Assignment 3

DD2424 Deep Learning in Data Science

Márk Antal Csizmadia
macs@kth.se

April 2021

1 Introduction

This short report describes my solution to Assignment 3, the implementation of which may be found in the script uploaded alongside with the report. I chose Option 1 (of the 2 available options) that is an assignment based on implementing batch normalisation to fully connected networks.

2 Implementation

In this section, a brief description of my implementation is followed by the explanation of the numerical gradient check.

2.1 Neural network blocks

In my implementation, I built on top of my code from Assignment 2. At the core of my implementation is the `Dense` layer (fully-connected layer) that can have activations such as `SoftmaxActivation`, and `ReLUActivation`. The learnable (or trainable) parameters of the `Dense` layer can be initialized with the `XavierInitializer` and the `NormalInitializer`, and can be regularized with the `L2Regularizer`. Further to the `Dense` layer, batch normalization was implemented in the `BatchNormalization` layer. A `Model` comprises any number of layers, its loss function is the `CategoricalCrossEntropyLoss`. The trainable parameters are optimized with the mini-batch gradient descent algorithm via the `SGDOptimizer` and the learning rate schedule of the optimizer is set to be `LRCyclingSchedule` [1]. The `Model` is first compiled with the loss function, some metrics such as the `AccuracyMetrics`, and the optimizer, and then it is fit to the data with the `Model.fit` method.

My code can be used to build a `Model` with k-layers. In this assignment, I implemented a three-layer and a nine-layer neural network for multi-class classification such that all of the `Dense` layers' trainable parameters are initialized with the `XavierInitializer` (or the `NormalInitializer` when specified) and regularized with the `L2Regularizer`. The variants of the aforementioned networks also contain a `BatchNormalization` layer after each hidden layer. Note that there is no batch normalization after the last layer which outputs the scores, or predictions of the network. In the three-layer network, there are 50 and 50 nodes in the first and second hidden layers respectively. In the nine-layer network, there are [50, 30, 20, 20, 10, 10, 10, 10] nodes in the hidden layers. The hidden `Dense` layers have `ReLUActivation` activations, and the last `Dense` layer (classifier) has `SoftmaxActivation` activation. The `Model` is then compiled with the `CategoricalCrossEntropyLoss` loss function, the `AccuracyMetrics` performance metric, and the `SGDOptimizer` optimizer that uses the `LRCyclingSchedule` learning rate schedule.

The coarse-to-fine hyperparameter search, which sets the L2 regularization rate, is implemented with the `Tuner` class.

2.2 Numerical gradient check

The analytic gradient computations for the for k-layer network with batch normalization is implemented with the `numerical_gradient_check_model` method. At the core of the method lies the `eval_numerical_gradient` method that computes the numerical gradient of a function with respect to some input. The `test_model` method compares the analytical to the numerical gradients of the layers with learnable parameters in the k-layer network, such as the `Dense` and the `BatchNormalization` layers. In the case of the three-layer network without batch normalization, the three-layer network with batch normalization, the nine-layer network without batch normalization, and the nine-layer network with batch normalization, generally, the largest relative error between any entry in the analytical and numerical gradient arrays is in the range of 10e-7 - 10e-4.

As outlined in [2], the aforementioned discrepancies are satisfactory to assert that the analytical gradients of the model are correct. To avoid kinks in the objective function, only a few, namely 2, data points, and the first 10 dimensions were used in the gradient check. The step size for computing the numerical gradient was set to 1e-05. In computing the relative error, the formula provided in the Assignment 1 document was used with a slight modification based on [2]. Furthermore, the `numpy.testing.assert_array_almost_equal` function was also used to make the comparison.

3 Results

In generating the results discussed below, the whole CIFAR-10 data set was used. For training the networks, the training and validation data sets included `data_batch_1`, `data_batch_2`, `data_batch_3`, `data_batch_4`, and `data_batch_5`. Where not specified otherwise, the validation set was 5000 randomly sampled image and the training data set was the 45000 remaining image. At test-time, the `test_batch` set of 10000 images was used. Please note that the accuracy measures are given as real numbers between 0.0 and 1.0, where 0.0 mean 0.0% and 1.0 means 100.0%.

The *default parameter settings* are as follows:

- The k-layer network has k `Dense` layers. When specified, a `BatchNormalization` layer is attached after each hidden layer.
- The learnabele parameters of each `Dense` layer are initialized with the the `XavierInitializer` with a mean of 0 and a standard deviation of $1/\sqrt{in\_dim}$ where `in_dim` is the number of hidden nodes in the previous layer. When specified, the `NormalInitializer` with a mean of 0 and a custom standard deviation is used (see in the sensitivity to initialization section).
- The number of hidden nodes in the case of the 3-layer and the 9-layer networks are as specified before.
- The base and the maximum learning rate (`eta_min` and `eta_max`) of the `LRCyclingSchedule` are 1e-5 and 1e-1, respectively.

- The step-size (half-cycle) of the `LRCyclingSchedule` is $n\_s = 5 * 45,000/n\_batch$. Since $n\_batch = 100$ throughout the assignment, $n\_s = 2250$ update steps. Equivalently, one cycle is two step-size, that is 4500 update steps, which equals to 10 epochs. The networks are generally trained for 2 cycles, that is, for 20 epochs. The network trained with the best `lambda` from the coarse-to-fine search is trained for 3 cycles, that is, for 30 epochs.

- The L2 regularization rate (`lambda`) is 0.005. In the coarse-to-fine search, it varies as described later.

- The `gamma` and `beta` learnable parameter matrices of the `BatchNormalization` layer are initialized to ones and zeros, respectively. The `momentum` hyperparameter and `epsilon` of each batch normalization layer are set to 0.9 and 1e-5, respectively.

3.1 Three-Layer Network with and without Batch Normalization

In this section, the results of training and testing the aforementioned three-layer network and its variant with batch normalization is discussed. The parameter setting of the training is the *default parameter* setting.

Figure 1 and 2 show the results of training the three-layer network without and with batch normalization, respectively.

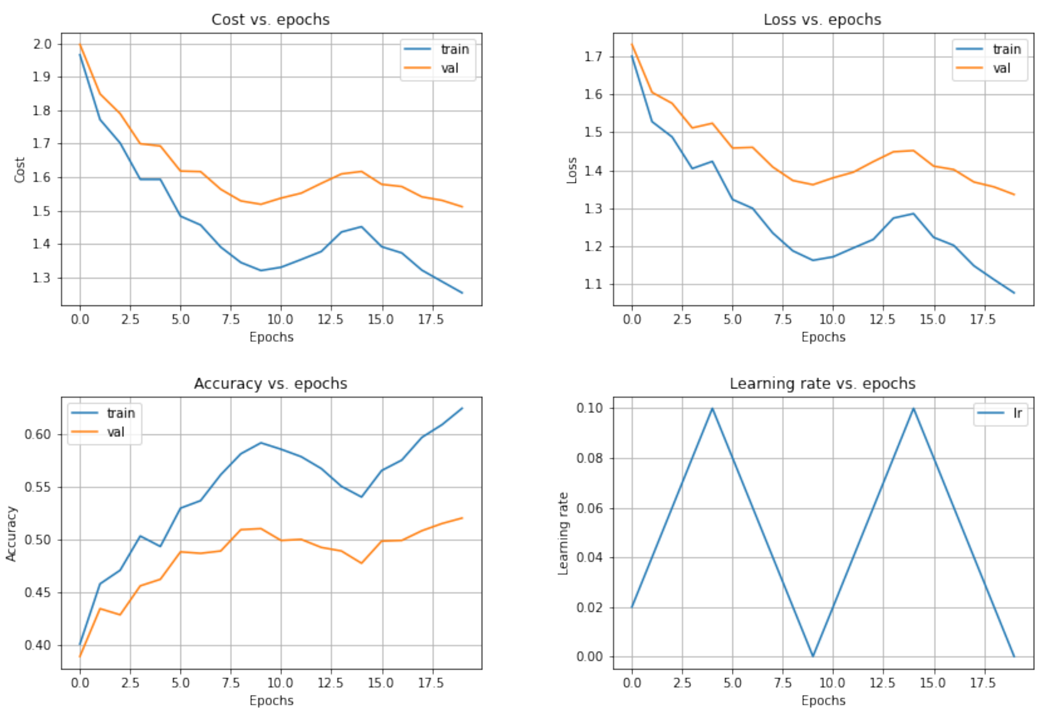

Figure 1: The cost, loss, accuracy, and learning rate curves of training the three-layer network without batch normalization for 2 cycles (i.e.: 20 epochs). Note that accuracy measures are given as real numbers in the range 0-1.

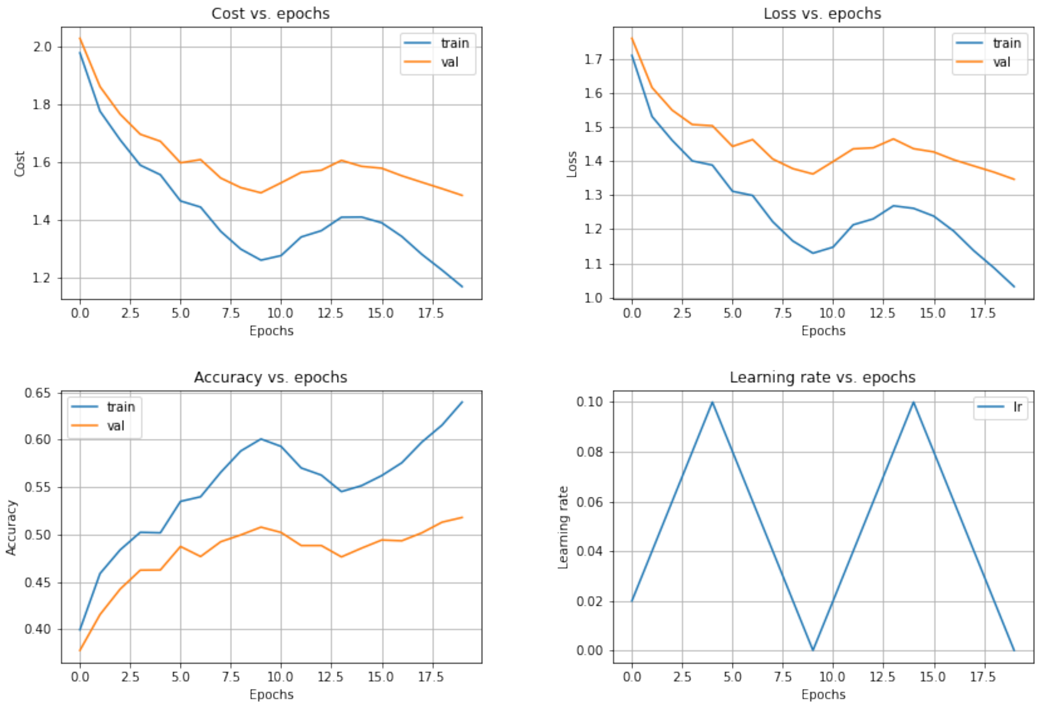

Figure 2: The cost, loss, accuracy, and learning rate curves of training the three-layer network with batch normalization for 2 cycles (i.e.: 20 epochs). Note that accuracy measures are given as real numbers in the range 0-1.

Note that the reason for the learning rate curve being shifted is that the learning rate is always recorded at the end of an epoch (i.e.: update steps have already taken place).

The test accuracy of the trained model when run on the test data set in the case of the network without and with batch normalization was 0.5360 and 0.5323, respectively. Note that accuracy measures are given as real numbers in the range 0-1. As shown in Figure 1 and 2, and comparing the test accuracy metrics, it may be concluded that adding a batch normalization layer after each hidden layer does not significantly improve the model performance. It may be attributed to the fact that the three-layer network is not deep enough to cause activation saturation, and hence, vanishing gradients. Thus, the gradient descent-based parameter updates are stable and the model is efficiently learning in both cases.

3.2 Nine-Layer Network with and without Batch Normalization

In this section, the results of training and testing the aforementioned nine-layer network and its variant with batch normalization is discussed. The parameter setting of the training is the *default parameter* setting.

Figure 3 and 4 show the results of training the three-layer network without and with batch normalization, respectively.

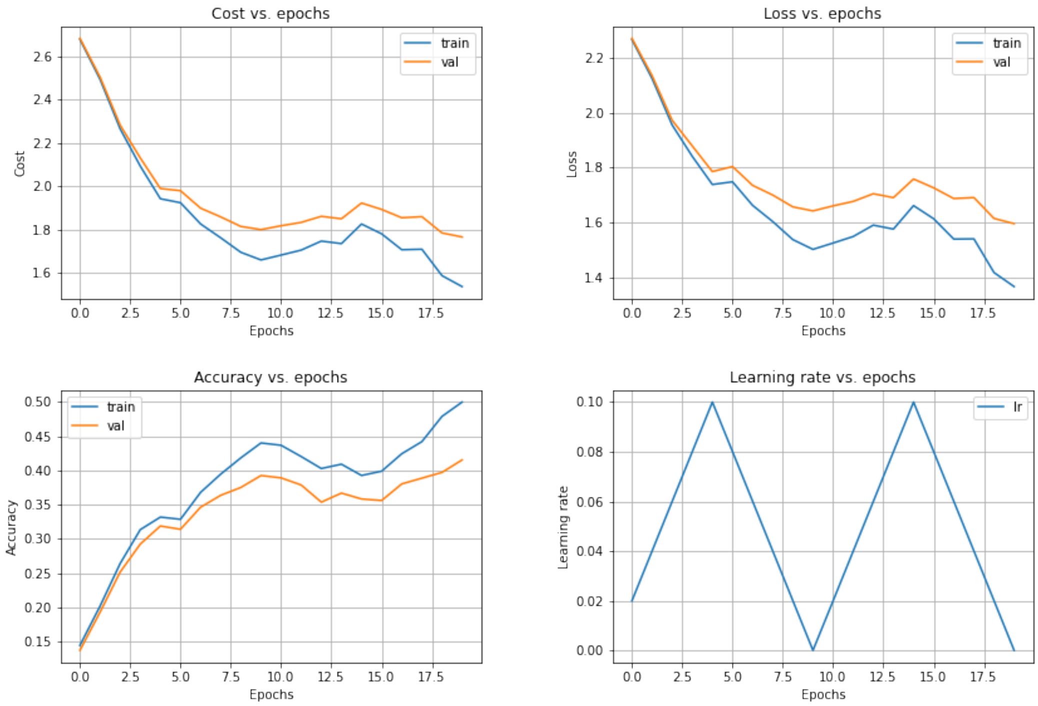

Figure 3: The cost, loss, accuracy, and learning rate curves of training the nine-layer network without batch normalization for 2 cycles (i.e.: 20 epochs).

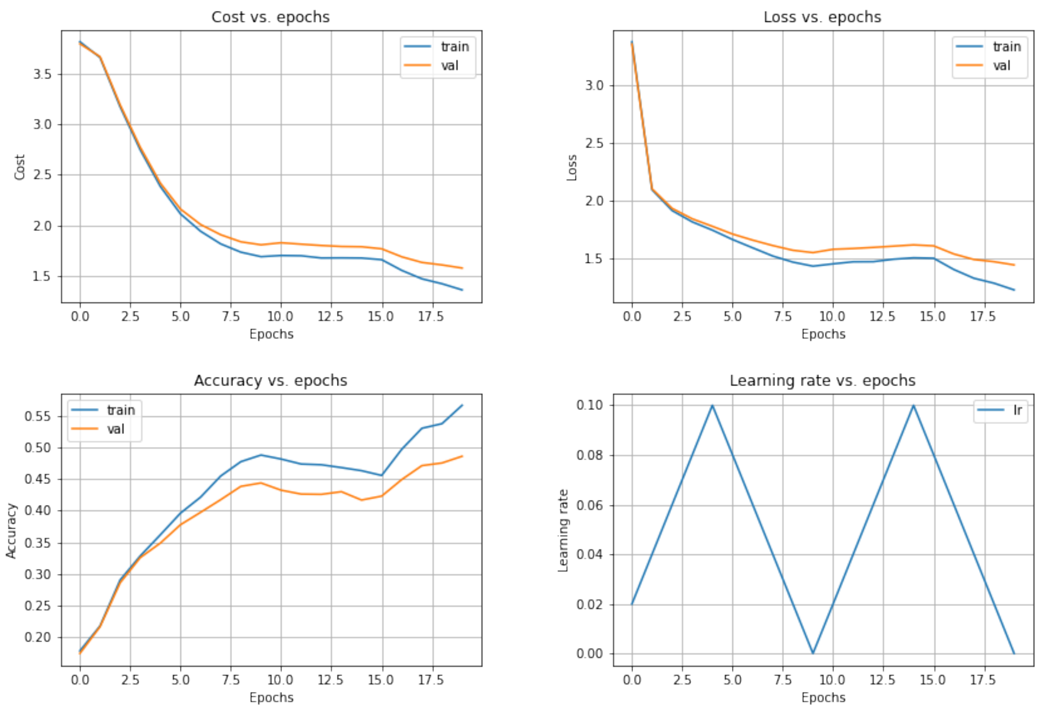

Figure 4: The cost, loss, accuracy, and learning rate curves of training the nine-layer network with batch normalization for 2 cycles (i.e.: 20 epochs).

The test accuracy of the trained model when run on the test data set in the case of the network without and with batch normalization was 0.4411 and 0.5019, respectively. Note that accuracy measures are given as real numbers in the range 0-1. As shown in Figure 3 and 4, and comparing the test accuracy metrics, it is evident that adding a batch normalization layer after each hidden layer improves the model performance. This may be attributed to the fact that nine layers are deep enough to cause activation saturation via increase internal covariate shift across layers. In turn, the training slows down due to vanishing gradients. Batch normalization helps remedy this by reducing the internal covariate shift across layers, thus eliminating vanishing gradients, and speeding up learning.

3.3 Coarse-to-Fine Hyperparameter Search for the L2 Regularization Strength

A coarse-to-fine hperparameter search for the best value of the L2 regularization strength was conducted with the three-layer network with batch normalization. In each run of the search, the model was trained for 2 full `LRCyclingSchedule` cycles, or equivalently, for 20 epochs.

In each run of the coarse search, the L2 regularization rate `lambda` was sampled as

$$lambda = 10^{l_{min}+(l_{max}-l_{min})*U(0,1)} \tag{3.1}$$

where $U(0,1)$ is a random number sampled from the uniform distribution between 0 and 1. The formula is taken from Assignment 2. In the coarse search 10 different, randomly sampled values were explored. The `lambda` of the model with the best performance on the validation set, `lambda_current_best`, was then used in the subsequent fine search. In the fine search, the L2 regularization rate `lambda` was sampled as

$$lambda = U(0.8 \times lambda\_current\_best, 1.2 \times lambda\_current\_best) \tag{3.2}$$

that is, a narrow interval around the `lambda_current_best` was further explored. In the fine search, 5 different, randomly sampled values were explored. In overall, 2 runs of fine-search were conducted. That is, first, the coarse search explored 10 randomly sampled values, then one run of fine search explored 5 new values around the best coarse value, and then another fine search further explored 5 values in the region around the best value from the previous fine search step. Therefore, in overall, the model was trained 20 times with different `lambda` values for 20 epochs.

The results of the aforementioned steps of the coarse-to-fine search are shown in Table 1, 2, and 3. Table 1 shows the results of the coarse search, Table 2 shows the results of the first subsequent fine search, and Table 3 shows the results of the second subsequent and final fine search. Note that accuracy measures are given as real numbers in the range 0-1.

lambda	val_accuracy
$4.14{\cdot}10^{-5}$	0.5106
$9.12{\cdot}10^{-3}$	0.528
$1.13{\cdot}10^{-4}$	0.51
$1.36{\cdot}10^{-3}$	0.5132
$1.14{\cdot}10^{-5}$	0.5146
$4.73{\cdot}10^{-2}$	0.5114
$4.01{\cdot}10^{-2}$	0.5128
$1.36{\cdot}10^{-5}$	0.5042
$6.73{\cdot}10^{-2}$	0.5098
$3.54{\cdot}10^{-5}$	0.5004

Table 1: Coarse search

lambda	val_accuracy
$7.86{\cdot}10^{-3}$	0.534
$1{\cdot}10^{-2}$	0.5262
$8.26{\cdot}10^{-3}$	0.5338
$9.25{\cdot}10^{-3}$	0.5216
$7.35{\cdot}10^{-3}$	0.5264

Table 2: Fine search 1

lambda	val_accuracy
$6.77{\cdot}10^{-3}$	0.5282
$8.62{\cdot}10^{-3}$	0.5268
$7.12{\cdot}10^{-3}$	0.5248
$7.97{\cdot}10^{-3}$	0.5198
$6.34{\cdot}10^{-3}$	0.5258

Table 3: Fine search 2

As shown in Table 1, the best `lambda` of the coarse search was 9.12×10^{-3} which yielded a validation accuracy of 0.528. Searching around this value as shown in Table 2, the best `lambda` of the first fine search step turned out to be 7.86×10^{-3} which yielded a validation accuracy of 0.534. Searching around the value from the first fine search step, as shown in Table 3, did not result in a better validation accuracy as in the previous step. Therefore, the best value for the L2 regularization strength `lambda` was found to be 7.86×10^{-3} via the coarse-to-fine hyperparameter search. Generally, the search explored worthwhile regions as demonstrated by the generally increasing validation accuracy metrics as the search went from coarse to finer.

The network with the best `lambda` from the coarse-to-fine search was trained for 3 cycles, that is, for 30 epochs. The results of the training are shown in Figure 5.

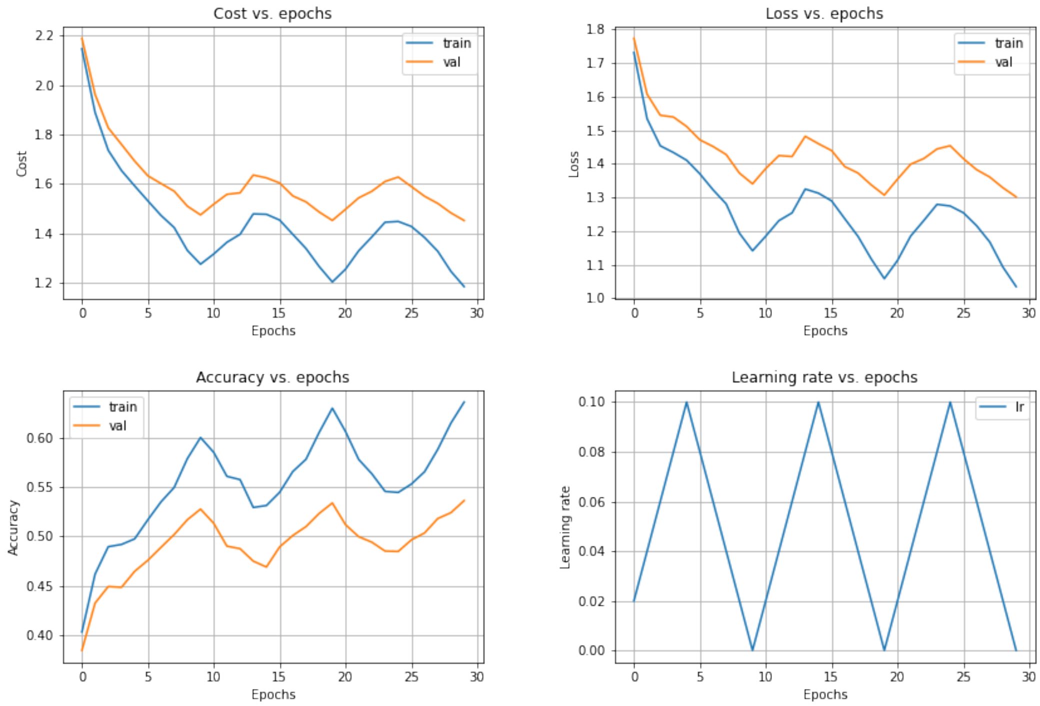

Figure 5: The cost, loss, accuracy, and learning rate curves of training the three-layer network with batch normalization for 3 cycles (i.e.: 30 epochs) with the best L2 regularization strength rate `lambda` found via the coarse-to-fine random search.

The network's test accuracy after training for 30 epochs with lambda $= 7.86 \times 10^{-3}$ was 0.5392. The coarse-to-fine search efficiently explored worthwhile regions of `lambda`. The limitation of the search was the lack of adequate computational resources and the lack of probabilistic strategy such as that of in Bayesian hyperparameter optimization.

3.4 Sensitivity to Initialization

In this section, it is explored how the three and nine-layer network's sensitivity to the learnable parameter initialization varies when batch normalization is not used and when it is used. In the experiments, the learnable parameters are initialized such that they are drawn from a normal distribution with mean 0 and standard deviation `sigma` (instead of the Xavier initialization). As before, each network is trained for 2 fully cycles of `LRCyclingSchedule`, that is, for 20 epochs. The test accuracy metrics after training the aforementioned networks are depicted in Table 4 where `n_layers` stands for the number of hidden layers in the network, `if_bn` stands for whether batch normalization was used, `sigma` is the standard deviation of the normal parameter initialization, and `test_accuracy` is the test accuracy of the trained model.

n_layers	if_bn	sigma	test_accuracy
3	False	0.1	0.5346
3	False	0.001	0.5193
3	False	0.0001	0.1
3	True	0.1	0.536
3	True	0.001	0.5346
3	True	0.0001	0.535
9	False	0.1	0.1
9	False	0.001	0.1
9	False	0.0001	0.1
9	True	0.1	0.5192
9	True	0.001	0.5079
9	True	0.0001	0.1

Table 4: Results of the experiment of the sensitivity to initialization

As shown in Table 4, batch normalization helped make both the three and the nine-layer networks more robust to the learnable parameter initialization. In the case of the three-layer network without batch normalization, the test accuracy drops from around 53 % to 10 % as `sigma` goes from 0.1 to 0.0001. When batch normalization is used, the three-layer network is less susceptible to the the decreasing sigma and still achieves a test accuracy around 53.5 %. In the case of the nine-layer network, the lack of batch normalization results in a test accuracy of 10 % regardless of the value of `sigma`. When batch normalization is used, the nine-layer network is more robust to the decreasing `sigma` and it maintains a test accuracy of around 51 % until it drops back to 10 % for sigma = 0.0001.

4 Conclusions

In this assignment, a three and a nine-layer neural network were trained to classify images in the CIFAR-10 dataset. Using these network architectures, batch normalization was explored as a means of helping deep networks become less sensitive to the learnable parameter initialization schemes and the vanishing gradient problem. Lastly, a coarse-to-fine hyperparameter search was conducted to find the best L2 regularization rate for the three-layer network with batch normalization. As demonstrated by the results, batch normalization indeed helps deep networks become less sensitive and susceptible to parameter initialization, and become less prone to the vanishing gradient problem.

References

[1] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*, IEEE, 2017, pp. 464–472.

[2] CS231n Team, *Learning*. Stanford University, 2020. [Online]. Available: `https://cs231n.github.io/neural-networks-3/`.