DD2424 – Assignment 1

Deep Learning

September 25, 2017

Addi Djikic – TSCRM1
addi@kth.se
941028-5473

Abstract

The assignment was successfully finished by training a multi-linear, one-layer classifier by computing the gradient and later using it for a mini-batch gradient decent algorithm to a cost function that computes cross-entropy loss. The CIFAR-10 dataset was used for the training, with different batch sets from the dataset to be used as training, validation and test.

The Multi-Linear Classifier

All mathematical notions can be found in [1] along with [2] and thereof to reduce redundant information about the mathematical background one could look up the equations there.

The steps in the process

Firstly the CIFAR-10 data was loaded and later the weight vector W and bias b where randomly initialized as a Gaussian distribution $G(0, 0.01^2)$. Later a function that evaluates the network was built based on (1) and (2) from [1] and the softmax-function according to Lecture 2 as:

$$Softmax(s) = \frac{exp(s_j)}{\sum_k exp(s_k)}$$

Later a cost function was introduced as (5) from (1) and slide 27 from Lecture 2 in [2]. Then the gradients were computed as (10),(11) from [1] and with the help of slide 91 and 93 in Lecture 3 from [2] as:

$$g = \frac{y^T}{y^T p}(diag(p) - pp^T)$$

$$\frac{\partial J}{\partial b} = g$$

$$\frac{\partial J}{\partial W} = g^T x^T + 2\lambda W$$

Now the final step was to implement the mini-batch gradient decent, and that was done with (8) and (9) from [1]. Function to test the accuracy was also implemented, which gave a result of 10.29% when running it only at the training data, which was reasonable because the dataset is 10 000 images and around 10% if a fair probability if you would only make guesses.

To test that the gradient is correctly computed we can compare it with function that use numerical estimation, therefor we will use `ComputeGradNumSlow` and `ComputeGradNum` from [1] to compare it. The slower one, based on center difference formula, gave the error $3.5023 \cdot 10^{-10}$ for W and $2.1609 \cdot 10^{-10}$ for b when comparing the absolute difference between the maximum values in the vector. For the faster finite difference method the result were that the error difference were $6.0039 \cdot 10^{-8}$ for W and same for b, which are both lower than 10^{-6} as we wanted.

This was tested with reduced input data, which can be seen in the Matlab code for this report.
To train our network we now have a various range of parameters we can tweak. Firstly `lambda` which gains the regularization term, `n-batch` which is the size of the mini-batches, `eta` which is the learning rate, and finally `n-epochs` which is the number of runs through the whole training set we use.

Below we will now try different values of the parameters and plot the cost function with the number of epochs of the training data vs the validation data.

Parameters: lambda=0, n-epochs=40, n-batch=100, eta=.01

Accuracy: 0.3641 (Approximately same as in the lab example with `rng(400)`)

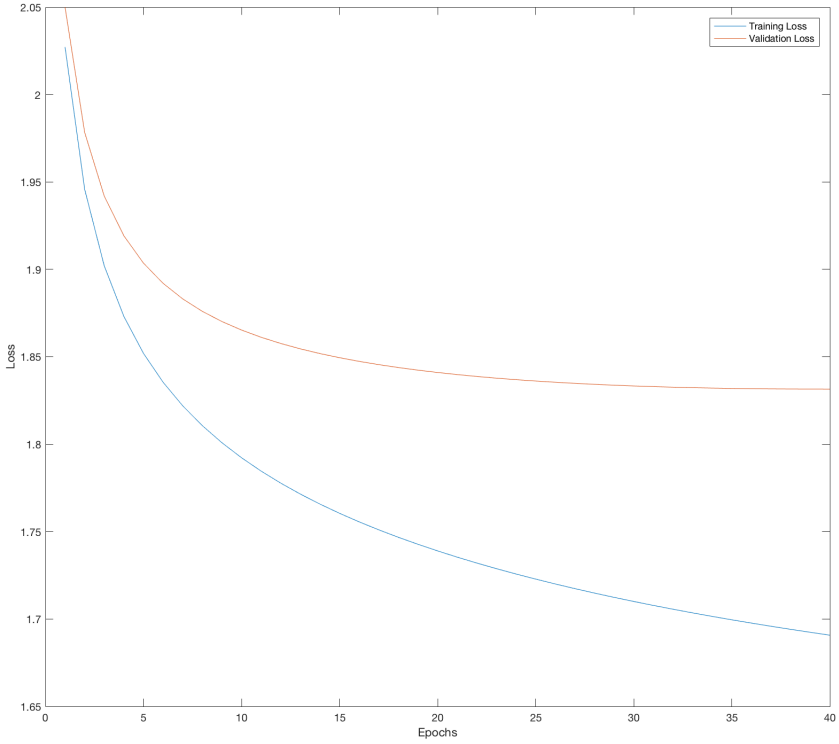

Figure 1:

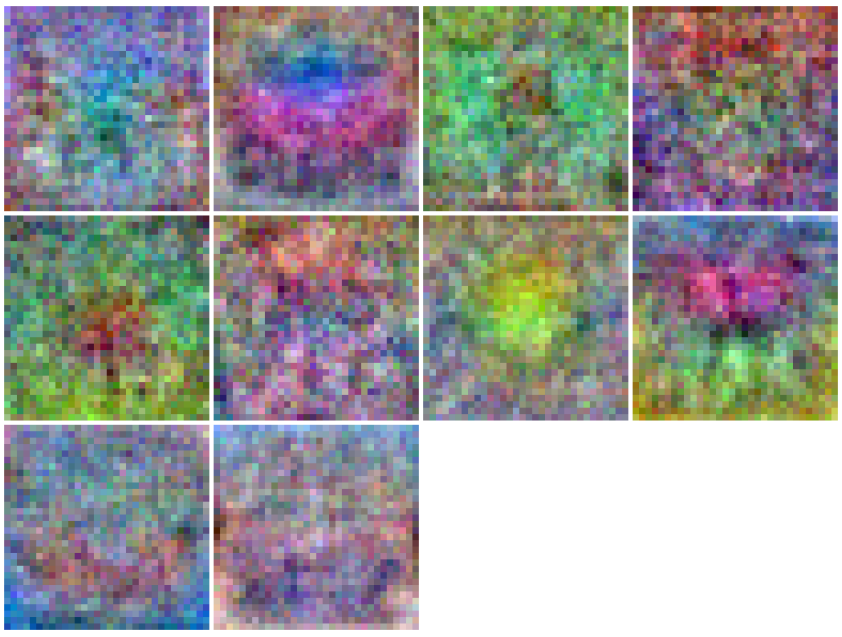

Figure 2:

Parameters: lambda=0, n-epochs=40, n-batch=100, eta=.1

Accuracy: 0.2131

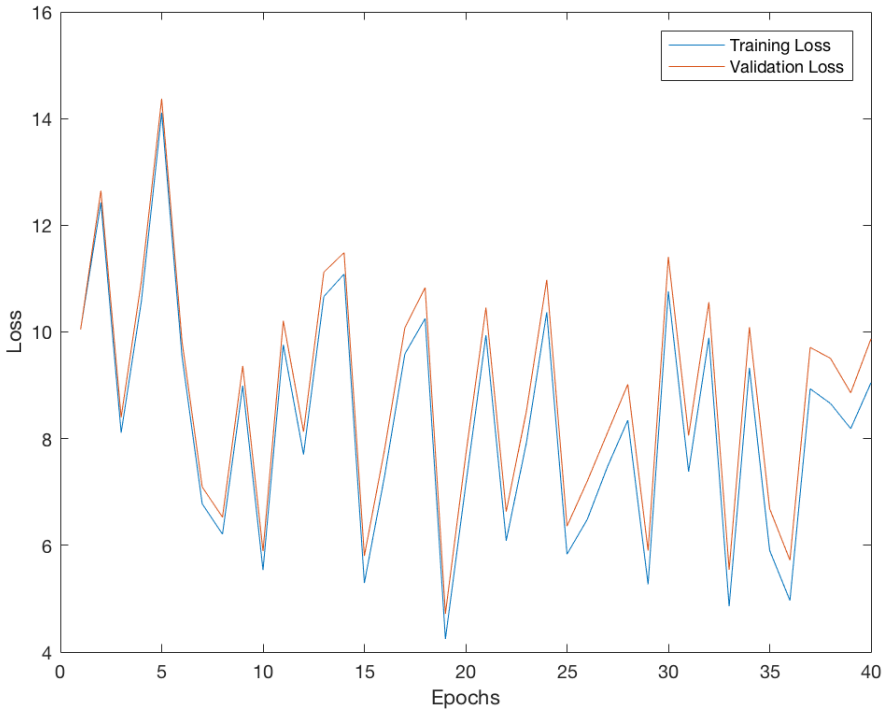

Figure 3:

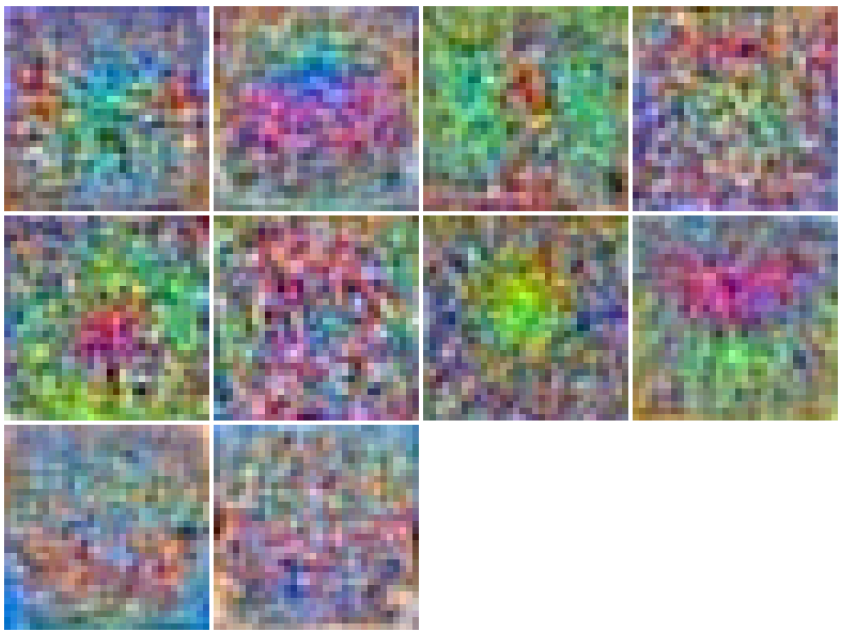

Figure 4:

Parameters: lambda=.1, n-epochs=40, n-batch=100, eta=.01

Accuracy: 0.3180

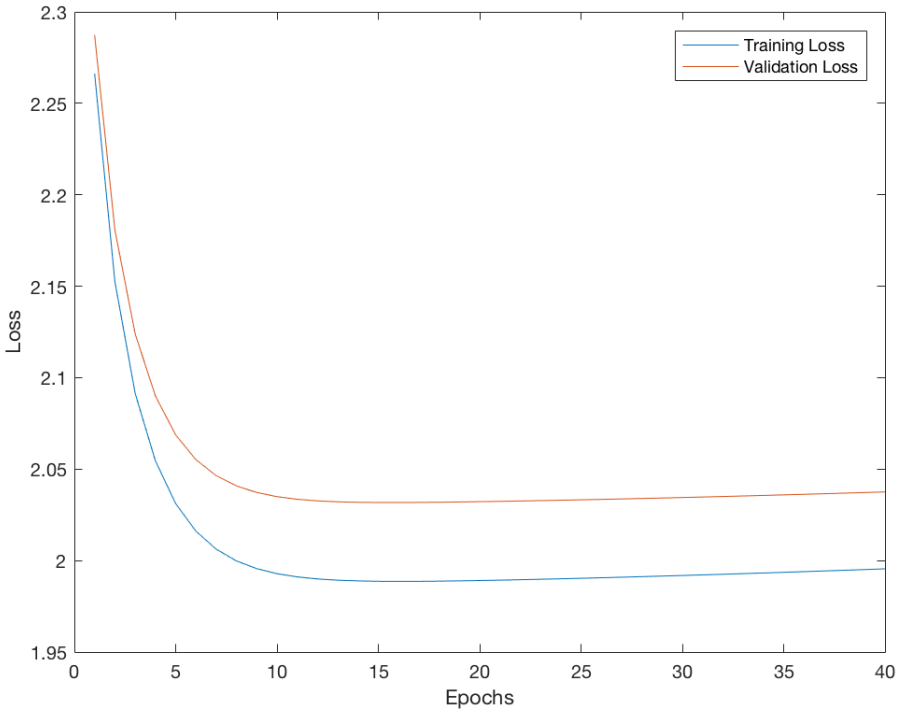

Figure 5:

Figure 6:

Parameters: lambda=1, n epochs=40, n batch=100, eta=.01

Accuracy: 0.2189

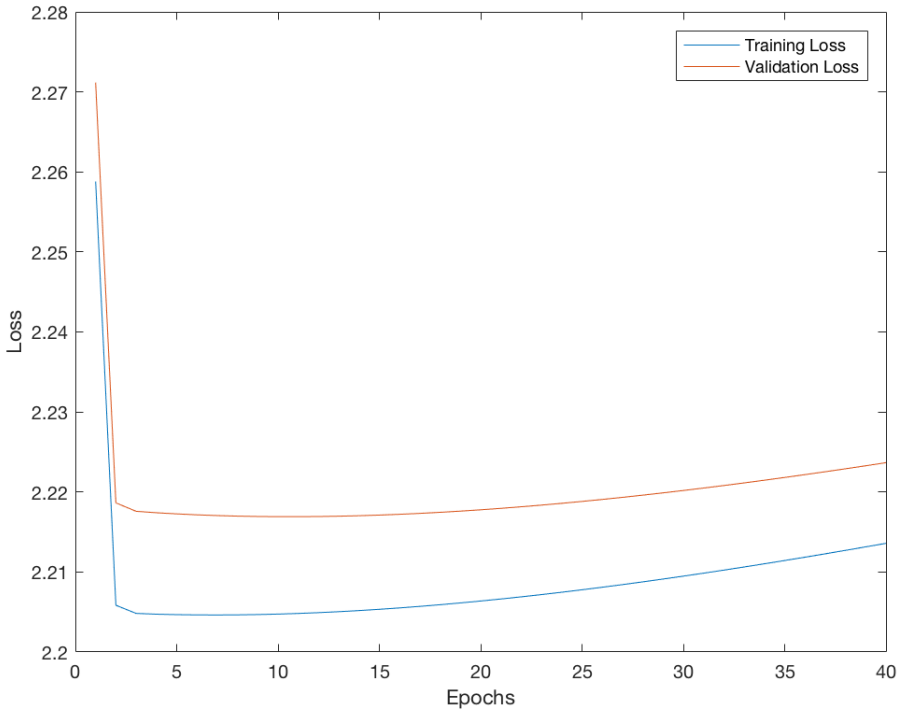

Figure 7:

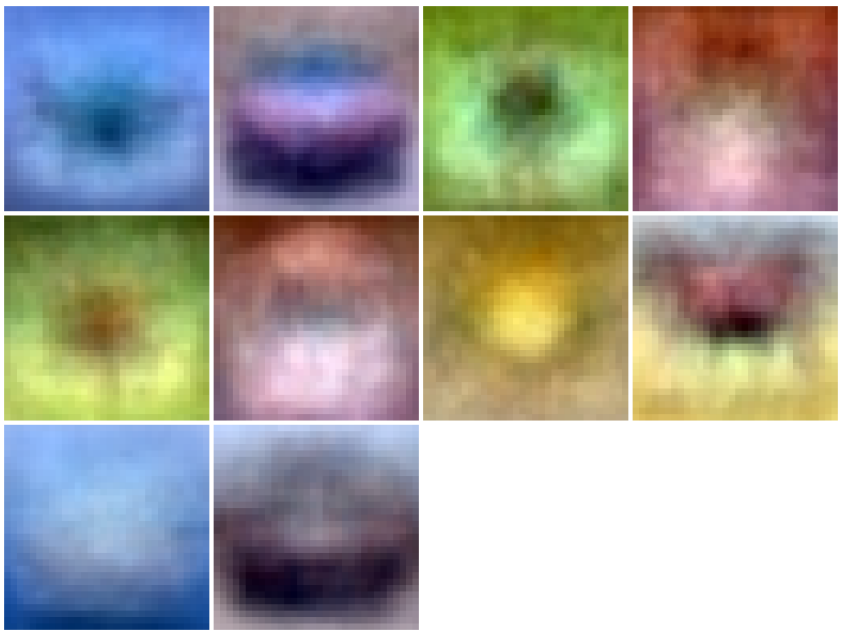

Figure 8:

Conclusion on the Regularization and Learning Rate

Regularization Term

From lecture 4 from [2] slide 13 we can get the idea on how our regularization term affects our model. The regularization, `lambda` term is used so that our data will not over-fit. However this is mostly for quite large models, while our is quite small, and thus it makes it worse. This is due to that we generalize 'too much', meaning that when it sees a red car, it maybe only classifies a car if it is red, which we do not want in our case. This phenomena can be visualized by comparing fig. 1 with fig. 7 where our accuracy becomes significantly lower. In fig. 5 we used a smaller regularization gain than fig. 7, and we already see that we have a much better improvement with accuracy, but still not as good as in fig. 1 where `lambda` is zero.

Learning Rate

The learning rate `eta` will work as an increase in how much we would like to 'leap ' or compensate for the direction of the gradient when we estimate our weight and bias. We want to minimize the loss as much as possible, meaning that we want to basically find the minimum value of the loss function just before it starts to increase again or has converged. When we apply the learning rate we tell our model to step in the direction of the gradient for every epoch we compute. A good example is to compare fig. 1 where we have a smooth decreasing curve as we want, but in fig. 3 we observe a quite 'noisy' loss function that jumps up and down, which is caused by the increase of a significantly higher learning rate. What we observe is that we 'compensate' too much in the gradient direction which leads to inconsistent steps, which explains the behaviour of the loss function fig. 3. There must exist a balance, if we lower the learning rate much lower than we already have we will converge too slowly and it would take to long to find the point of convergence.

References

[1] Assignment1 DD2424 Instructions paper:
https://www.kth.se/social/files/58d270a5f27654417f4c89c6/Assignment1.pdf

[2] Lecture notes by Josephine Sullivan on DD2424 KTH: https://www.kth.se/social/course/DD2424/subgroup/vt-2017-958/page/lectures-164/