DD2424 - Assignment 1

Marcus Jirwe

March 2020

Introduction

The purpose of the assignment was to train a multi-linear classifier with one layer to perform classifications on images from the CIFAR-10 dataset. For use in the assignment we were given several functions of matlab code which was intended to be used to calculate the gradient to the loss function numerically and to represent the images that could be retained from the trained weight matrices of the classifier. Practically, this report relies on code programmed in python and primarily using the packages "numpy" and "matplotlib". Thus the numerical gradient function were adapted from matlab code to corresponding python code and the function to show the image was rewritten to make better use of available functions in the "matplotlib" library.

1 Practical and implementation

The code relies heavily on the use of the "numpy" library with "matplotlib" being used for plotting the results. Care was taken so as to ensure the dimensions of the matrices were the same as the one in the assignment description, which made it easier and more straightforward to follow the instructions. The testing was performed in the code's main function. The assignment necessitated the implementation of the analytical form of the cross-entropy loss and its gradients as opposed to relying on the numerical gradient. To do this, a derivation shown in the lecture notes were followed and the equations were extended to matrix form. To test that the results of the analytical gradient function was correct, it was tested against the values from the more accurate numerical gradient function using the centered difference. The testing was performed by calculating the Frobenius norm of the difference of the two gradients and dividing it with the sum of the individual norms. This gives a measure of a relative difference between the two. It should be noted that the similarity between the bias gradients were much higher, but since the weight calculation then have lower accuracy it will not have much of an impact.

Numerical vs Analytical gradient

More specifically, the relative difference between the gradients is calculated by the equation:

$$diff = \frac{||grad_{analytical} - grad_{numerical}||}{||grad_{analytical}|| + ||grad_{numerical}||}$$

The numerical difference between the two were:
For the W-gradients: 1.811144074805989e-08
For the b-gradients: 3.034593391334638e-10
Which seems small enough to assume the calculations to be correct. The results below seem to lend credit to this conclusion.

2 Results

Below are are plots of the loss, the learned weight matrix rows projected onto 32x32 pixel images with 3 channels each, and the resulting accuracy scores on all the data sets for each of the given parameter settings. The accuracy is reported as the mean combined with the standard deviation for the network after ten iterations of training with 100 mini-batches and 40 epochs each. This is chosen to represent a kind of distribution of the model's accuracy for different degrees of training, from potential under- to overfitting. It should however be noted that due to the low variance nature of the linear classifier, the accuracy for the same degree of training will show very little variation. The same amount of mini-batches and epochs were used for each training iterations for all parameter settings, being 100 and 40, respectively. The loss plots were taken from the first 40 training epochs, when the convergence of the loss was most apparent. The learned weight matrices were instead taken from the end of the ten iterations so as to give as much time for convergence as possible. This might be undesirable if no regularisation is present, however.

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

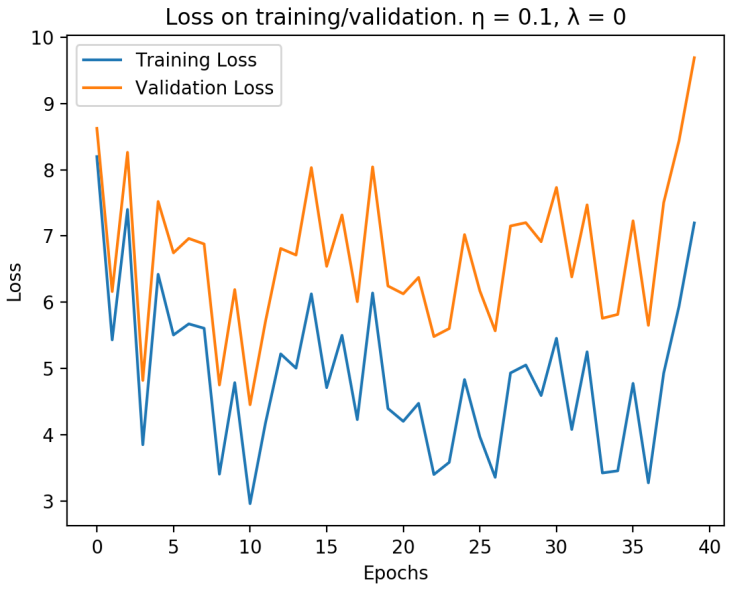

Figure 1: Convergence of the loss function as function of epochs.

It can be seen that for this high a learning rate the loss function and the learned images are very noisy. Indeed, the loss function does not appear to converge properly at all.

Accuracy

Training accuracy: 0.5078 ± 0.0554
Validation accuracy: 0.2698 ± 0.0145
Test accuracy: 0.2740 ± 0.0154

2.2 $\eta = 0.001$ and $\lambda = 0$

The resulting convergence of the loss plot looks better for a lower learning rate. However, since we have no regularisation the learned images are a mess, and most likely the result of

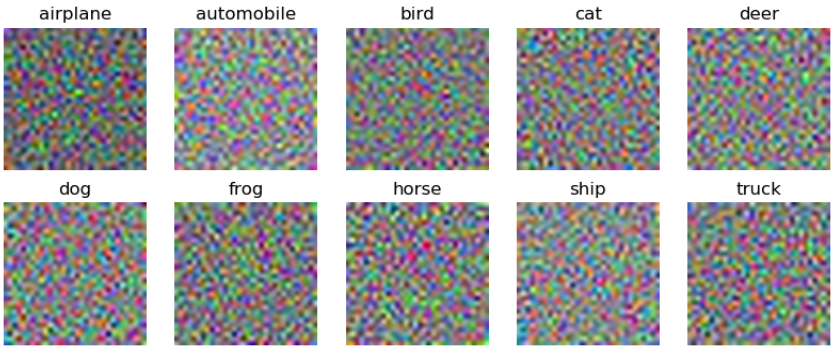

Figure 2: The learned weight matrix rows unraveled into RGB images.

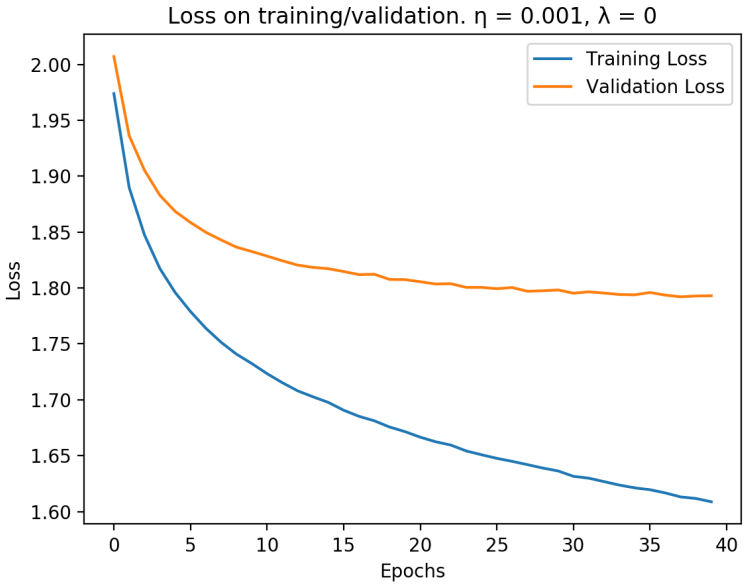

Figure 3: Convergence of the loss function as function of epochs.

Figure 4: The learned weight matrix rows unraveled into RGB images.

the system overfitting.

Accuracy

Training accuracy: 0.5170 ± 0.0272
Validation accuracy: 0.3769 ± 0.0065
Test accuracy: 0.3815 ± 0.0050

2.3 $\eta = 0.001$ and $\lambda = 0.1$

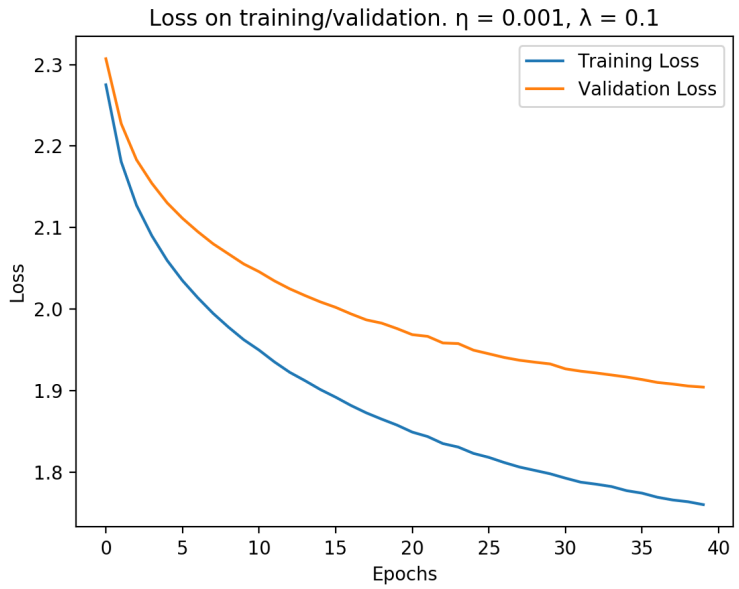

Figure 5: Convergence of the loss function as function of epochs.

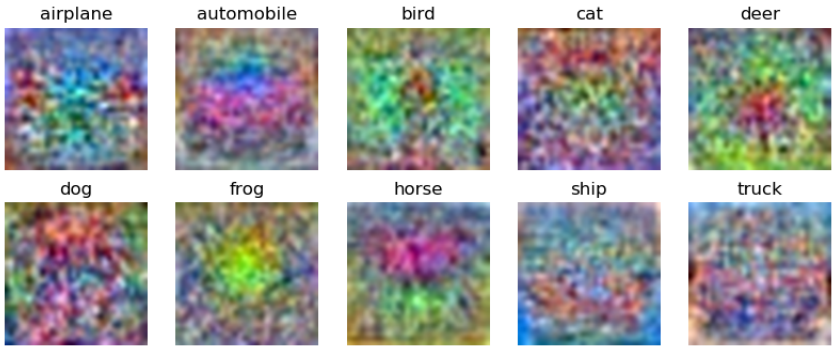

Figure 6: The learned weight matrix rows unraveled into RGB images.

Now the shape of the loss plot looks a little better, still. The higher values of the loss can be chalked up to the magnitude of the regularisation also being present. We can see that the images now having converged to something reasonable, with the ”automobile” and ”horse” classes being the most defined.

Accuracy

Training accuracy: 0.4628 ± 0.0051
Validation accuracy: 0.3903 ± 0.001
Test accuracy: 0.3953 ± 0.0009

2.4 $\eta = 0.001$ and $\lambda = 0.1$

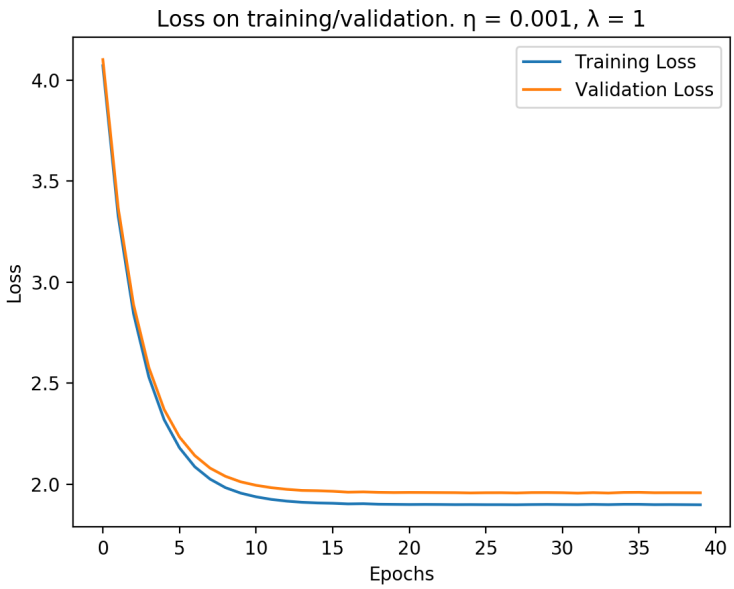

Figure 7: Convergence of the loss function as function of epochs.

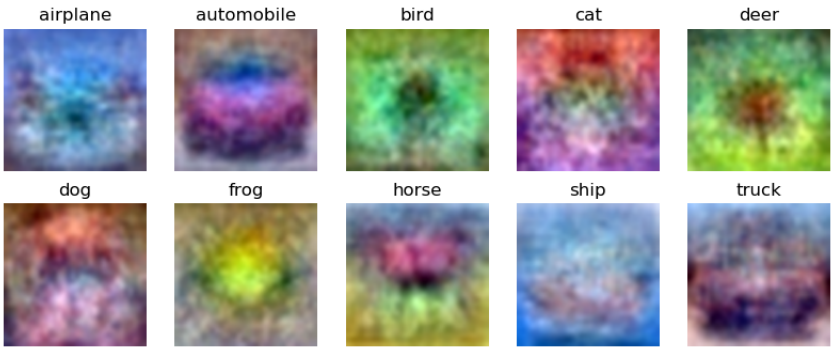

Figure 8: The learned weight matrix rows unraveled into RGB images.

We can see that finally the learned images seem to be the highest quality with this level of regularisation. However, it should be noted that the regularisation strength enforced here seems to prevent further learning after only a few iterations, which is most likely not desirable behaviour, regardless of the quality of the images.

Accuracy

Training accuracy: 0.3999 ± 0.0026
Validation accuracy: 0.3669 ± 0.002

Test accuracy: 0.3766 ± 0.0025

3 Conclusion

As a conclusion, while the performance of this linear classifier is at best rather mediocre (although still far better than classifying at random) it is robust and manages to learn images representing the classes rather well. As expected from the lectures, the classifier seems to have an easier time with learning the class of "automobile" than the others. At least it seems that way to from looking at the images. This would be in line with the performance-by-class curves shown during the lecture slides. As expected, without regularisation the network is prone to overfitting and having poor generalisation. However, too much regularisation prevents the decision boundary from moving too much, also as shown in the lecture slides. This also seems to have an impact on generalisation when comparing the model with the largest regularisation to the one with less.

Learning rate

A proper selection of the learning rate seems to be very important. If the learning rate is too high, it leads to large jumps in parameter space. This in effect leads to bouncing around a local minima or missing them entirely, which could have been avoided with a lower learning rate, see figure 1. Of course, the opposite problem exists with a too low learning rate, where the model learns too slowly or practically not at all. As with many things in deep learning, a balance is necessary.