Assignment 1: One layer network - Bonus

DD2424 - Deep Learning
Svenja Räther

March 2020

1 Introduction

This assignment 1 in the course DD2424 Deep Learning in Data Science trains and tests a one layer network with multiple outputs to classify images from the CIFAR-10 dataset. Bonus 1 requires to optimize the performance of the network. The tricks tested for this task are: shuffling the order of the training examples in each epoch, using all available data for training, training for a longer time, introducing a decay factor to the learning rate, and applying Xavier initialization. (most of the same tricks you can be used for more complicated networks and they would probably have more of an effect for those networks). Bonus 2 replaces the cross-entropy loss by applying SVM multi-cross loss.

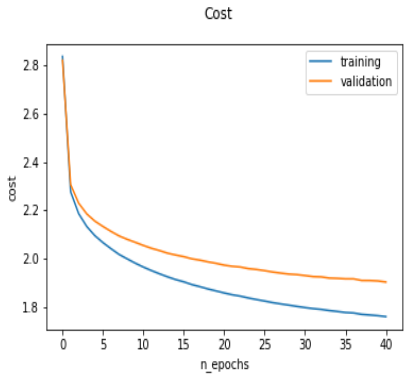
2 Bonus 1: Optimize the performance of the network

Different setups were executed to test the implemented improvements and compared to setup 3 in the previous report to measure their individual impact. The tested parameters for each setup and the results are displayed in the corresponding subsections.

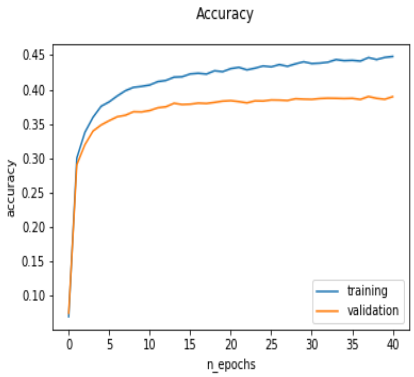
- Setup 5: lambda=.1, epochs=40, batch=100, eta=.001, permute=True
- Setup 6: lambda=.1, epochs=40, batch=100, eta=.001, permute=False, training size=49000
- Setup 7: lambda=.1, epochs=100, batch=100, eta=.001, permute=False
- Setup 8: lambda=.1, epochs=40, batch=100, eta=.03, permute=False, momentum =.9, update_round=1
- Setup 9: lambda=.1, epochs=40, batch=100, eta=.001, permute=False, xavier=True

2.1 Setup 5:
1g) Shuffle the order of your training examples at the beginning of every epoch.

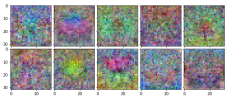
Backgroud: Shuffling is done to ensure that the training batches change their samples in each epoch. By this, we avoid that our model may use the order of the fed training data as a way to reduce the training error (overfit). Furthermore, it enhances to come out of a local minimum and supports convergence.
Overservation: Shuffled graphs appear less smooth. It lead to a very slight increase in accuracy on Test with *39.5* for shuffled and *39.27* for non-shuffled (setup 3). The slightly worse values on Train might be due to reduced overfitting.



(a) Cost for setup 5



(b) Accuracy for setup 5



(c) Weigths for setup 5

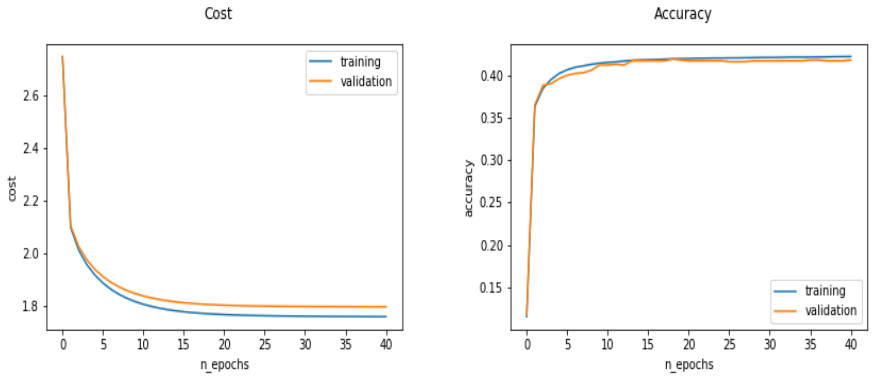
Figure 1: Plots for setup 5

Measure	Training	Validation	Testing
Cost	1.7604	1.9033	-
Accuracy	0.448	0.3895	0.395

Table 1: Results for setup 5

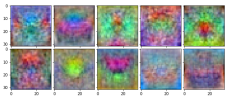
2.2 Setup 6:
1a) Use all the available training data for training.

Backgroud: More (good quality) training data adds information and should improve the fit.
Overservation: Using more data leads to an improve in accuracy on Validation with *41.8*. It is further visible that curves and values for Train and Validation data appear closer together. This seems to result in the largest effect among all tested improvements.



(a) Cost for setup 6

(b) Accuracy for setup 6



(c) Weigths for setup 6

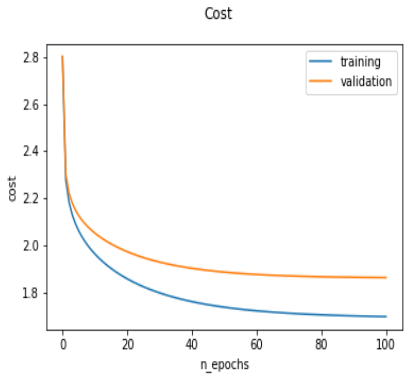
Figure 2: Plots for setup 6

Measure	Training	Validation	Testing
Cost	1.7616	1.7989	-
Accuracy	0.4221	0.418	-

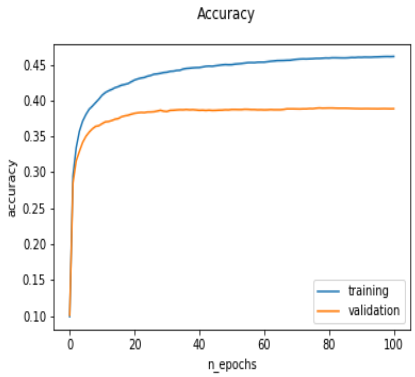
Table 2: Results for setup 6

2.3 Setup 7: 1b) Train longer and keep a record of the best model.

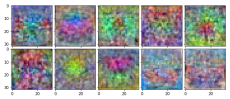
Backgroud: It takes a while for the model to converge and an improvement over time can be seen in the graphs. However, at some point the training starts to overfit on the data. Therefore, a record of the parameters performing best on Validation should be held.
Overservation: The example shows that the best result was achieved in *epoch 76 of 100* resulting in a Test accuracy of *39.48*. This is slightly better than the one from the last epoch which is *39.36*. Different parameters might have been better visualize the importance of keeping a record.



(a) Cost for setup 7



(b) Accuracy for setup 7



(c) Weigths for setup 7

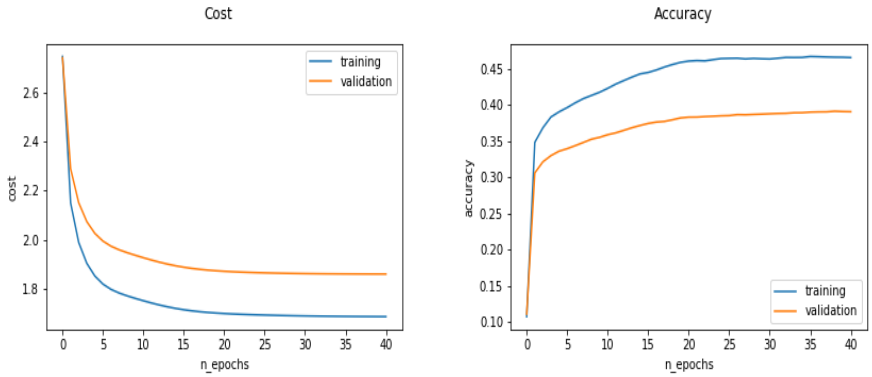
Figure 3: Plots for setup 7

Measure	Training	Validation	Testing	Test on best
Cost	1.7065	1.8672	-	-
Accuracy	0.4584 (76)	0.3895 (76)	0.3936	0.3948 (76)

Table 3: Results for setup 7

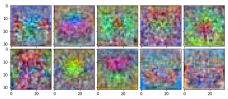
2.4 Setup 8:
1d) Play around with decaying the learning rate by a factor .9 after each epoch.

Backgroud: The learning rate decides the step that is made along the gradient. A large eta value takes large steps while a smaller value takes small steps. If the learning rate is kept high we might not be able to converge into a minimum (setup 2). If it is too small we might not learn fast. We decrease the learning rate to allow the network to take smaller steps in higher epochs.
Overservation: A good combination was found using eta = .03 and decay of .9, updating in every round. The Test accuracy reaches up to *39.82*.



(a) Cost for setup 8

(b) Accuracy for setup 8



(c) Weigths for setup 9

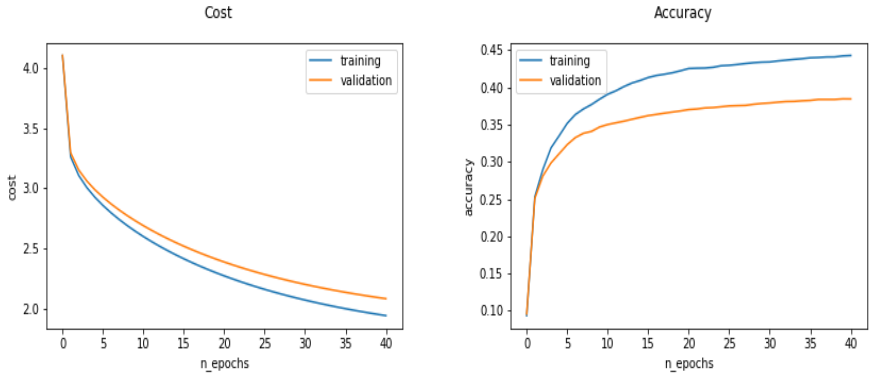
Figure 4: Plots for setup 9

Measure	Training	Validation	Test
Cost	1.6875	1.8605	-
Accuracy	0.4656	0.3907	0.3982

Table 4: Results for setup 9

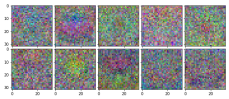
2.5 Setup 9:
1e) Implement Xavier initialization and comment if it stabilizes training.

Backgroud: Xavier initialization is often used so that the mean and spread of activations at each layer remain approx. constant.
Overservation: Since we only have a one layer network no improvement is detected when applying the Xavier initialization over the initiation used in setup 3. The results are slightly worse in this setup.



(a) Cost for setup 9

(b) Accuracy for setup 9



(c) Weigths for setup 9

Figure 5: Plots for setup 9

Measure	Training	Validation	Test
Cost	1.9403	2.0811	-
Accuracy	0.4429	0.3845	0.3848

Table 5: Results for setup 8

3 Bonus 2: Train network by minimizing the SVM multi-class loss

3.1 Implementation

Bonus task 2 required to implement the SVM multi-cross loss instead of the cross-entropy loss. Therefore, I implemented / modified the following functions:

- *ComputeGradientsSVM_slow*: iterative computation of loss and gradients
- *ComputeGradientsSVM*: vectorized computation of gradients
- *ComputeCostSVM*: vectorized computation of loss
- *testGradientsSVM*: compares values of vectorized with iterative version
- *GenerateMiniBatchesSVM*: generates X,Y,y batches
- *MiniBatchGDSVM*: SVM version of the miniBatch training

There are two possible implementations for the SVM multi-class loss. One is the slower iterative approach realized in *ComputeGradientsSVM_slow*. Since this took a long time to run all tests, I decided to additionally implement the vectorized version of the code. This can be found in *ComputeGradientsSVM*. This is further used for the comparison with the cross-entropy loss. In order to make sure, the vectorized and iterative versions result in the same gradients, I modified the *testGradients_svm* to calculate the relative difference and checked that it is small. Additional modifications were made in *GenerateMiniBatches_svm* and *MiniBatchGD_svm* to adapt the existing version to the SVM specific parameters and functions.

3.2 Comparison of SVM loss to cross-entropy loss

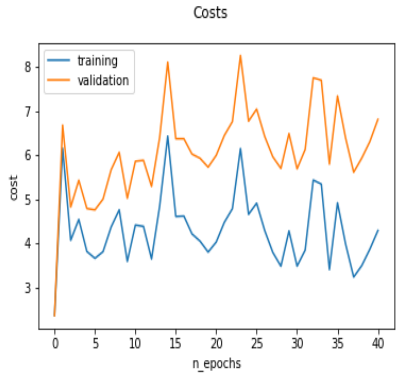
In order to examine the sensitivity towards parameters and accuracy of the SVM loss, I decided to test it on the parameter settings mentioned in task 1.

- Setup 1: lambda=0, n epochs=40, n batch=100, eta=.1
- Setup 2: lambda=0, n epochs=40, n batch=100, eta=.001
- Setup 3: lambda=.1, n epochs=40, n batch=100, eta=.001
- Setup 4: lambda=1, n epochs=40, n batch=100, eta=.001

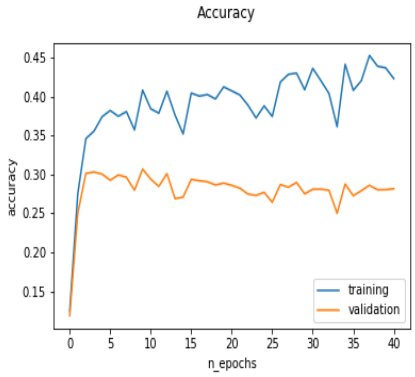
The comparison of those results show that the cross-entropy loss performs slightly better in terms of accuracy on this problem. Furthermore, the best weights for SVM were usually achieved after a low number of epochs while the cross-entropy improved until a higher number of epochs for those settings. By looking at the table and graphs below, it seems like the SVM version has a greater gap between validation and training values.

Setup	Measure	**Training**	**Validation**	**Testing**	**Test best W**
1	Cross-entropy cost	3.2670	5.7419	-	-
1	Cross-entropy accuracy	0.4639	0.2918	0.2953	0.3112 (3)
1	SVM cost	18.1374	44.5133	-	-
1	SVM accuracy	0.3856	0.2751	0.2779	0.2993 (4)
2	Cross-entropy cost	1.6115	1.7924	-	-
2	Cross-entropy accuracy	0.4537	0.385	0.3869	0.3873 (38)
2	SVM cost	2.9957	4.9300	-	-
2	SVM accuracy	0.4802	0.3484	0.3502	0.3632 (5)
3	Cross-entropy cost	1.7607	1.9025	-	-
3	Cross-entropy accuracy	0.4459	0.3864	0.3902	0.3902 (39)
3	SVM cost	3.3198	5.0773	-	-
3	SVM accuracy	0.4762	0.3498	0.3579	0.3597 (8)
4	Cross-entropy cost	1.8999	1.9573	-	-
4	Cross-entropy accuracy	0.3985	0.3635	0.3748	0.3741 (11)
4	SVM cost	4.0694	5.1330	-	-
4	SVM accuracy	0.4333	0.3612	0.3657	0.366 (34)

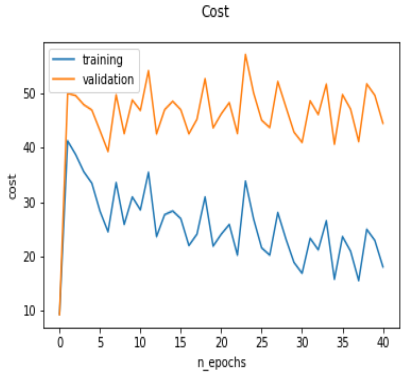
Table 6: Results for comparison of cross entropy and SVM loss



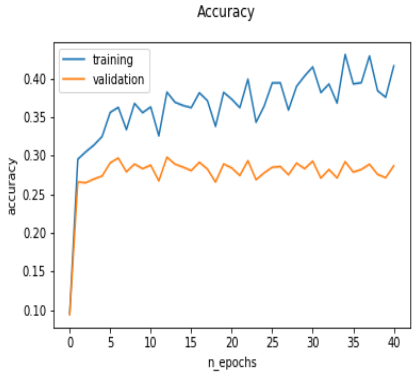
(a) Cost setup 1 Cross entropy



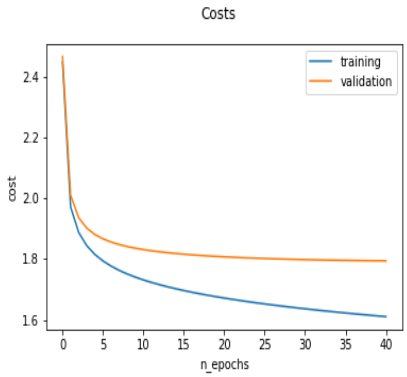
(b) Accuracy setup 1 Cross entropy



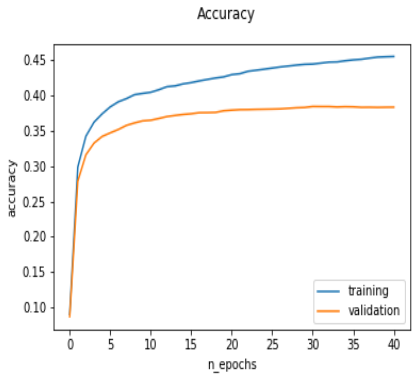
(c) Cost for setup 1 SVM



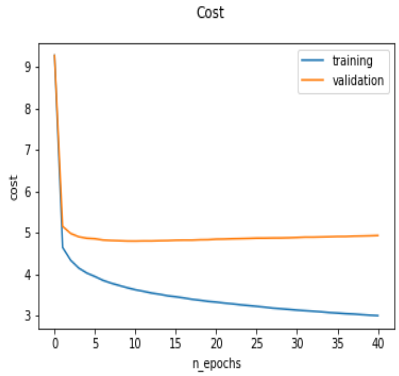
(d) Accuracy for setup 1 SVM



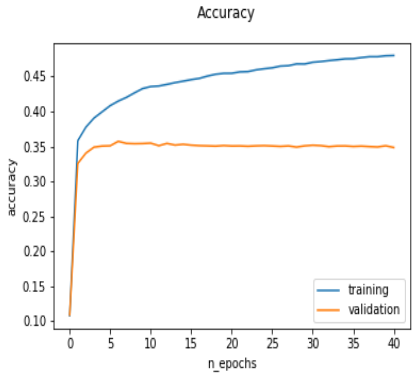
(e) Cost setup 2 Cross entropy



(f) Accuracy setup 2 Cross entropy

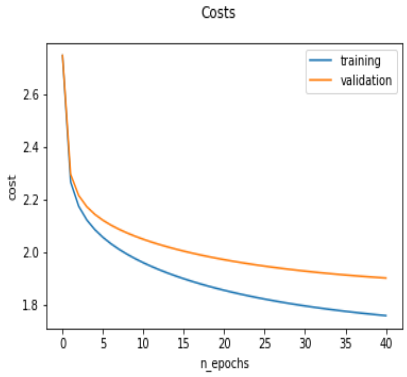


(g) Cost for setup 2 SVM

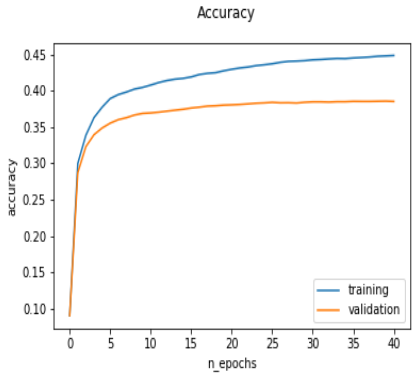


(h) Accuracy for setup 2 SVM

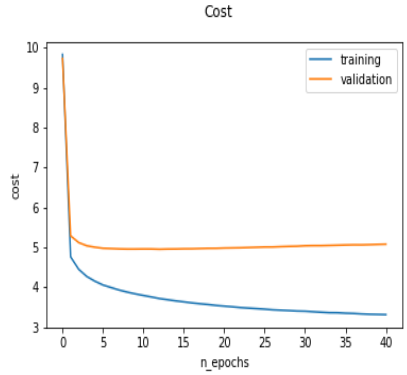
Figure 6: Plots for comparison of cross entropy and SVM loss



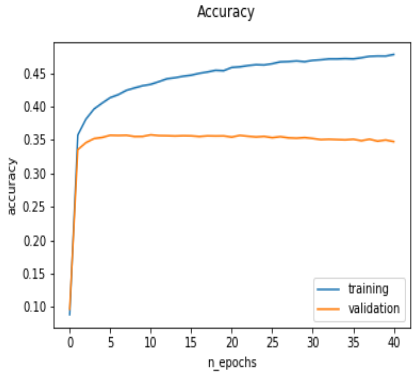
(a) Cost setup 3 Cross entropy



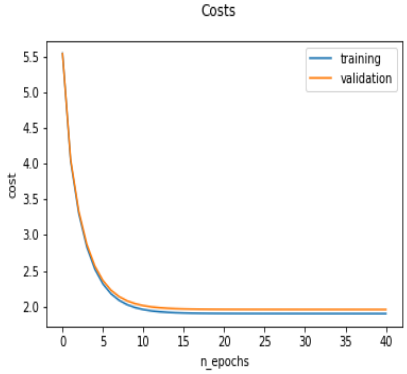
(b) Accuracy setup 3 Cross entropy



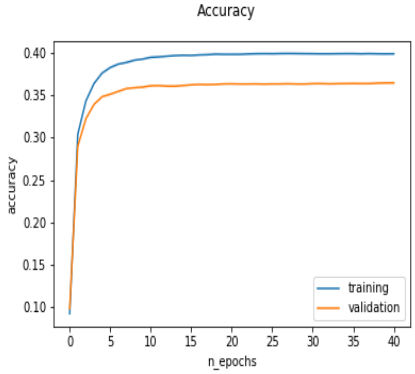
(c) Cost for setup 3 SVM



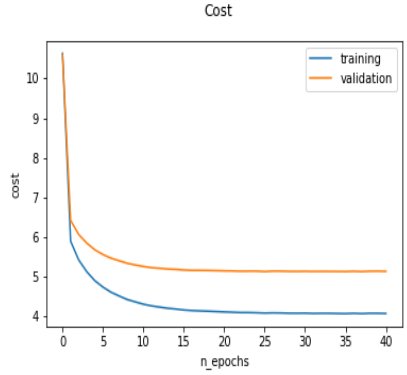
(d) Accuracy for setup 3 SVM



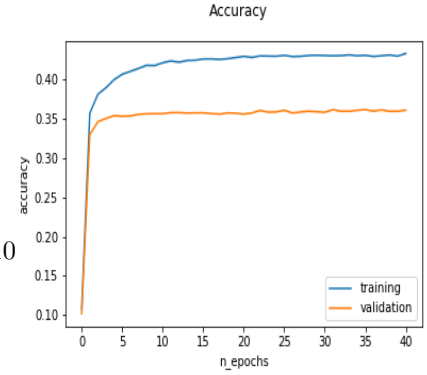
(e) Cost setup 4 Cross entropy



(f) Accuracy setup 4 Cross entropy



(g) Cost for setup 4 SVM



(h) Accuracy for setup 4 SVM

Figure 7: Plots for comparison of cross entropy and SVM loss