Assignment 2

Fred Lindahl (`frlinda@kth.se`), 950213-7772
Deep Learning in Data Science, DD2424

Checking the gradient

Computing the gradient analytically was pretty straight forward using the instructions from the lectures. To test the precision of the analytically computed gradient I used the suggested method given in the lab instructions:

$$\varepsilon = \frac{|g_a - g_n|}{\max(\text{eps}, |g_a| + |g_n|)}$$

where g_a and g_n are the values of the analytically and numerically computed gradients, respectively and eps is a small number, $\sim 10^{-10}$. I computed the gradient using two images with reduced dimension (20) to keep down the computation time for the numerical method (using the provided function `ComputeGradsNumSlow`). From that I obtained

$$\varepsilon_{W1} = 7.5775 \cdot 10^{-11}$$
$$\varepsilon_{W2} = 4.9706 \cdot 10^{-10}$$
$$\varepsilon_{b1} = 8.0865 \cdot 10^{-11}$$
$$\varepsilon_{b2} = 3.7252 \cdot 10^{-11}$$

In addition, I trained the network on a small amount of the data, 100 images with no regularisation for 200 epochs to see if I was able to overfit the data using mini-batch gradient descent. I computed the cost function for each epoch for the small training sample and for the validation data to check this. The results are shown in Figure 1

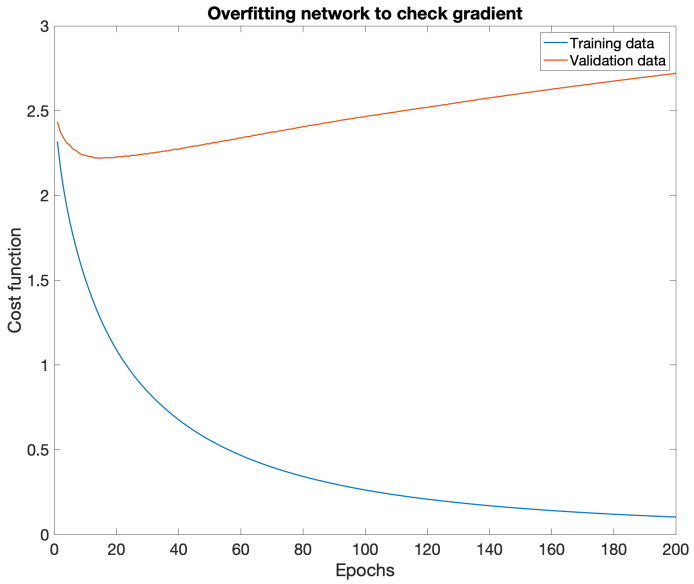

Figure 1: Overfitting the network to check validity of gradient

The data is clearly overfitted, since the cost function for the validation data *increases* the more we train the network, while the cost function for the training data decreases quickly. This indicates that our our gradients computations and mini-batch algorithms are okay.

Exercise 3: training the network with cyclical learning rates

In this part I am trying to reproduce the plots from Figure 3 in the lab instruction to make sure my implementation is bug free. Using the same parameters as suggested in the instruction I was able to obtain the results shown in Figure 2 with a test accuracy of `45.86%`.

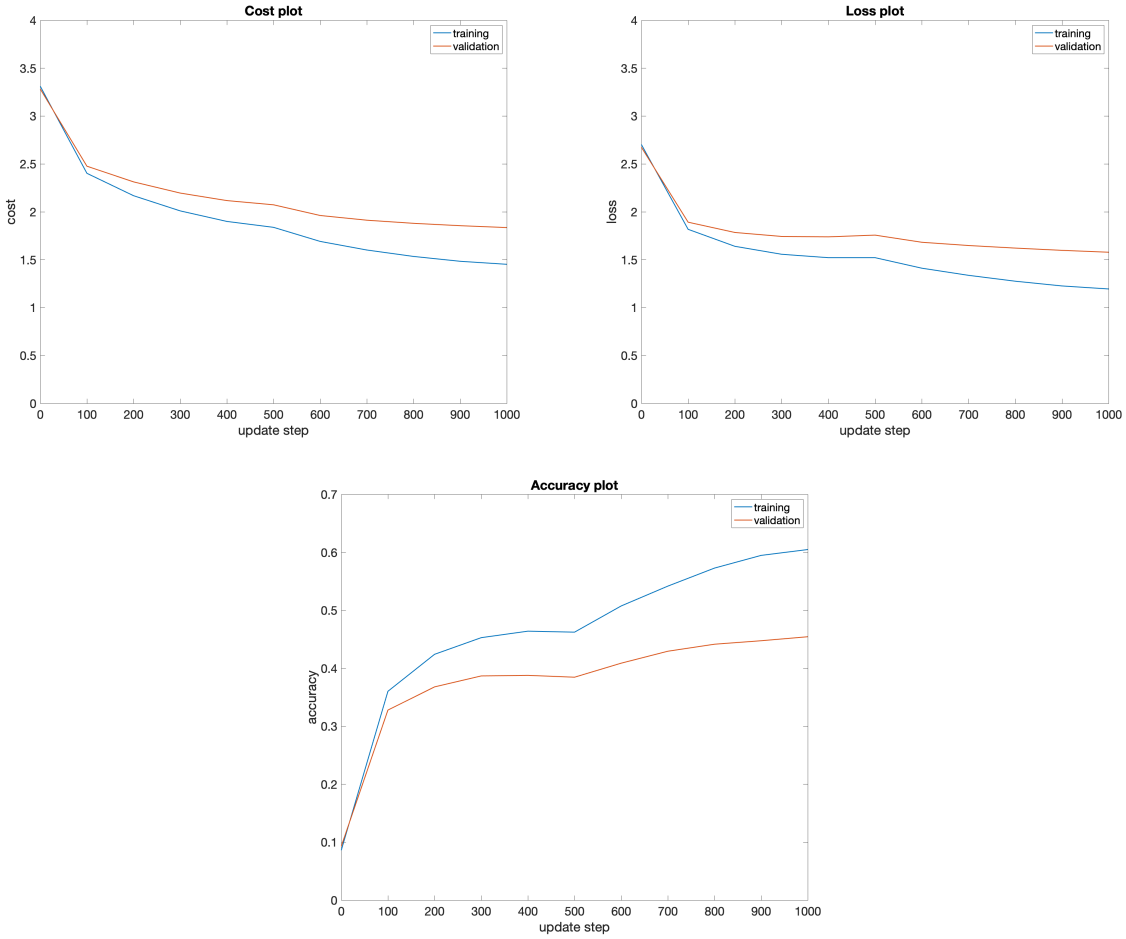

Figure 2: Attempt to reproduce results from Figure 3 of lab instruction

The plots look similar to the ones in the lab instruction. The cost and loss plots decrease steadily after the first $\sim$ 100 steps, with the validation line slightly above than the training line, which is expected. We see the same behaviour from the Accuracy plot: after an initial rapid increase in accuracy, the line levels off, with the training data showing higher accuracy than the validation.

Exercise 4: training the network for real

Again, I am trying to reproduce results from the instructions, now from Figure 4. I set `ns = 800` and perform 3 cycles, leaving the other parameters unchanged. From this I obtain the results shown in 3 with a test accuracy of `47.53%`.

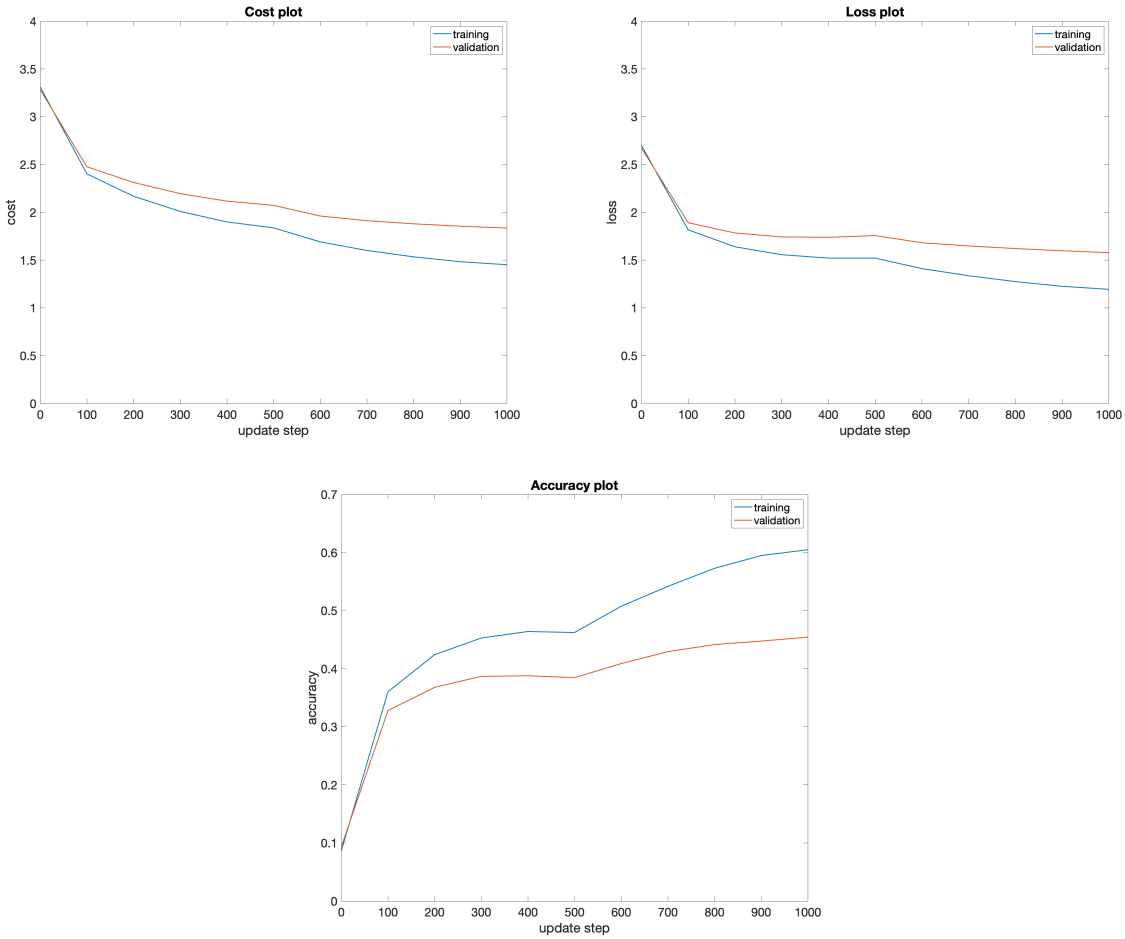

Figure 3: Attempt to reproduce results from Figure 4 of lab instruction

These plots look similar to those from the instruction and exhibit the same behaviour as in the previous exercise.

Coarse-to-fine random search to find `lambda`

Coarse search

For my coarse search for `lambda`, I used a uniform grid of eight values $l \in [-5, -1]$ such that $\lambda = 10^l$. All available images were used, bar 5000 for validation. The parameter `ns` was set to `2*floor(n/nbatch)` where $nbatch = 100$ and two cycles were used for the mini-batch. Otherwise, all parameters remained unchanged from the previous part of this lab. The best accuracies on the validation set are shown in table 1 together with the corresponding `lambda` value. It should be noted that the values for lambda have been rounded off to four decimal points.

`lambda`	$\varepsilon_{\text{validation}}$
0.0001389	52.48 %
0.0005179	52.30 %
0.001931	53.56 %

Table 1

Fine search

For the fine random search, I used eight values for lambda spread randomly around 0.00193069 since that achieved the best result for the coarse search. This time I used three cycles for the mini-batch, while keeping the other parameters the same. The results are presented in 2.

`lambda`	$\varepsilon_{\text{validation}}$
0.00193069655428538	53.56 %
0.00193069121154068	53.56 %
0.00193069042544098	53.56 %

Table 2

As can be seen from these results, the best value for `lambda` was found already during the coarse search.

Final results

Finally, using the best value for `lambda` found above, together with four cycles, `ns = 2*floor(n/nbatch)` and all available data, bar 1000 for validation I was able to achieve a test accuracy of `52.02%`. The loss plot is shown in Figure 4

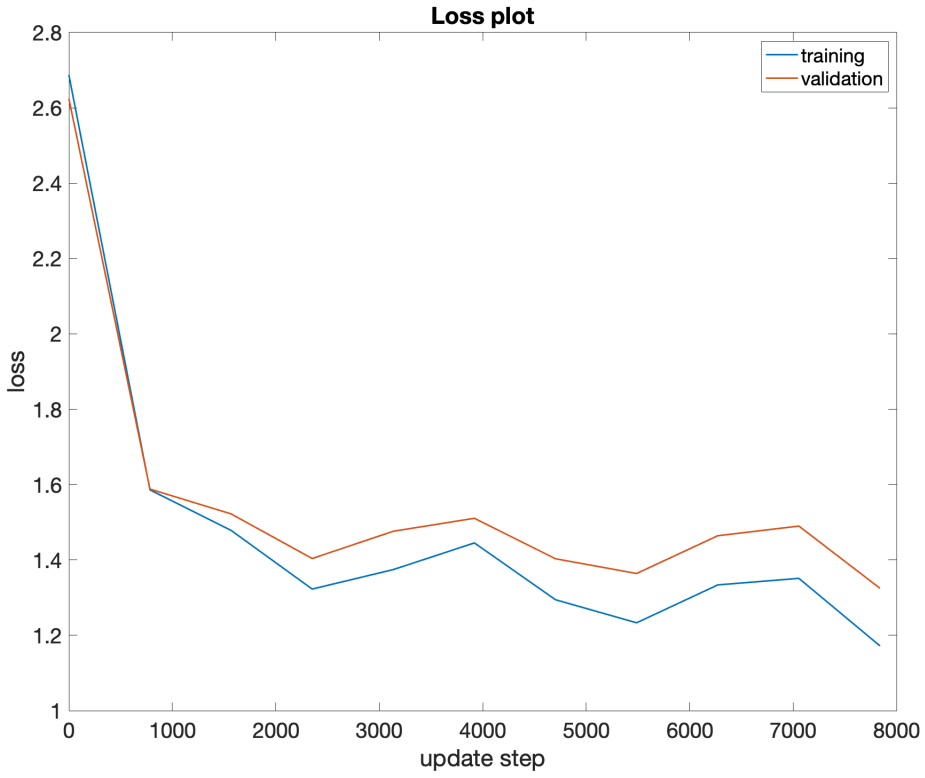

Figure 4: Loss plot of training of final network. Data collected at 11 points of training.