * 45000 / n_batch, n_batch = 100, cycles = 2.

3.3 Lambda coarse search

Below are the randomly generated lambda values used for the coarse search. The lambdas were generated on a log scale.

```
[0.0004950159553733192, 0.0005627932047415164, 0.0015119336467640998, 0.0015676677195506057,
0.002576638574613588, 0.0038333321561566606, 0.007257005721594274, 0.036905577292137581]
```

Figure 5:. Lambda values for the coarse search.

Depicted below is a random search for an optimal lambda value. The graphs can be summarized as the higher the lambda value and thus regularization term, the higher loss and lower accuracy. Given the performance graphs, the lambda values of ~0.0005 and ~0.0015 are good indicators on what values to select for the fine search.

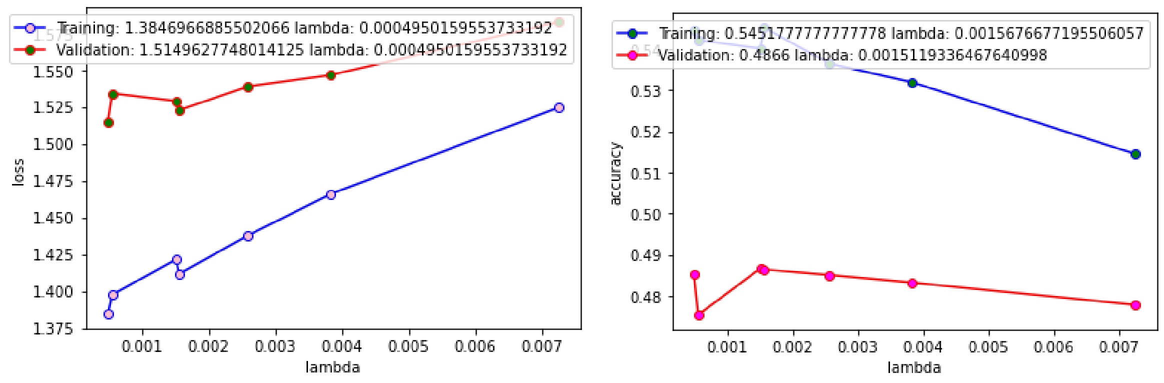


Figure 6: Results of a random search of an optimal lambda value.
eta_min = 1e-5, eta_max=1e-1, step size = 5 * 45000 / n_batch, n_batch = 100, cycles = 2.

3.4 Lambda fine search

Below are the lambda values used for the fine search. The lambdas were manually added to a list and inputted in the grid search function, having a range with respect to the results from the coarse search.

```
[0.00025,0.0005,0.001,0.0015,0.002,0.0025,0.003,0.005]
```

Figure 7:Lambda values for the fine search.

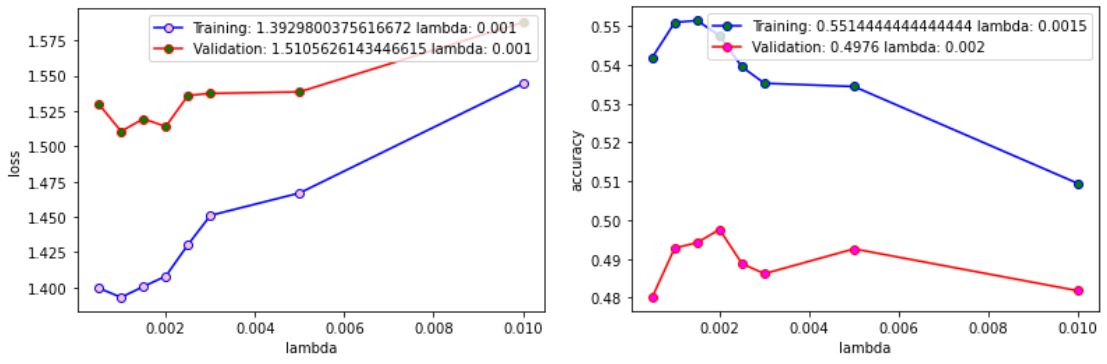
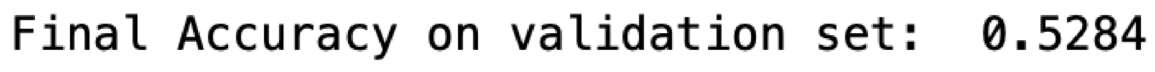


Figure 8: Results of a fine search of an optimal lambda value.
eta_min = 1e-5, eta_max=1e-1, step size = 5 * 45000 / n_batch, n_batch = 100, cycles = 2.

Results indicate that the lower the lambda, the lower the cost. But given the highest accuracy on the validation set when lambda = 0.002, this is the final lambda value that was used for the test set.

3.4 Testing the 3-layered batch norm network

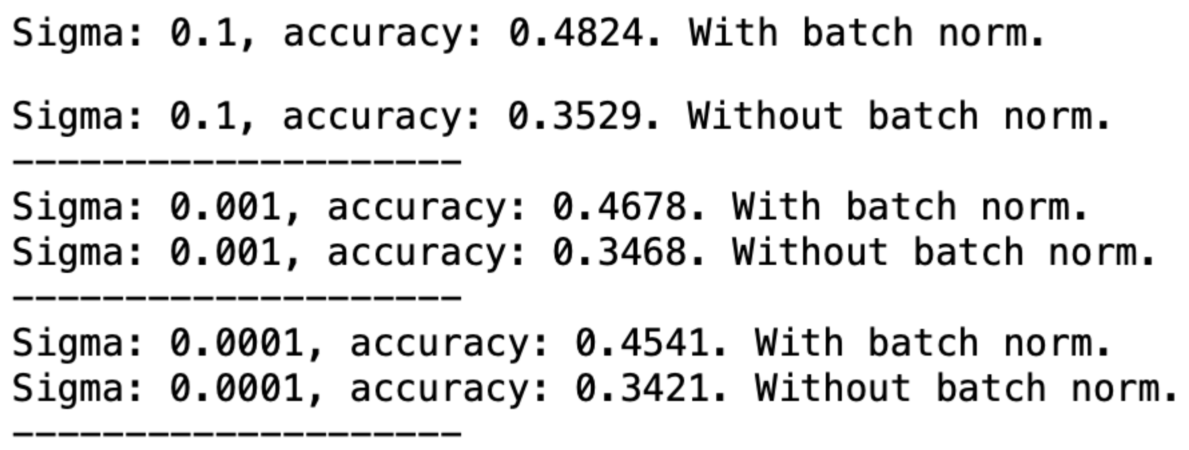
Final tests with 1000 validation data instances are depicted below. The accuracy for the test set achieved a score of approximately 53 % which is around 2 % higher from the previous assignment.



```
Final Accuracy on validation set:  0.5284
```

Figure 9: Final test. Results of the network with an optimal lambda value (=0.002).
Hyper-params: eta_min = 1e-5, eta_max = 1e-1, step size = 5 * 45000 / n_batch, n_batch = 100, cycles = 3.

3.4 Testing the network on the test set with different standard deviations



```
Sigma: 0.1, accuracy: 0.4824. With batch norm.

Sigma: 0.1, accuracy: 0.3529. Without batch norm.
--------------------
Sigma: 0.001, accuracy: 0.4678. With batch norm.
Sigma: 0.001, accuracy: 0.3468. Without batch norm.
--------------------
Sigma: 0.0001, accuracy: 0.4541. With batch norm.
Sigma: 0.0001, accuracy: 0.3421. Without batch norm.
--------------------
```

Figure 10: Testing the network on the test set with different standard deviations.
Lambda = 0.002, eta_min = 1e-5, eta_max = 1e-1, step size = 5 * 45000 / n_batch, n_batch = 100, cycles = 2.