Kungliga Tekniska Högskolan

DD2424 Deep Learning in Data Science
Assignment 1

Quentin LEMAIRE

May 7, 2018

1 Gradient validation

In order to validate my computation of the gradient, I compared my values with of the gradient with those of a numerically computed gradient function. To have a better estimate of the difference between both, I used the relative error. I decided to work on a subset of the data of 100 images to compute the gradients faster.

The result after one epoch was good, so I did it over more epoch to be sure about my conclusion (not so many because it is quite long to compute). For two of these epochs, I got the following results:

- For grad W: 3.6040215677237976e-07 and 6.476252872510549e-07
- For grad b: 6.2740359942424493e-07 and 1.4162336944826677e-06

The results are quite good. Those results were with the faster version of the numerically computed gradient thus less precise version. Therefore, I ran the same tests with the slower version to comfort my idea that it was right. I got the following results for two of the epochs :

- For grad W: 2.7847502237389628e-08 and 1.1017004743806446e-08
- For grad b: 1.513953779983363e-08 and 6.6321318074255186e-09

The results are even better, the relative error is very low. For all the epochs I did, the results were similar. Therefore, I assumed that my gradient calculations were correct for the rest of the assignment.

2 Graphics and images

From the training of the network, we can plot the following figures:

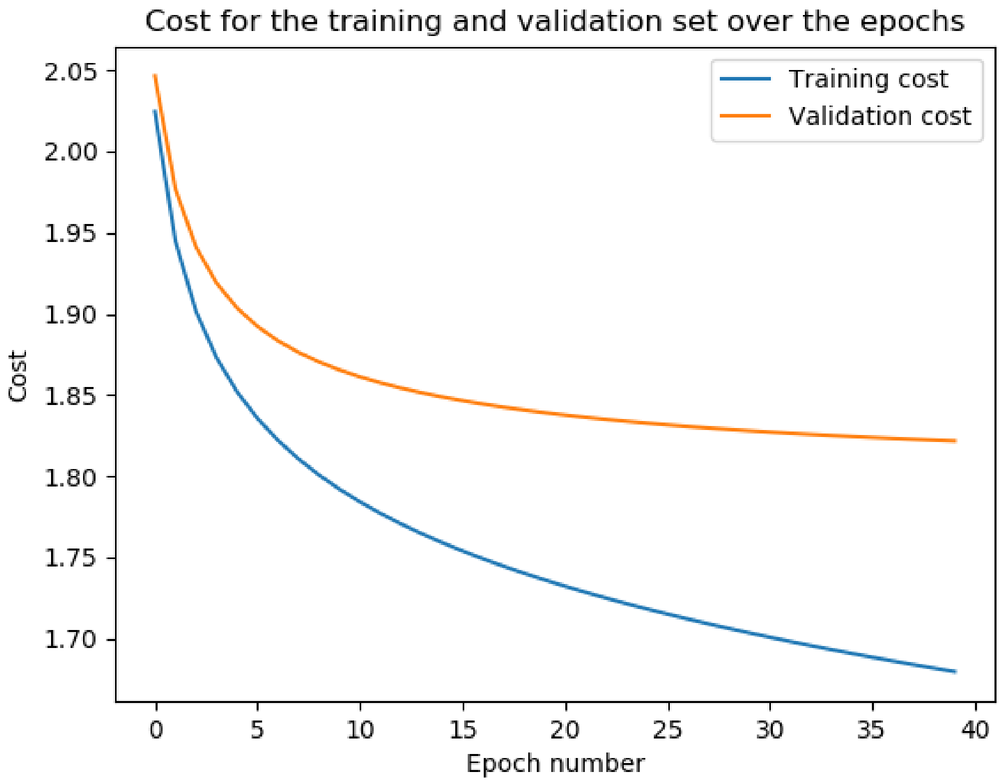


Figure 1: Cost for the training and the validation set over the epochs.

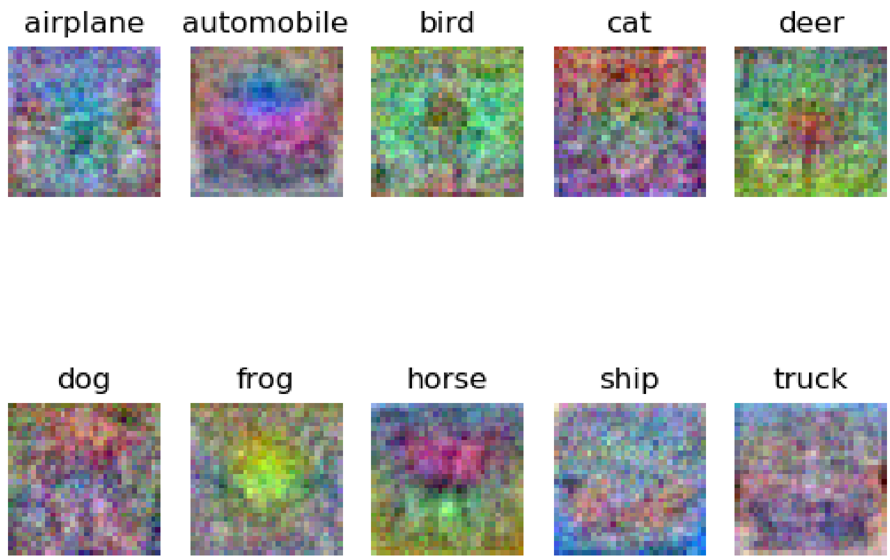


Figure 2: Image representation of the weights of the network.

We can see that it is hard to recognize the class behind the weights. It might be one of the limits of this simple architecture, we can't really get different representation for one class as we only have one weight for one class.

3 Performances

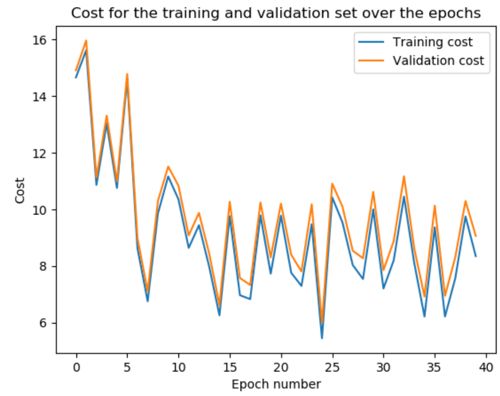
In order to have an idea of the influence of each parameters, some tests with different parameters were made:

- lambda = 0, n epochs = 40, n batch = 100, eta = 0.1, accuracy: 0.2676
- lambda = 0, n epochs = 40, n batch = 100, eta = 0.01, accuracy: 0.3836
- lambda = 0.1, n epochs = 40, n batch = 100, eta = 0.01, accuracy: 0.3063
- lambda = 1, n epochs = 40, n batch = 100, eta = 0.01, accuracy: 0.2295

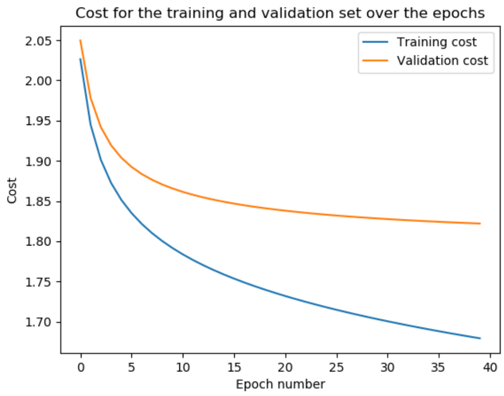
The first two examples are showing the importance of a correct learning rate. If the learning rate is to high, we will kind of oscillate around the local minima without finding it. With a lower learning rate, we were able to go much closer to this minima. In both cases, we are not sure that the minima we were going for is the global one.

The two last examples are showing the influence of increasing the amount of regularization. In our case, increasing the regularization too much causes the accuracy to decrease. The constrain on the weight were too big and they were unable to go toward to local minima like they did without any constrain. However, in some cases, increasing the amount of regularization can be a way to avoid the overfitting of the training set.

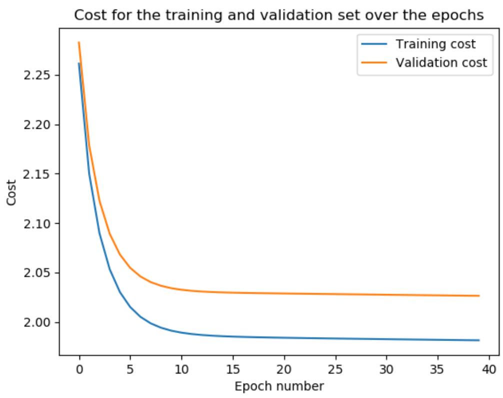
We can also plot the previous graphs for each set of parameters we tried before:



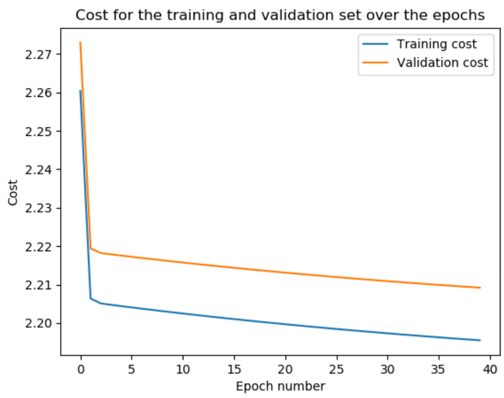
(a) lambda = 0, n epochs = 40, n batch = 100, eta = 0.1



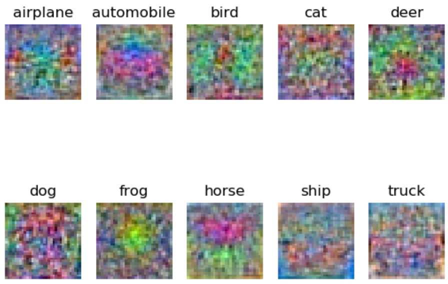
(b) lambda = 0, n epochs = 40, n batch = 100, eta = 0.01



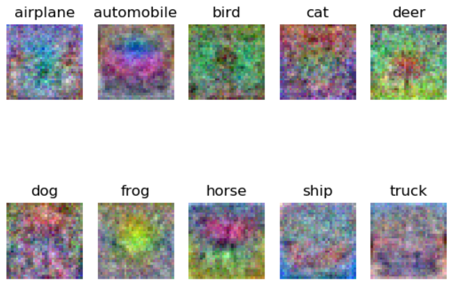
(c) lambda = 0.1, n epochs = 40, n batch = 100, eta = 0.01



(d) lambda = 1, n epochs = 40, n batch = 100, eta = 0.01

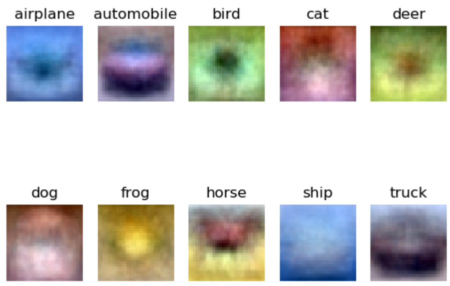


(e) lambda = 0, n epochs = 40, n batch = 100, eta = 0.1



(f) lambda = 0, n epochs = 40, n batch = 100, eta = 0.01

(g) lambda = 0.1, n epochs = 40, n batch = 100, eta = 0.01



(h) lambda = 1, n epochs = 40, n batch = 100, eta = 0.01