Deep Learning DD2424 - Assignment 1

Anton Stagge

March 2019

1 Introduction and test against numerical gradient

In this assignment I have created a one layer network that is trained on CIFAR-10 images. I have succeeded in implementing all the functions, including the one calculating the gradient analytically.

I had several issues at first computing both the numerical and analytic gradient, since I decided to do this assignment in python and not Matlab. I implemented a function to compute the relative error for the two gradients, and started by just calculating the gradients with one single input vector. At first the error was sky high. I then realized that the Matlab code $W\_try = W$; needed to copy the matrix W, which the python code did not do. After fixing this bug I was sure that the numerical functions were implemented correctly, and sure enought the relative error was reduced to around $2.3e-6$ for W. I then proceeded to increase the amount of input vectors to 2, which caused the error to rise to 0.33, which is too high. I soon found a bug in my *compute_cost* function, specifically where I calculated $y^T p$. In the background section of the assignment instructions, p and y where $Kx1$ vectors, but in my functions they were Kxn matrices. Therefore, I had to take the diagonal of $y^T p$.

In the end, the relative error between the numerically and analytically computed gradient for a mini-batch of 100 samples: Using **forward** difference:
For W: $2.8972593586024902 \times 10^{-6}$
For b: $1.741924518732855 \times 10^{-7}$
Using **central** difference:
For W: $4.434541038005285 \times 10^{-8}$
For b: $4.2280019721913735 \times 10^{-8}$

2 Results

For all of the following sections, n_batch and n_epochs were 100 and 40 respectively.

2.1 lambda$= 0$ and eta$= 0.1$

The accuracy of the model: 0.267300

Figure 1: The graph of the training and validation loss computed after every epoch. The network was trained with the following parameter settings: eta=0.1 and lambda=0.

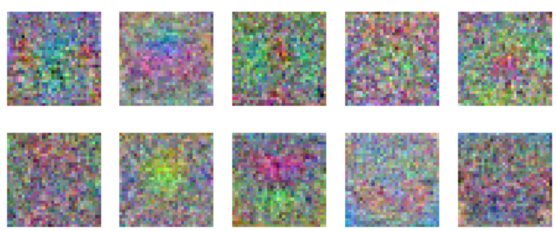

Figure 2: The learnt W matrix visualized as class template images. The network was trained with the following parameter settings: eta=0.1 and lambda=0.

2.2 lambda$= 0$ and eta$= 0.01$

The accuracy of the model: 0.344200

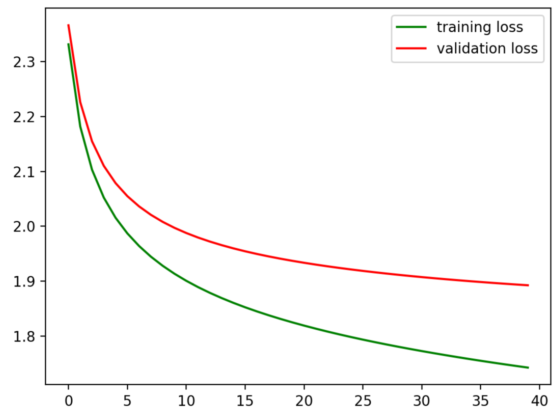

Figure 3: The graph of the training and validation loss computed after every epoch. The network was trained with the following parameter settings: eta=0.01 and lambda=0.

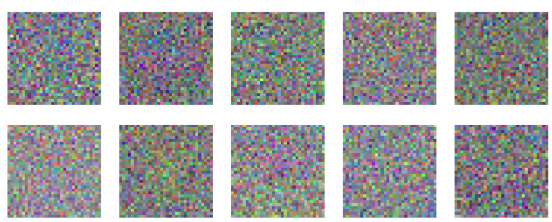

Figure 4: The learnt W matrix visualized as class template images. The network was trained with the following parameter settings: eta=0.01 and lambda=0.

2.3 lambda$= 0.1$ and eta$= 0.01$

The accuracy of the model: 0.333100

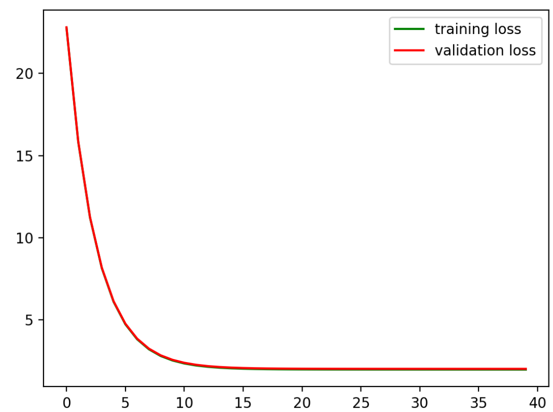

Figure 5: The graph of the training and validation loss computed after every epoch. The network was trained with the following parameter settings: eta=0.01 and lambda=0.1.

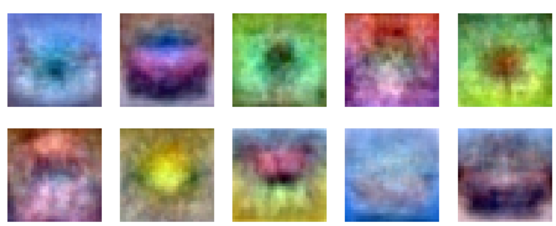

Figure 6: The learnt W matrix visualized as class template images. The network was trained with the following parameter settings: eta=0.01 and lambda=0.1.

2.4 lambda$= 1$ and eta$= 0.01$

The accuracy of the model: 0.219100

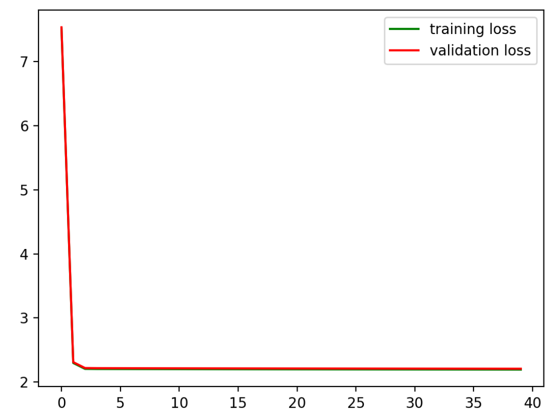

Figure 7: The graph of the training and validation loss computed after every epoch. The network was trained with the following parameter settings: eta=0.01 and lambda=1.

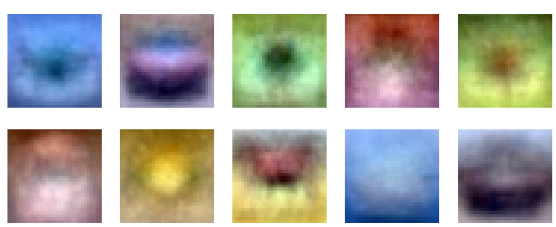

Figure 8: The learnt W matrix visualized as class template images. The network was trained with the following parameter settings: eta=0.01 and lambda=1.

3 Discussion

As we can see from the result, it is very important to have a good *eta*, a good learning rate. Figure 1 displays the effect of having a too large learning rate. The loss sometimes increases since we take too large steps each iteration. Figure 3 has a much smoother curve where the loss is always decreasing.

Also important is the effect of *lambda*, the amount of regularization. The regularization decreases the risk of overfitting to the training data. Therefore if you compare Figure 3 and Figure 5, you can see that the two curves for training and validation loss are much closer to each other in Figure 5. However, with a too large *lambda*, the model accuracy is going to get worse, which is shown if you compare the model having $lambda = 0.1$ and accuracy 0.33 with the model having $lambda = 1$ and accuracy 0.22.