DD2424 Deep Learning in Data Science

Assignment 3

Ali Shibli

April 2021

1 Introduction

In this part, I will report on the bonuses that I tackled from assignment 3 of the Deep Learning Course. I discuss the solution for the following 4 parts:

1. (b) Do a more thorough search to find a good network architecture. Does making the network deeper improve performance?
2. (c) It has been empirically reported in several works that you get better performance by the final network if you apply batch normalization to the scores after the non-linear activation function has been applied. You could investigate whether this is the case. You will have to update your forward and backward pass of the back-prop algorithm accordingly.
3. (d) Apply dropout to your training if you have a high number of hidden nodes and you feel you need more regularization.
4. (e) Augment your training data by applying small random geometric and photometric jitter to the original training data. You can do this on the fly by applying a random jitter to each image in the mini-batch before doing the forward and backward pass.

The network by default is the 3 layers network, with 50 units each, unless specified. In addition in the experiments, unless specified as well, I will use the best parameters gotten from the main assignment 3 for regularization as well as the same parameters for the Mini-Batch function as follows:

- lambda = lambda_best = 0.0001
- eta_min = 1e-5
- eta_max = 1e-1
- batch_size = 100

- number of cycles = 2
- number of sample per batch = 2*45000//batch_size

Finally, all the accuracy and loss plots shown have points sampled at 10 steps at a time, so that to make the plots more meaningful. If we add all 3600 points, the plots will be blurry.

2 Bigger network architecture

In this part, we investigate whether making the network bigger and changing the number of hidden nodes will affect the network. Since we tried with 3 and 9 layer networks in the original assignment, we will test with 6 layer networks now. We try with 3 different architectures of hidden nodes as following:

1. Hidden layers having [50,50,50,50,50] units each
2. Hidden layers having [100,100,100,100,100] units each
3. Hidden layers having [50,100,150,100,50] units each

The reasoning behind these tests is that we want to check first if we have the same number of units (50) in each layer, does the performance drop? And then if we increase the number of units to the double (100), will that help the performance become better? Finally, we have a varying number of nodes where it goes from 50 up to 150, then goes down to 50. This could help better encode the data and convergence during training.

The graphs and reported test accuracies are shown in the following figures:

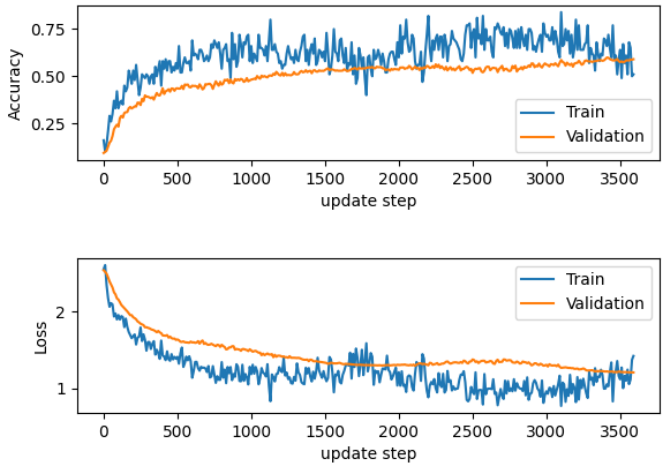

Figure 1: Loss-Accuracy for model 1. The reported test accuracy was 0.4997

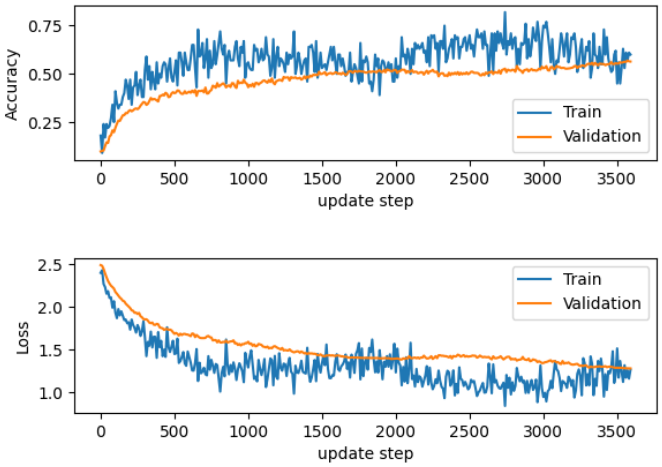

Figure 2: Loss-Accuracy for model 2. The reported test accuracy was 0.5224

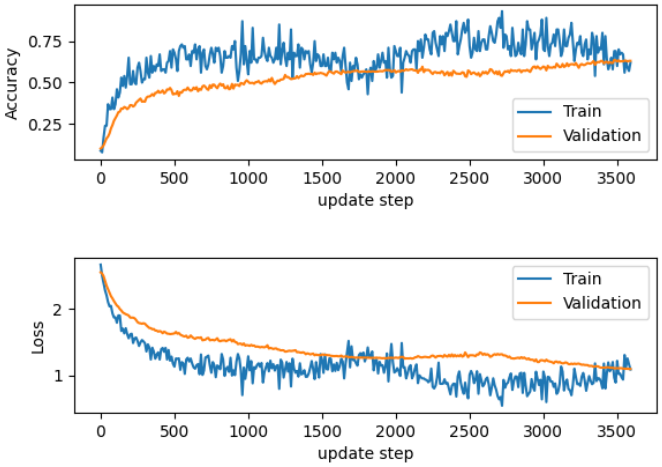

Figure 3: Loss-Accuracy for model 3. The reported test accuracy was 0.5323

We notice from the results that we can answer the questions posed at the beginning. An equal number of units per layer is number the most favourable option that we would follow. Even after increasing the number of hidden units, having the same number of these performed less than having varying number of units per layer. We can conclude that varying the number of units is important given the network structure.

3 Batch normalization: before activation vs after

In this part, we investigate whether applying batch normalization before or after the non-linear activation function ReLu makes a difference. In the following 2 plots, I show the loss and accuracy evolution, as well as the test accuracies reported once when applying batch normalization before and once after the ReLu:

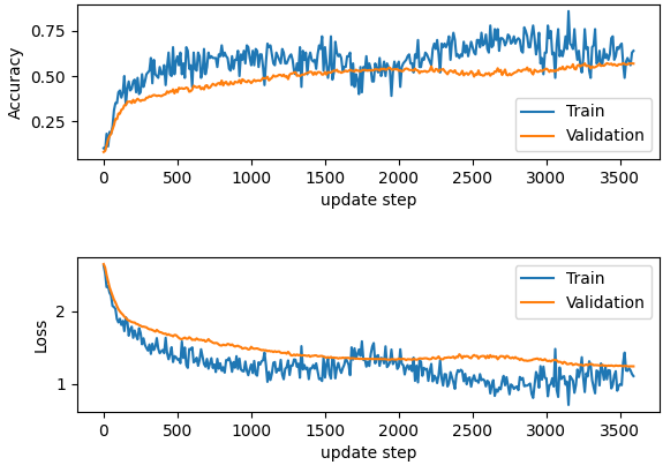

Figure 4: Loss-Accuracy when applying batch norm before ReLu. The reported test accuracy was 0.5111

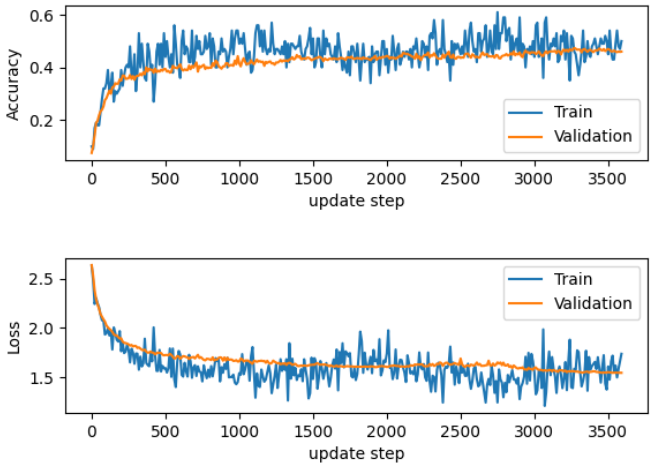

Figure 5: Loss-Accuracy when applying batch norm after ReLu. The reported test accuracy was 0.4499

We can clearly recognize that applying the normalization before the ReLu has gained much better performance on the same dataset, an increase of about 5% accuracy. This can be justified that if we applying really high variant datapoints (with high and maybe sparse values) to the activation function, this will not but ruin the performance more and add more noise and instability to the results. Thus, applying it before can give more meaningful inputs to the activation function before processing the data.

4 Adding Dropout

We try in this case to add more regularization to the network by applying dropout. We investigate with 3 different dropout rates, and report their test performances here:

Dropout Rate	Test Accuracy
0.3	0.5116
0.5	0.512
0.7	0.5089

Table 1: Dropout rate vs test accuracy results

We notice there is not much difference between the different dropout rates. As well, the results are a margin close (and less) than the best results obtained using the 3 layer network without dropout in the main assignment (0.521). For the best case with 0.5 dropout rate, we plot the loss and accuracy plots:

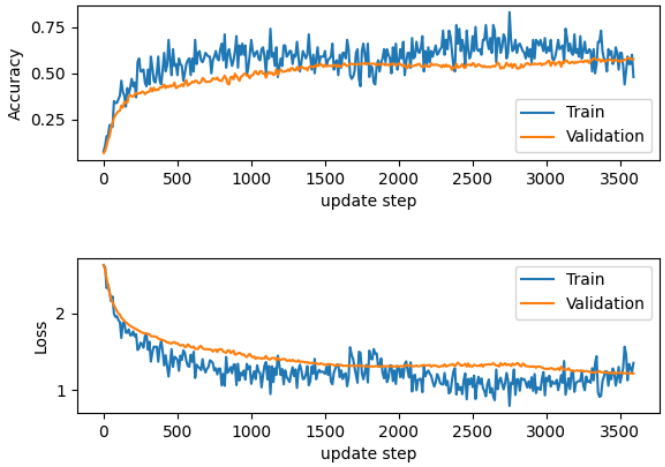

Figure 6: Loss-Accuracy when applying dropout with rate = 0.5

5 Augmenting jitter

Finally, we experiment with adding jitter noise to the model dataset after each batch. We add normal gaussian noise with 0 mean and 0.1 standard deviation. However, we notice that test accuracy also drops a bit to 0.509, a decrease of about 2%. Thus we realize that in this case adding noise wasn't very helpful. The plots for training loss and accuracy are shown as well:

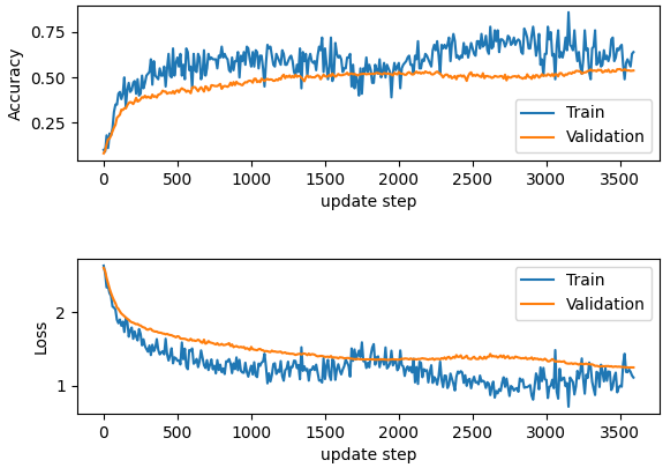

Figure 7: Loss-Accuracy curves when applying jitter noise