Deep Learning in Data Science, DD2424

Short report on assignment 1

Single layer classification network for CIFAR-10

Simon Jarvers

April 5, 2023

Version 2: Standard deviation in the weights and bias initialization is now set to 0.01 instead of 1 as in the original report.

1 Main objectives and scope of the assignment

My goals in the assignment were:

- to implement a basic one-layer classification network,
- to use the gradient decent method to train the network,
- to train the network on the well-known CIFAR-10 data set.

To study the points of interest, the experiments and plots were created in Python using the widely known libraries, i. e. NumPy and Matplotlib. This report was written without the use of ChatGPT.

2 Classification with a single-layer classification network

After loading, normalizing and splitting the data into training, validation and test sets, I implemented the functions `eval()`, `cost()`, and `acc()` to compute the networks forward pass, cost, and accuracy respectively. As stated in the assignment for this first part the softmax function was used in the forward pass as well as the cross entropy loss function together with $\mathcal{L}_2$ regularization for the cost assessment.

Next, the gradients for the Gradient Decent Algorithm were computed based on the scheme in Lecture 3. Using the provided `ComputeGradsNum(h=0.00001)` function, exemplary testing on the first 10 training images with random initialized weights resulted in a maximal relative gradient difference of `6.9020e-06` and a maximal absolute gradient difference of `3.8413e-15`. With the correctness of the analytical gradients verified, we can run the four given parameter settings.

3 Results and discussion

To optimise the networks capabilities it is crucial to find suitable hyperparameters such as learning rate, regularization factor, number of epochs, and batch size. In Section 4 a more systematic approach of hyperparameter optimization in the form of grid search will be performed as well as other additional optimization methods such as learning rate decay are explored. For now we evaluate the networks performance based on the four given settings and one additional setting that I found to be interesting with corresponding test accuracies:

1. λ=0, n_epochs=40, n_batch=100, η=0.1 Final Test accuracy: 26.87%
2. λ=0, n_epochs=40, n_batch=100, η=0.001 Final Test accuracy: 38.89%
3. λ=0.1, n_epochs=40, n_batch=100, η=0.001 Final Test accuracy: 39.23%
4. λ=1, n_epochs=40, n_batch=100, η=0.001 Final Test accuracy: 37.46%
5. λ=0.1, n_epochs=40, n_batch=100, η=0.005 Final Test accuracy: 38.88%

The magnitude of the regularization factor λ seems to have a high effect on the weights' images as well as overall performance as seen in Figure 4. What I find even more interesting is the small increase of the learning rate η from 0.001 in the 3. configuration to 0.005 in the 5. configuration. These slightly different learning rates result in a significant higher test accuracy which however might also have been scored with the lower learning rate for an higher number of epochs considering the steep slope of the cost and accuracy plots in Figure 3.

4 Bonus assignment

4.1 Improve the performance of the network

To improve the networks performance I (1) used the remaining training data batches to increase the amount of training data, (2) implemented a learning rate decay after every n epochs (additional hyperparameter), and (3) performed a grid search to find optimal hyperparameters.

Using the additional training data and halving the learning rate every 10 epochs starting from $\eta = 0.1$ the performance increased as illustrated in Figure 6.

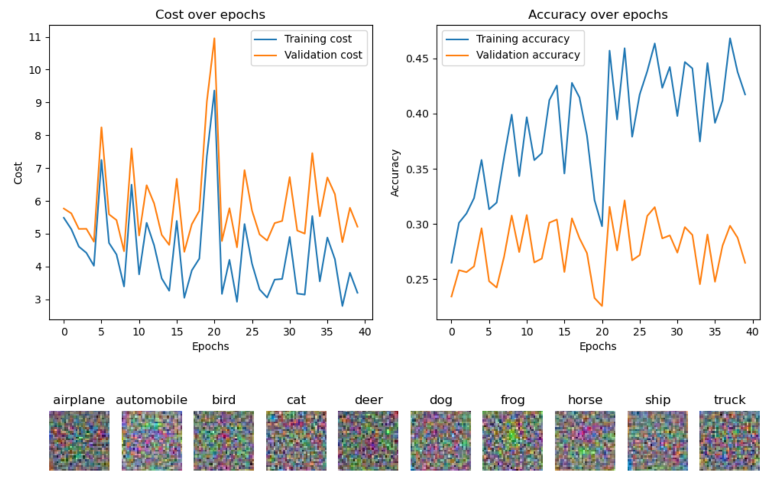

Figure 1: λ=0, η=0.1; Final Test accuracy: 26.87%

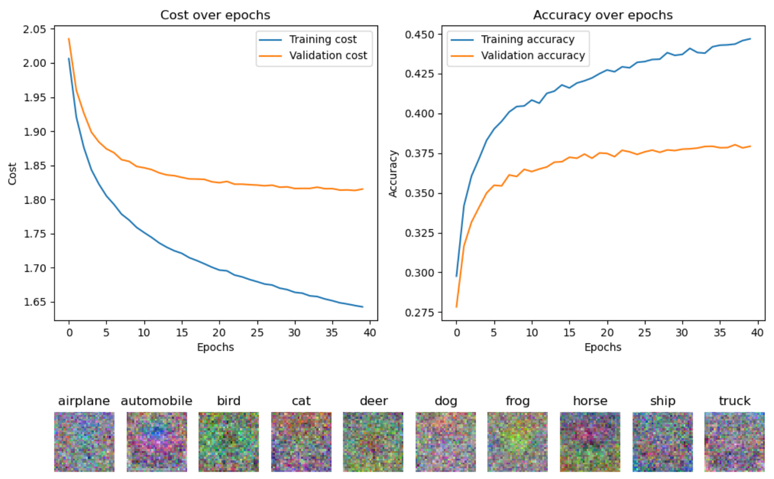

Figure 2: λ=0, η=0.001; Final Test accuracy: 38.89%

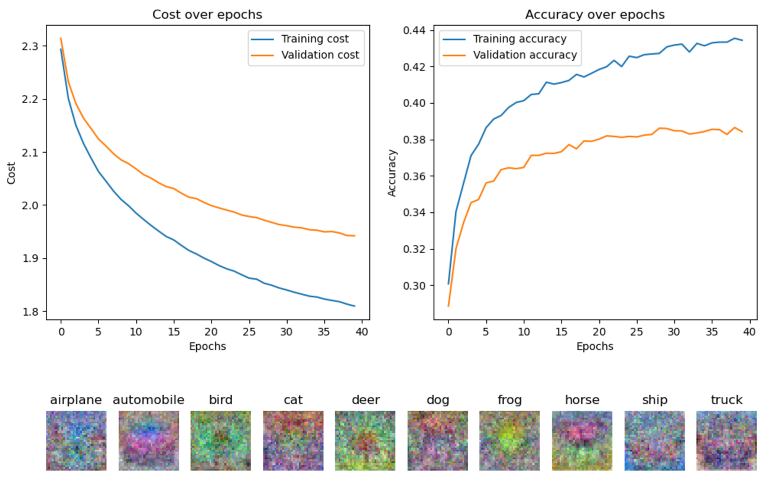

Figure 3: λ=0.1, η=0.001; Final Test accuracy: 39.23%

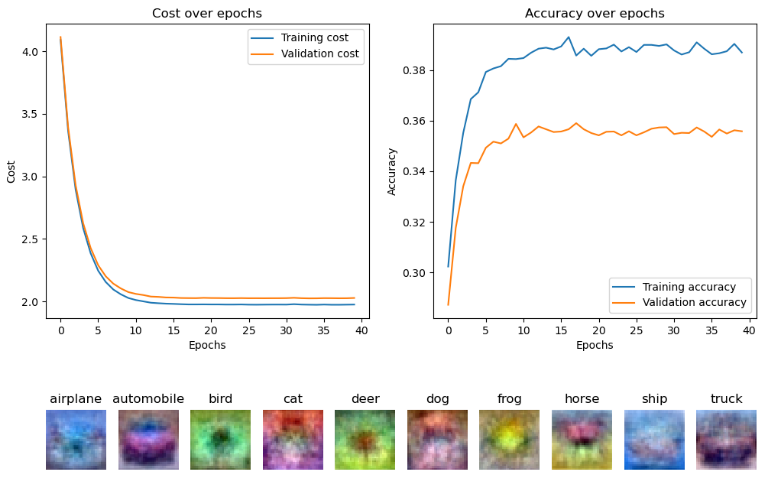

Figure 4: λ=1, η=0.001; Final Test accuracy: 37.46%

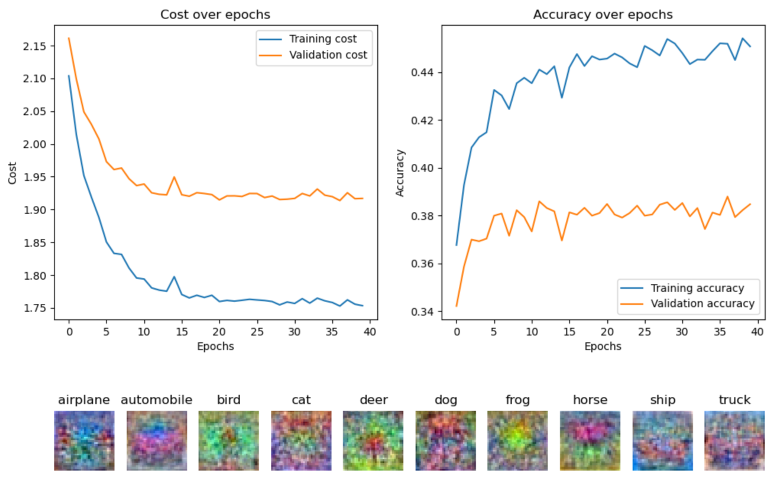

Figure 5: λ=0.1, η=0.005; Final Test accuracy: 38.88%

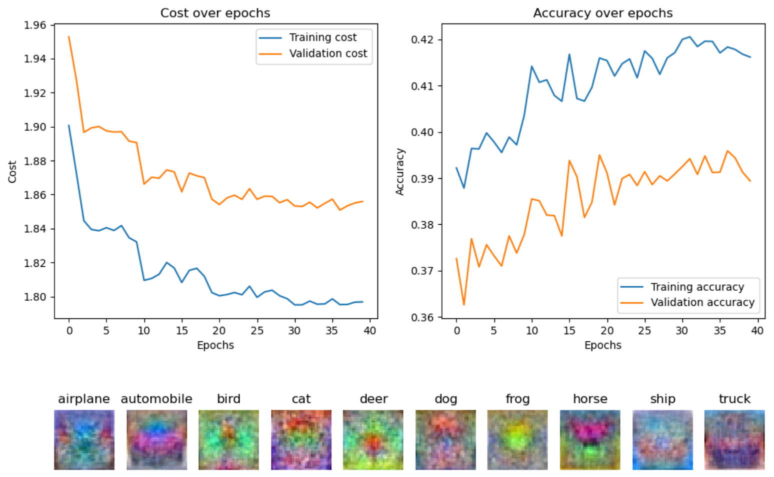

Figure 6: λ=0.1, η=0.01, halve every 10 epochs; Final Test accuracy: 40.16%

		$\lambda = 1$	$\lambda = 0.1$	$\lambda = 0.01$
n_batch = 50	$\eta = 0.1$	35.4%	38.8%	40.1%
	$\eta = 0.01$	35.7%	39.0%	38.8%
	$\eta = 0.001$	34.6%	36.6%	25.0%
n_batch = 100	$\eta = 0.1$	36.9%	39.4%	40.2%
	$\eta = 0.01$	36.2%	39.1%	30.7%
	$\eta = 0.001$	35.2%	30.1%	22.8%
n_batch = 500	$\eta = 0.1$	36.1%	39.2%	31.8%
	$\eta = 0.01$	35.5%	33.6%	24.0%
	$\eta = 0.001$	33.4%	18.0%	18.3%

Table 1: Final validation accuracies for different configuration of hyperparameters.

For the grid search I decided to try 3 different settings for 3 different hyperparameters resulting in a total of 27 configurations.

- n_batch $= \{50, 100, 500\}$
- $\lambda = \{1, 0.1, 0.01\}$
- $\eta = \{0.1, 0.01, 0.001\}$ where after {5, 10, 20} epochs η was halved

The final validation accuracies (as a measurement of performance) can be found in table 1. The highest validation accuracy of 40.2% with a similar final test accuracy of 40.4% was achieved by n_batch $= 100$, $\lambda = 0.01$ and $\eta = 0.1$ (halved every 5 epochs). This optimal networks performance can be seen in Figure 7.

4.2 Train network - multiple binary cross-entropy losses

In order to use the proposed multiple binary cross-entropy loss function to interpret the networks outputs as probabilities I had to adjust the models the forward pass, cost computation and theoretically the gradient computation. However, with the given multiple bce loss definition

$$l_{multiple\_bce}(x, y,) = -\frac{1}{K}\sum_{k=1}^{K}[(1 - y_k)\log(1 - p_k) + y_k\log(p_k)] \tag{1}$$

the gradient computes to

$$\frac{\partial l_{bce}}{\partial w} = \frac{1}{K}\mathbf{X}^T(p_k - y_k) \tag{2}$$

which is identically to the cross-entropy gradient except for the $\frac{1}{K}$ term. As $K = 10$ is a constant this factor simply scales the learning rate η and can therefore be disregarded without loss of generality.

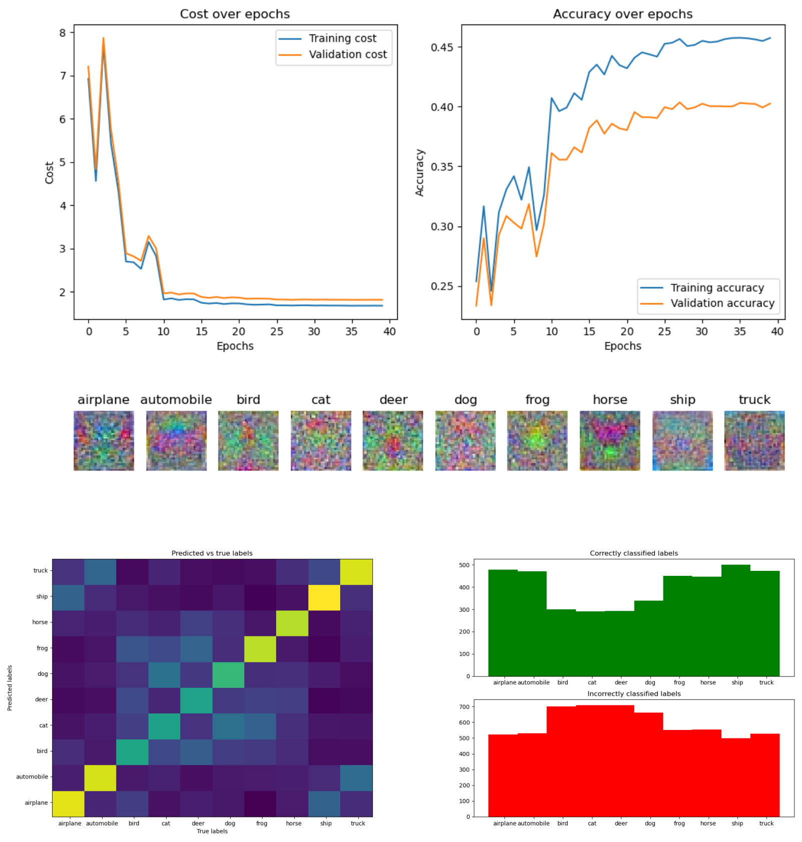

Figure 7: λ=0.01, η=0.1, halve every 5 epochs; Final Test accuracy: 40.41%

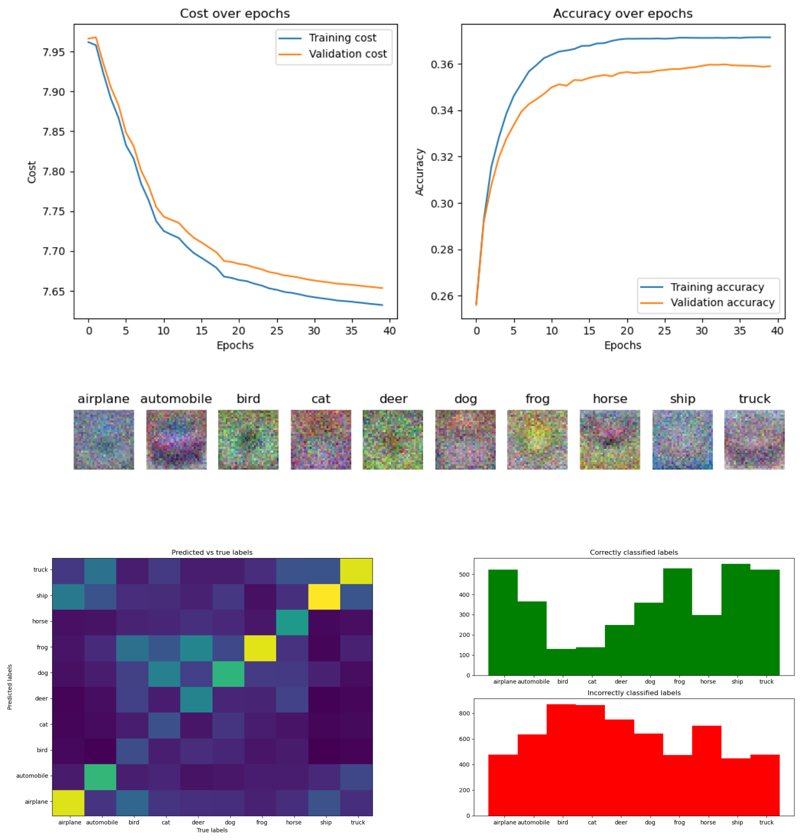

Figure 8: λ=0.1, η=0.001, halve every 10 epochs; Final Test accuracy: 36.65%

Highest accuracy was achieved by using $\lambda = 0.01$, 40 epochs, batch size $= 100$ and $\eta = 0.01$ which is halved every 10 epochs. The results are presented in Figure 8.