Lab 1 - Deep learning

Victor Stenmark (vstenm)

April 22, 2024

Contents

1 Analytic gradient computations

To check my analytic gradient computations, I compared them with numerical gradient computations for 30 iterations of downsized vectors of the input data. I also downsized W_1 to get the calculations to work but left W_2, b_1, and b_2 to their original sizes. The code for this can be seen in figure 1.

From these values, I calculated the gradients both analytically and numerically and compared the two using the function in figure 1. I then verified that the gradient differences were all below 10^{-5} for W_1, W_2, b_1, and b_2

```
for i in range(1, 30, 1):
    first_image_X = train_X[(i-1)*20:20*i, (i-1)*2:2*i]
    first_image_Y = train_Y[:, (i-1)*2:2*i]
    network.W_1 = network.W_1[:, 0:20]
```

Figure 1: Code for generating the downsized inputs and W1 for checking the gradient calculations.

```
def ComputeGradientDifference(g_a, g_n, eps = 1e-12):
    return np.sum(np.absolute(g_a - g_n)) / np.sum(np.maximum(eps,
         np.absolute(g_a) + np.absolute(g_n)))
```

Figure 2: Code for calculating gradient differences

I then trained the network on a small portion of the training data, using only 10 batches with 10 images per batch. On this data, I managed to overfit and get a whopping 94% training accuracy and 19.17% validation accuracy after 40 epochs. The results of this training can be seen in figures 3 and 4.

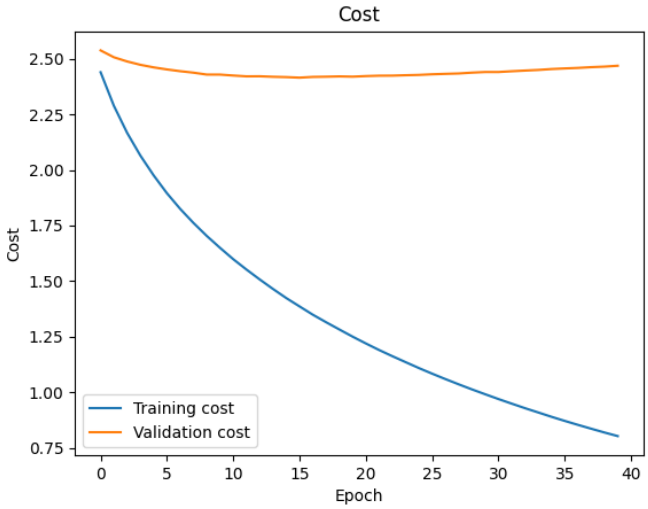

Figure 3: Graph showing the cost when training on 10 batches of 10 images per batch.

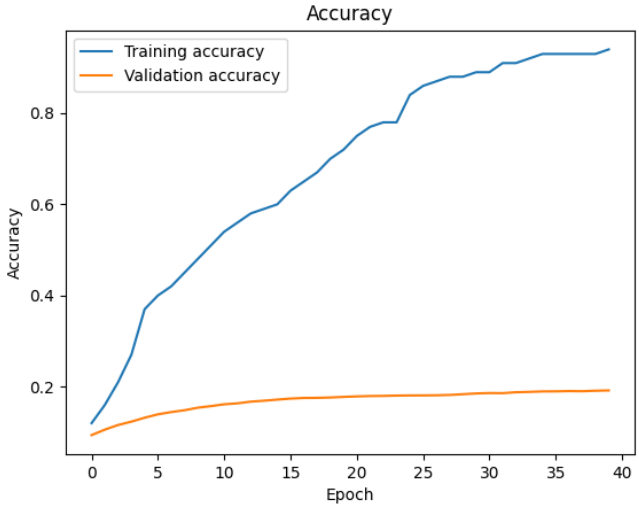

Figure 4: Graph showing the accuracy when training on 10 batches of 10 images per batch.

2 Cyclical learning rates with default values

The curves in figures 5 and 6 are roughly similar to the corresponding ones in the assignment. One interesting difference is that the curves in the assignment are much smoother. Still, my network's accuracy on the test dataset after training is 45.25% which is very close to 45.85% (the accuracy from the assignment).

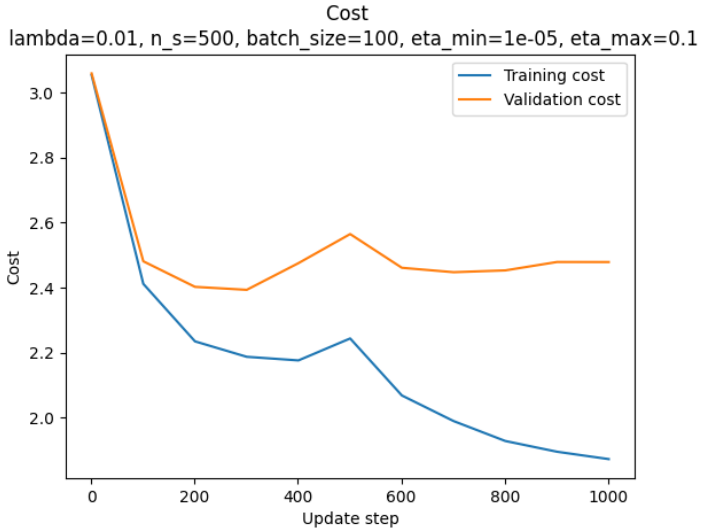

Figure 5: Graph showing the training and validation costs when training for one cycle. Measurements of cost are taken every 100 update step.

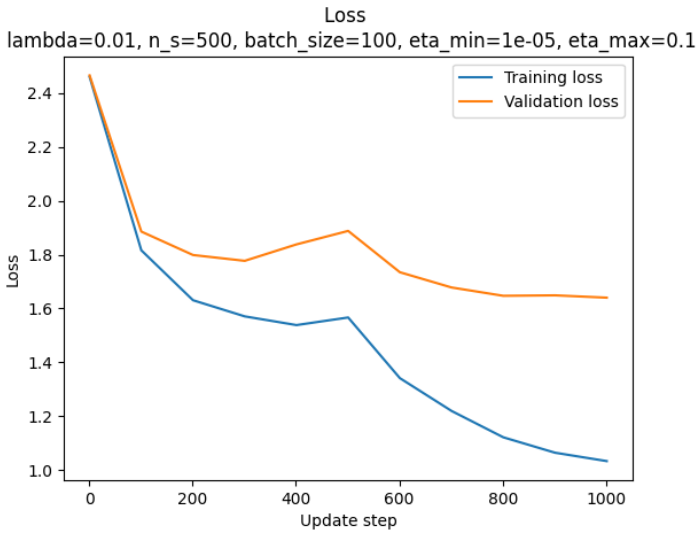

Figure 6: Graph showing the training and validation losses when training for one cycle. Measurements of loss are taken every 100 update step.

Interestingly, when I trained my network for three cycles, then the curves started to differ quite a lot from the ones in the assignment. I had problems with an increasing cost over time for the validation as well as for the training dataset. The loss on the other hand decreased on the training dataset while it remained high for the validation dataset. To ensure that I had implemented everything correctly, I asked one of the TAs to look at my graphs. She reassured me however that my implementation wasn't faulty. My guess as to why the curves look like they do is that it's because of a (sometimes) high learning rate combined with a relatively small amount of training data.

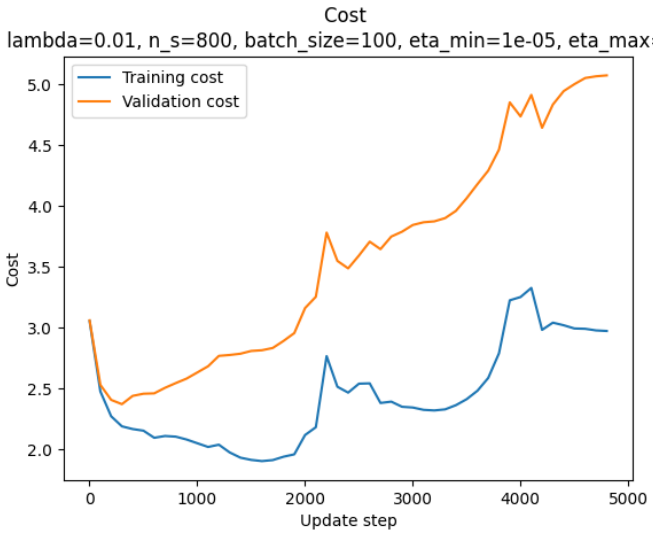

Figure 7: Graph showing training and validations costs when training for three cycles. Measurements of the costs are taken every 100 update steps.

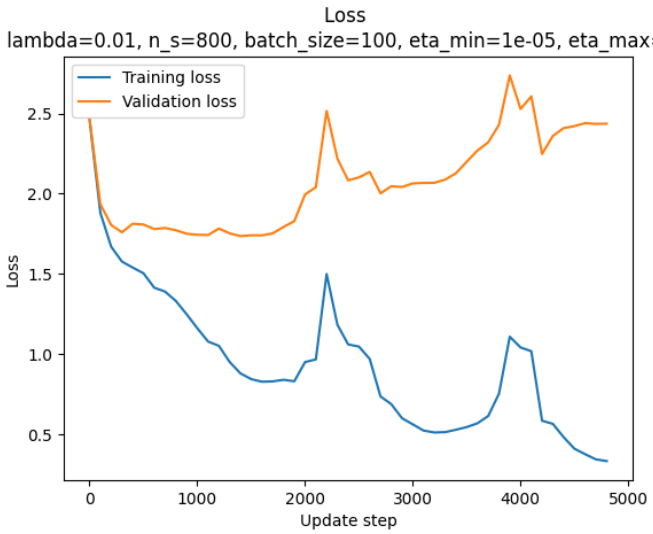

Figure 8: Graph showing training and validations losses when training for three cycles. Measurements of the losses are taken every 100 update steps.

3 Coarse search

I tested 8 values in the range of $[10^{-1}, 10^{-5}]$ that I created using np.logspace(-5, -1, 8). Training was then run for 2 cycles for every parameter setting. The four best values for lambda was, in descending order, 0.00193, 0.00052, 0.00014, and 0.0072. All the parameters tested can be seen in table 1.

λ	Best validation accuracy
1e-05	0.4804
3.727593720314938e-05	0.5092
0.00013894954943731373	0.518
0.0005179474679231213	0.518
0.0019306977288832496	0.5192
0.007196856730011514	0.517
0.026826957952797246	0.511
0.1	0.5128

Table 1: Coarse search for 8 values generated using np.logspace(-5, -1, 8). The accuracy is the best validation accuracy the network received with the corresponding lambda.

4 Fine search

Next, I ran a random search of 20 values in the range of $[0.000139, 0.00712]$. Each parameter setting was run for three cycles and the results can be seen in table 2. The best network came from having a λ of 0.004018416 which resulted in an accuracy of 52.04%. The second best was $\lambda = 0.003211415$ with an accuracy of 52.04%. Lastly, the third best was $\lambda = 0.000321934$ with an accuracy of 51.96%.

5 Training with the best lambda setting

The results from training with the best hyperparameters can be seen in figures 9, 10, and 11. Here I trained with 49000 training images and trained for three cycles. Results were plotted 9 times per cycle. This resulted in a final test accuracy of 50.12%.

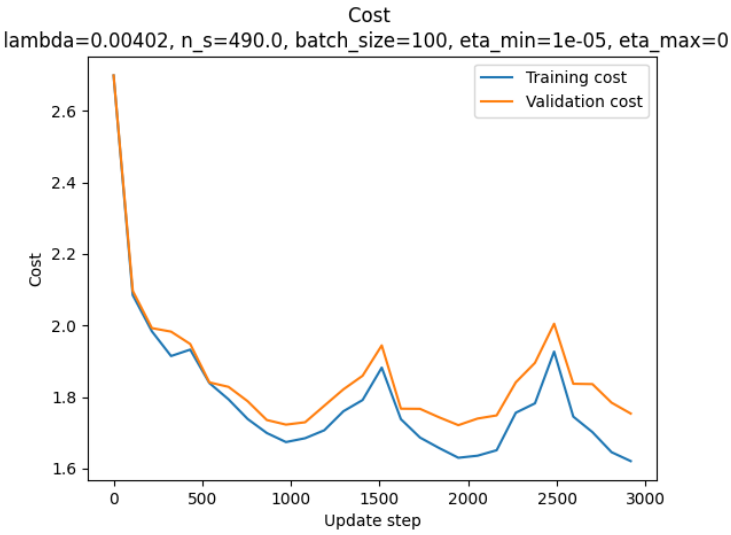

Figure 9: Graph of the training and validation cost for the best lambda value. eta_max was in this case 0.01.

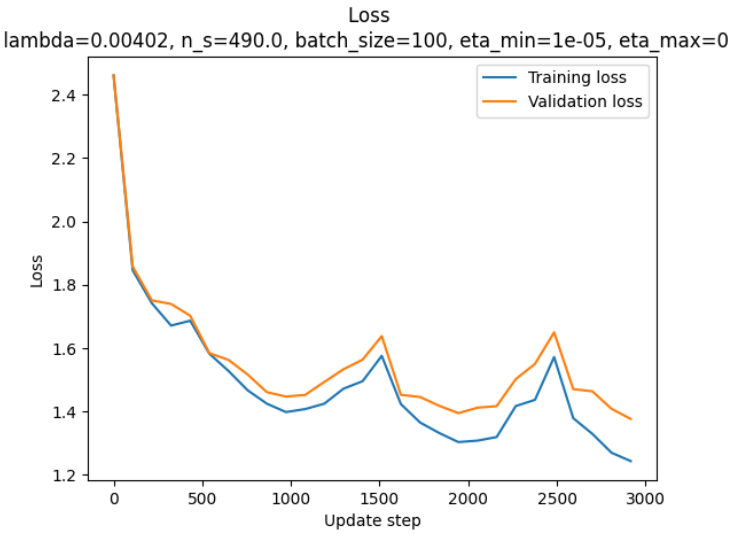

Figure 10: Graph of the training and validation loss for the best lambda value. eta_max was in this case 0.01.

λ	Best validation accuracy
0.003216161	0.5146
0.000321934	0.5196
0.004018416	0.5206
0.003211415	0.5204
0.003105866	0.5146
0.002470422	0.5146
0.001583341	0.5112
0.004509707	0.5112
0.002253884	0.5076
0.002022192	0.507
0.004522854	0.5052
0.003873585	0.4992
0.001088802	0.4972
0.003763736	0.4924
0.001440709	0.4918
0.005681772	0.4916
0.006166228	0.4908
0.003627227	0.4894
0.006113902	0.4818
0.000701080	0.4802

Table 2: Results from the fine search across different values of λ. The accuracy is the best validation accuracy the network received with the corresponding lambda.

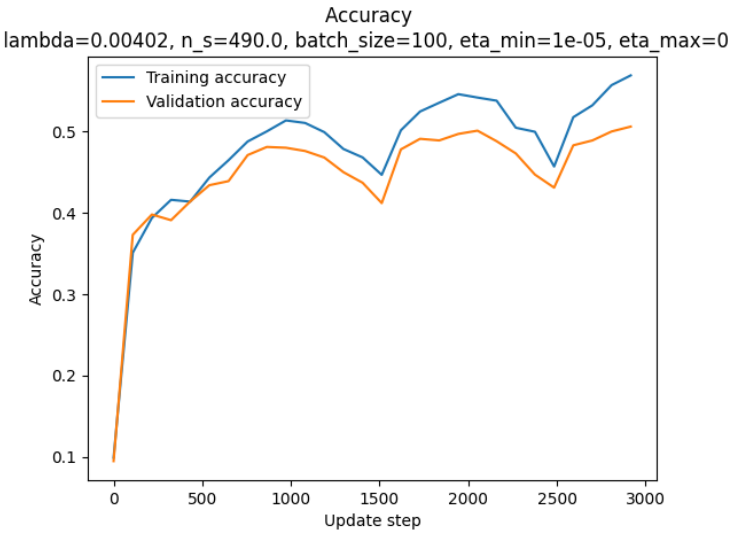

Figure 11: Graph of the training and validation accuracy for the best lambda value. eta_max was in this case 0.01.