Assignment 1 DD2424

Alexandra Hotti

June 12, 2019

1 Introduction

This report contains the results from an assignment were a feed forward neural network was used to classify the images of the Cifar-10 data set.

2 Relative Error between Analytical and Numerical Gradients

This section contains the relative computations between the numerical and analytically learned gradients computed for the weights and biases. The analytical gradients were computed using the function: *ComputeGradsNum*. The relative errors was computed using Equation 1.

$$\frac{|g_a - g_n|}{max(\epsilon, |g_a| + |g_n|)} \tag{1}$$

The figures below contain boxplots for the relative error for three different batch sizes (see the figure below) where all 3072 dimensions of the first 100 data points in *data_batch_1.mat* were used.

The parameter setting for these plots where $\epsilon = 0.1$ and $\lambda = 0$. The relative error differences ranged between 0 and up to the order of 10-7. Which is smaller than 1e-6 (a value provided for comparison on p.7, in Assignment 1). The relative error was also computed using a λ of 0.1 and the relative errors remained equivalently small.

Since the relative error between my analytical gradient and the numerical gradients was sufficiently small it was concluded that the gradients were computed accurately.

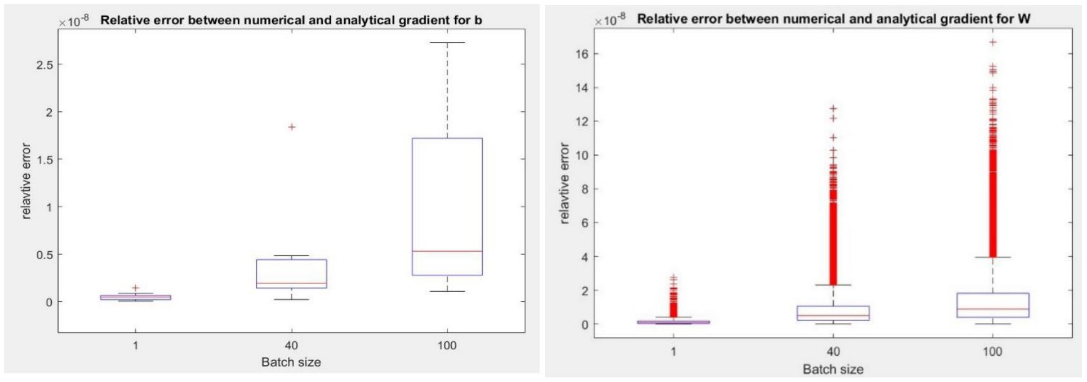

Figure 1: Boxplots for relative errors between the analytically and numerically computed gradients.

3 Results

This section first contains the results for the different parameter settings given in the assignment description. Lastly the section contains a note on the effect of increasing regularization as well as the significance of picking a correct learning rate.

3.1 Parameter setting 1

First the following parameters were used to train an one layer network. Using this setting a final test set accuracy of 27.07 % was obtained.

$$\lambda = 0, epochs = 40, batchsize = 100, eta = 0.1$$

Here are the training and validation costs and losses computed after every epoch of the mini-batch gradient descent algorithm using the above parameter setting.

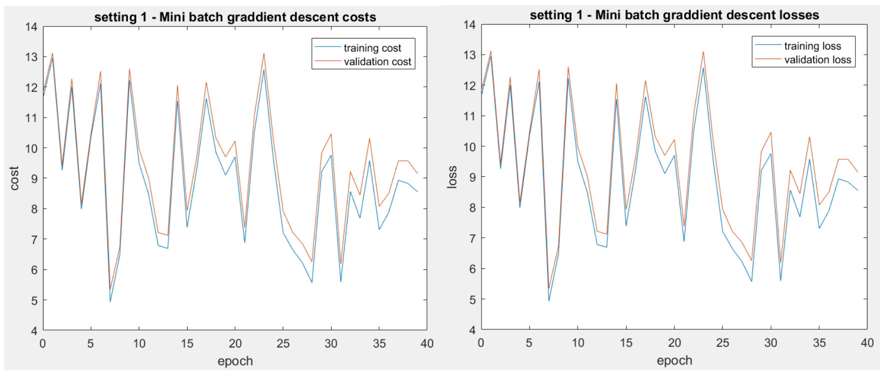

Figure 2: Costs and losses for parameter setting 1.

Here are the images representing the learnt weights after the training using the above parameter setting.

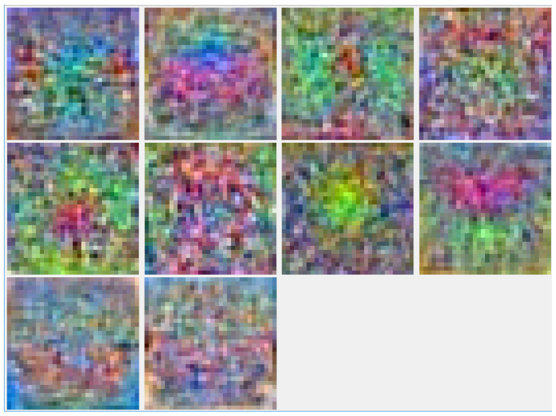

Figure 3: Learnet weights for parameter setting 1.

3.2 Parameter setting 2

Next, the following parameters were used to train an one layer network. Using this setting a final test set accuracy of 36.65 % was obtained.

$$\lambda = 0, epochs = 40, batchsize = 100, eta = 0.01$$

Here are the training and validation costs and losses computed after every epoch of the mini-batch gradient descent algorithm using the above parameter setting.

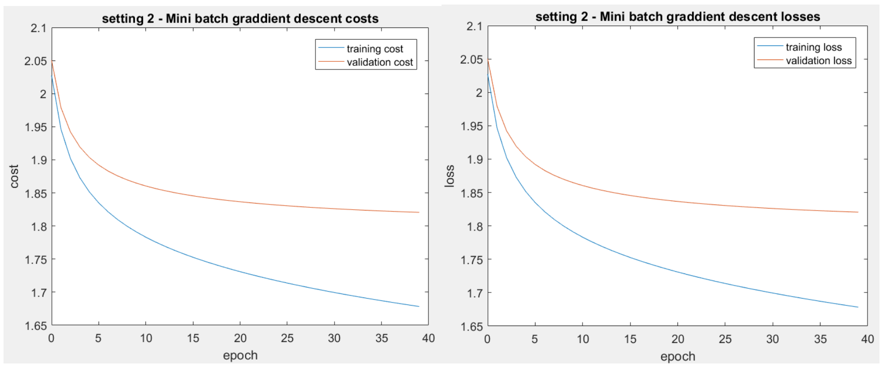

Figure 4: Costs and losses for parameter setting 2.

Here are the images representing the learnt weights after the training using the above parameter setting.

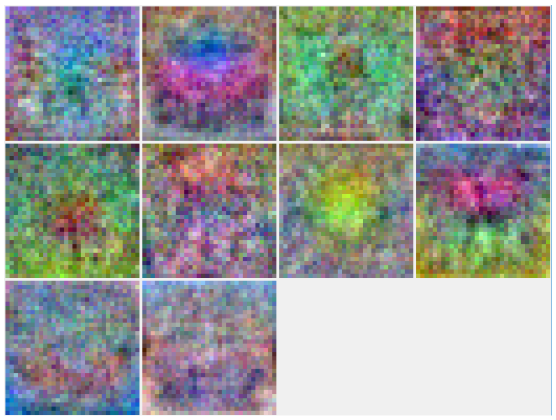

Figure 5: Learnt weights for parameter setting 2.

3.3 Parameter setting 3

Next, the following parameters were used to train an one layer network. Using this setting a final test set accuracy of 33.37 % was obtained.

$$\lambda = 0.1, epochs = 40, batchsize = 100, eta = 0.01$$

Here are the training and validation costs and losses computed after every epoch of the mini-batch gradient descent algorithm using the above parameter setting.

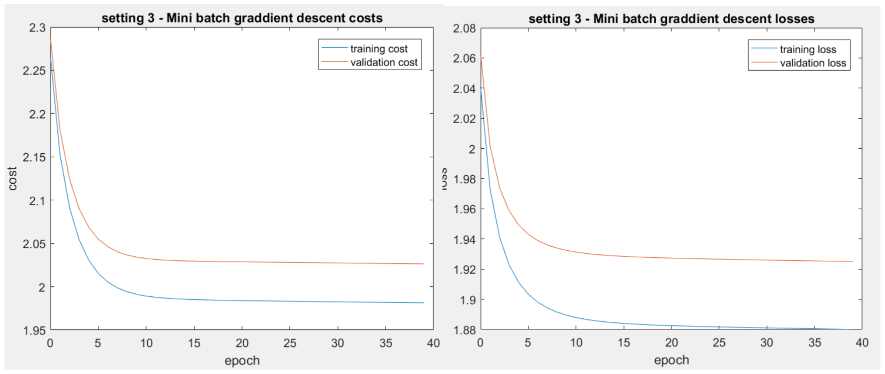

Figure 6: Costs and losses for parameter setting 3.

Here are the images representing the learnt weights after the training using the above parameter setting.

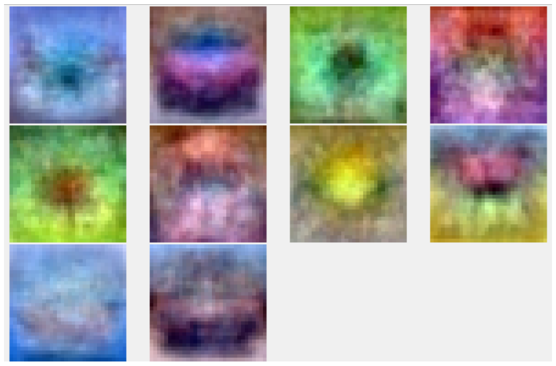

Figure 7: Learnt weights for parameter setting 3.

3.4 Parameter setting 4

Next, the following parameters were used to train an one layer network. Using this setting a final test set accuracy of 21.92 % was obtained.

$$\lambda = 1, epochs = 40, batchsize = 100, eta = 0.01$$

Here are the training and validation costs and losses computed after every epoch of the mini-batch gradient descent algorithm using the above parameter setting.

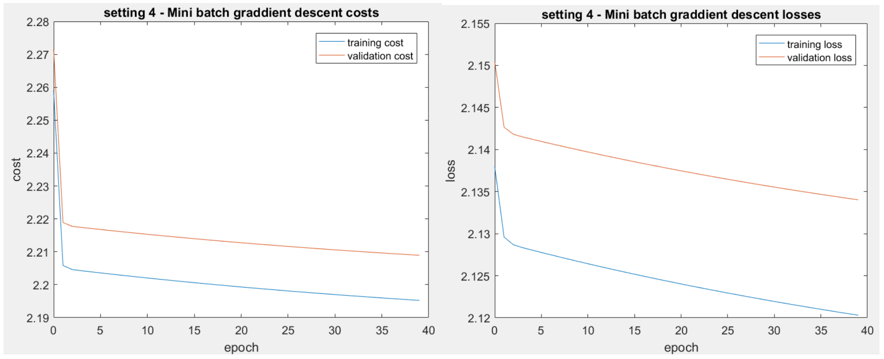

Figure 8: Costs and losses for parameter setting 4.

Here are the images representing the learnt weights after the training using the above parameter setting.

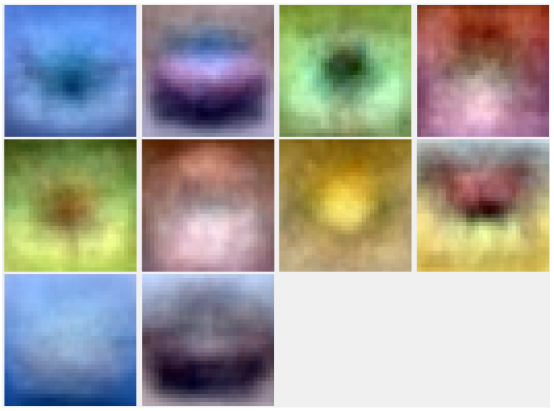

Figure 9: Learnt weights for parameter setting 4.

4 Conclusions

4.1 Importance of correct learning rate – eta

When we use a to large learning rate (eta) we risk missing the minimum function value by *"jumping over it"* as we get close to it. Instead we go towards a higher value at a different point during gradient descent. This is perhaps what is going on above when we use parameter setting 1, where we use an eta of 0.1. In this graph the plot of the costs and losses jump up and down during the gradient descent, i.e. these values do not stabilize during training. If we instead look at parameter setting 2-4, where we use a smaller learning rate, the graphs of the costs and losses stabilizes during training.

Thus, when we use a smaller learning rate, we can be more certain that we will find an optimal solution. However, doing so also means that finding that optimal solution will take more time.

4.2 Increasing regularization – λ

The purpose of regularization is to prevent overfitting to the training data set. This could be the case in parameter setting 2 where we use a lambda of 0. Here the plotted losses and costs seem to diverge in opposite direction.

However, if one uses to much regularization performance could become worse. This seems to be the case in the last parameter setting when I used a lambda of 1. This is shown in the plots of the costs and losses on the training and validation data sets, where difference in performance between the datasets was evidently larger compared to when we used a smaller lambda in parameter setting 3.