Henrik Holm henholm@kth.se 2020-03-30

DD2424 Deep Learning in Data Science

Assignment 1 - bonus version

1. Introduction

For the bonus version of the assignment, the aim was to optimize the performance of the single-layer neural network implemented in the base version of the assignment. In order to achieve this goal, at least three of a handful of performance-increasing methods had to be implemented. The "tricks" which I decided to implement were:

A) Using all available training data and decreasing the validation data set to 1000 images.

D) Implementing a decay factor for the learning rate.

E) Xavier initialization for the weights.

Moreover, a random seed of 12345 was used throughout the assignment.

Exercise 2.1 – implementing optimization schemes

A) Using all available training data

Results for running the same parameter combinations as in the base version of the assignment, but with the larger data set, are displayed below.

1. **lambda=0, n_epochs=40, n_batch=100, eta=0.1**

   which yielded the following accuracies:

- - training set accuracy: 0.3464
- - validation set accuracy: 0.3780
- - test set accuracy: 0.3131

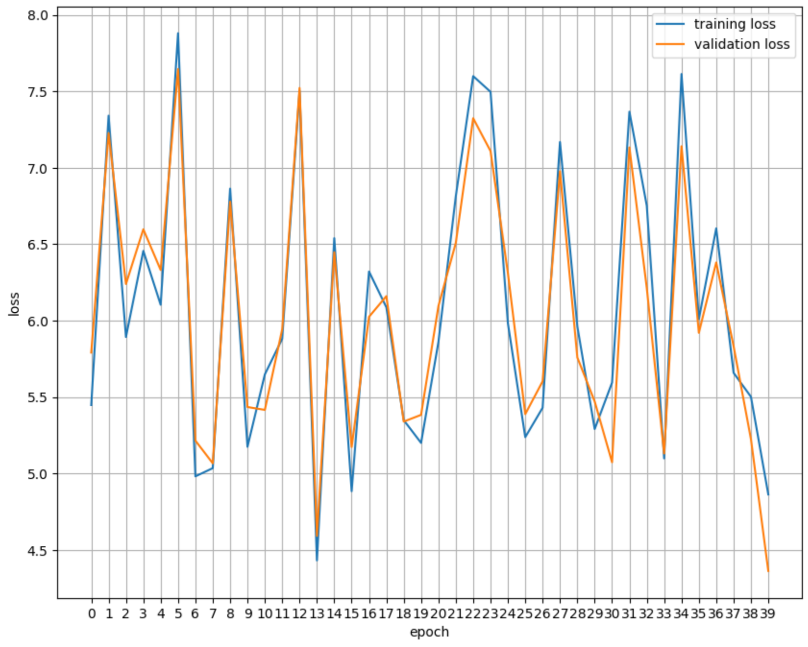

Figure 1: loss function for lambda=0, n_epochs=40, n_batch=100, eta=0.1

Here, one can see that increasing the size of the data set does not remedy the instability of the learning algorithm when a relatively large learning rate of eta=0.1 is used.

2. **lambda=0, n_epochs=40, n_batch=100, eta=0.001**

   which yielded the following accuracies:

- - training set accuracy: 0.4193
- - validation set accuracy: 0.4220
- - test set accuracy: 0.4097

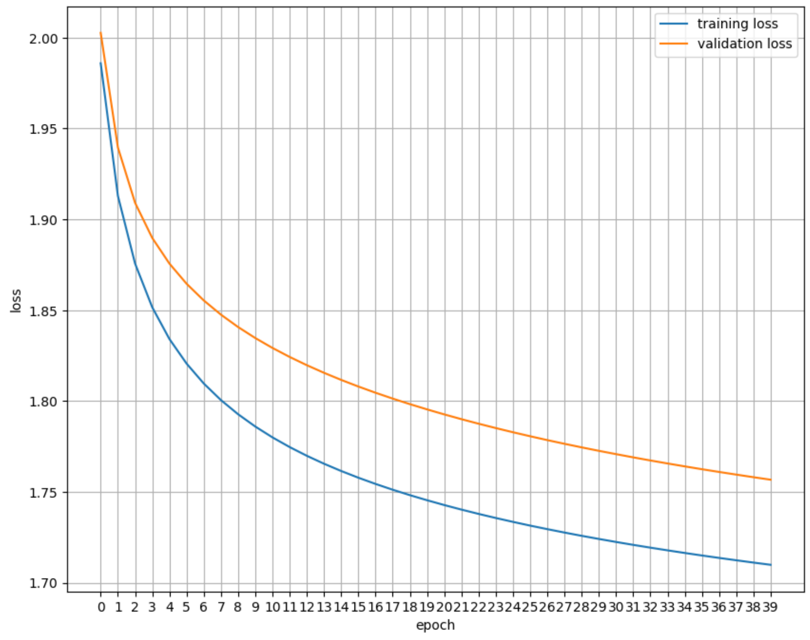

Figure 2: loss function for lambda=0, n_epochs=40, n_batch=100, eta=0.001

In this case, increasing the size of the data set used for training results in considerable improvements of the classification accuracy on the validation and test sets.

3. **lambda=0.1, n_epochs=40, n_batch=100, eta=0.001**

   which yielded the following accuracies:

- training set accuracy:	0.4157
- validation set accuracy:	0.4250
- test set accuracy:	0.4039

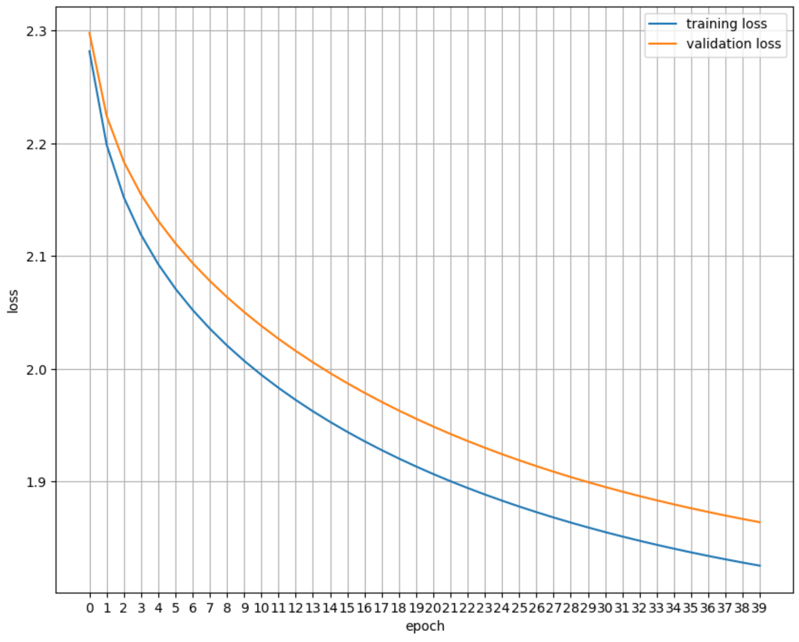

Figure 3: loss function for lambda=0.1, n_epochs=40, n_batch=100, eta=0.001

In this case, increasing the size of the data set used for training also results in improvements of the classification accuracy on the validation and test data.

4. **lambda=1, n_epochs=40, n_batch=100, eta=0.001**

   which yielded the following accuracies:

- training set accuracy:	0.3811
- validation set accuracy:	0.3810
- test set accuracy:	0.3785

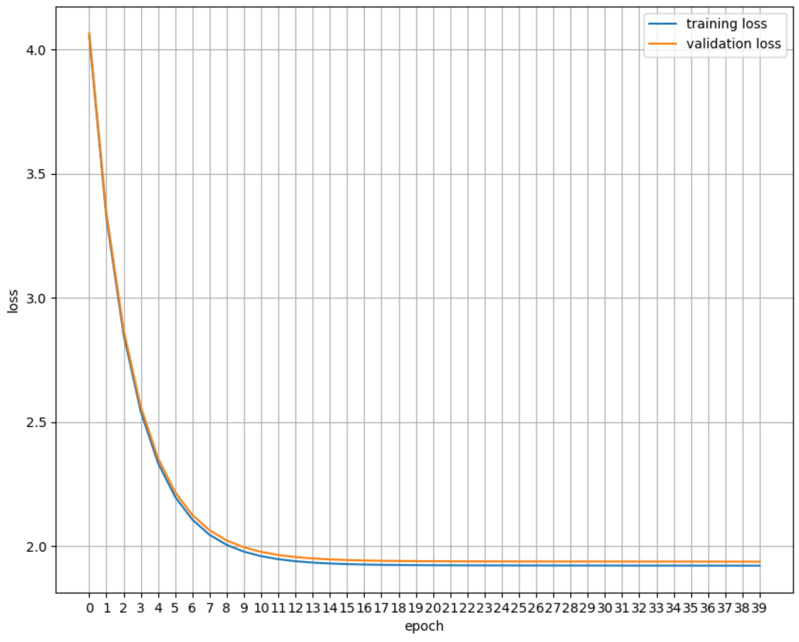

Figure 4: loss function for lambda=0.1, n_epochs=40, n_batch=100, eta=0.001

In this case, increasing the size of the data set used for training does not result in improvements of the classification accuracy on the validation and test data due to the same reasons as in the base version of the assignment: the too large regularization term renders the algorithm unable to learn.

D) Applying a decay factor on the learning rate

By implementing a decay factor of 0.9 in each epoch of the mini-batch gradient descent, improvements to the classification accuracy could be achieved. After a handful of sessions with different sets of parameters, the following combination proved to yield the highest increase in accuracy on the test set:

lambda=0, n_epochs=40, n_batch=100, eta=0.01, decay_factor=0.9

which yielded the following accuracies:

- training set accuracy:	0.4365
- validation set accuracy:	0.4410
- test set accuracy:	0.4107

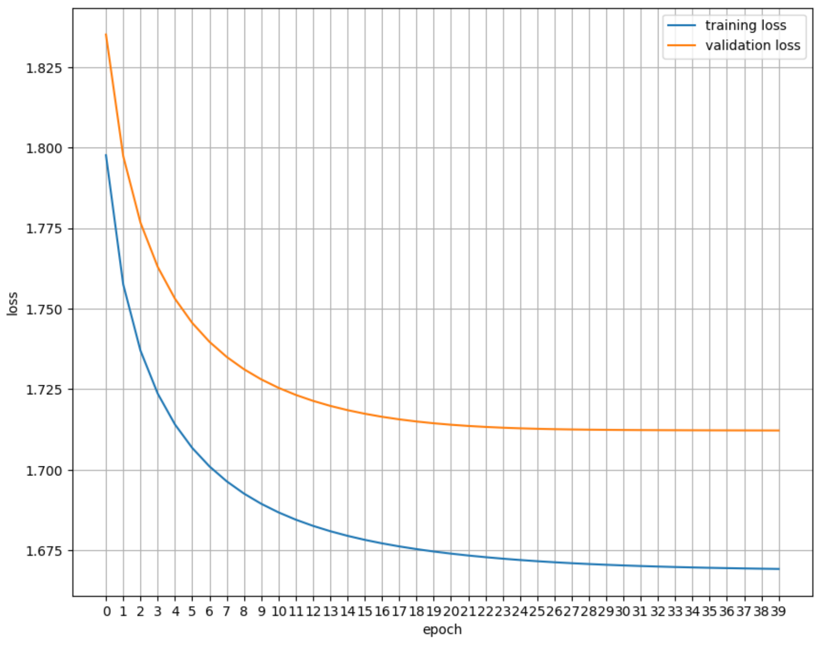

Figure 5: loss function for lambda=0.1, n_epochs=40, n_batch=100, eta=0.01, decay_factor=0.9

One can however easily note that the learning flattens out after about 20 epochs.

E) Xavier initialization for the weights

Implementing Xavier initialization for the weights appears to not yield any significant improvements, at least not with the current set-up. Here below are results of training the classifier with Xavier initialization of the weights and parameter combination:

lambda=0.1, n_epochs=40, n_batch=100, eta=0.01, decay_factor=0.9

which yielded the following accuracies:

- training set accuracy:	0.4234
- validation set accuracy:	0.4320
- test set accuracy:	0.4084

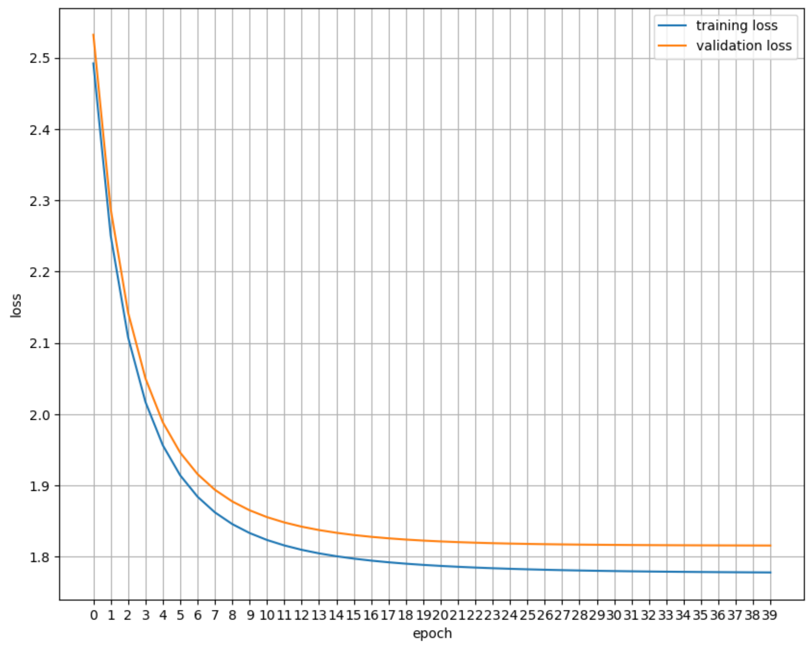

Figure 6: loss function for lambda=0.1, n_epochs=40, n_batch=100, eta=0.01, decay_factor=0.9 and xavier_init=True

As to whether Xavier initialization aids in stabilizing training, one can compare Figure 7 below with Figure 1 in the base report. In our case, Xavier initialization results in little improvements stability-wise.

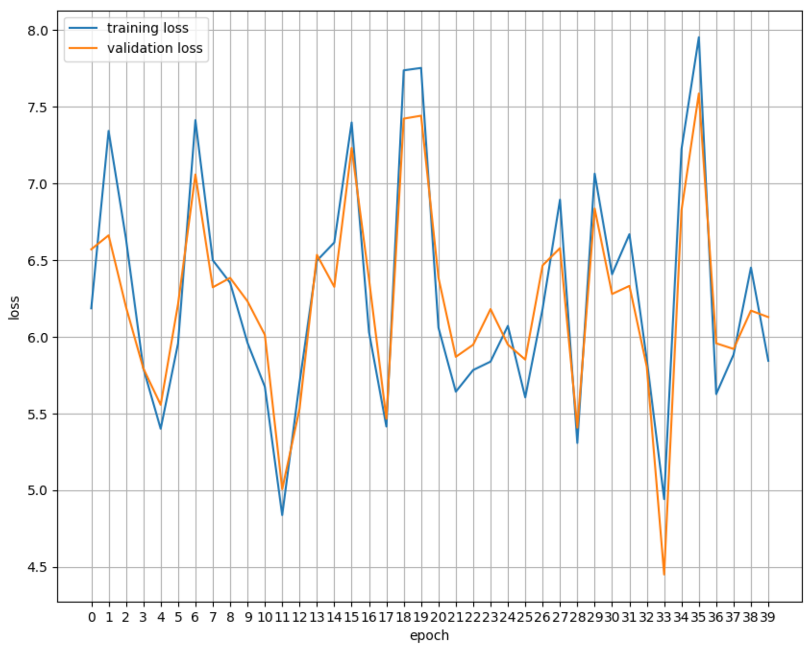

Figure 7: loss function for lambda=0, n_epochs=40, n_batch=100, eta=0.1, decay_factor=1 and xavier_init=True

In conclusion, by combining Xavier initialization with a learning rate decay factor of 0.9 for each epoch and the increased training data set, it is possible to achieve a higher accuracy, e.g. through the parameter setting displayed here below and in Figure 8, although it also flattens out after roughly 20 epochs. The following is the best classifier I was able to achieve.

lambda=0.1, n_epochs=40, n_batch=100, eta=0.1, decay_factor=0.92

which yielded the following accuracies:

- training set accuracy:	0.4241
- validation set accuracy:	0.4370
- test set accuracy:	0.4109

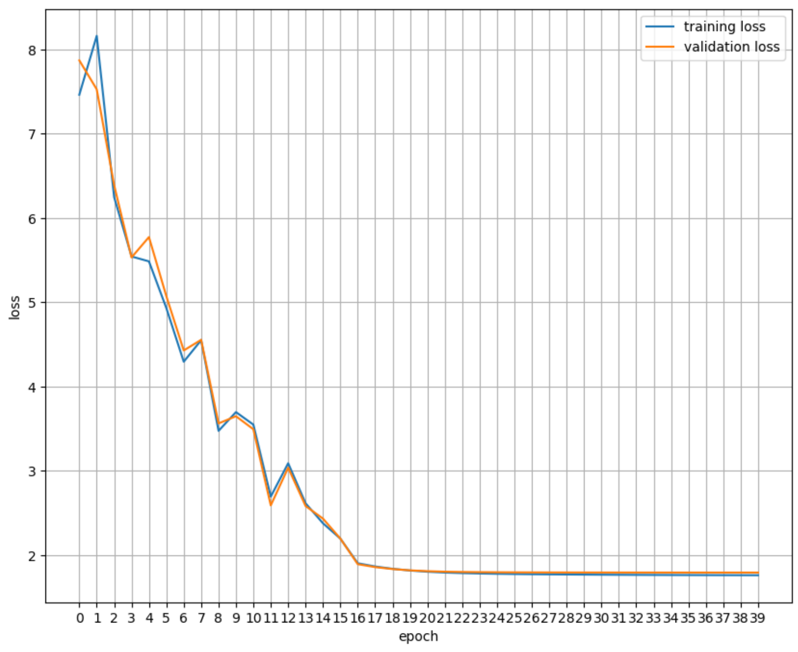

Figure 8: loss function for lambda=0.1, n_epochs=40, n_batch=100, eta=0.1, decay_factor=0.92 and xavier_init=True

From my results, I conclude that increasing the size of the training data set is the most efficient way of improving the accuracy of the classifier. Xavier initialization only brought mild improvements. Implementing a decay with regards to the learning rate was also an efficient way of improving the classifier, albeit not as good as increasing the data set size.

Exercise 2.2 – implementing multi-class SVM loss

For this part of the assignment, an SVM multi-class loss is to be used instead of the cross-entropy loss for training the network. Certain overflow-related issues arose in calculating the gradients. For testing the implemented classifier based on minimizing the SVM multi-class loss, the same tests were conducted as in Figures 1-4. The results are displayed below:

1. lambda=0, n_epochs=40, n_batch=100, eta=0.1

   which yielded the following accuracies:

- - training set accuracy: 0.3243
- - validation set accuracy: 0.3220
- - test set accuracy: 0.3269

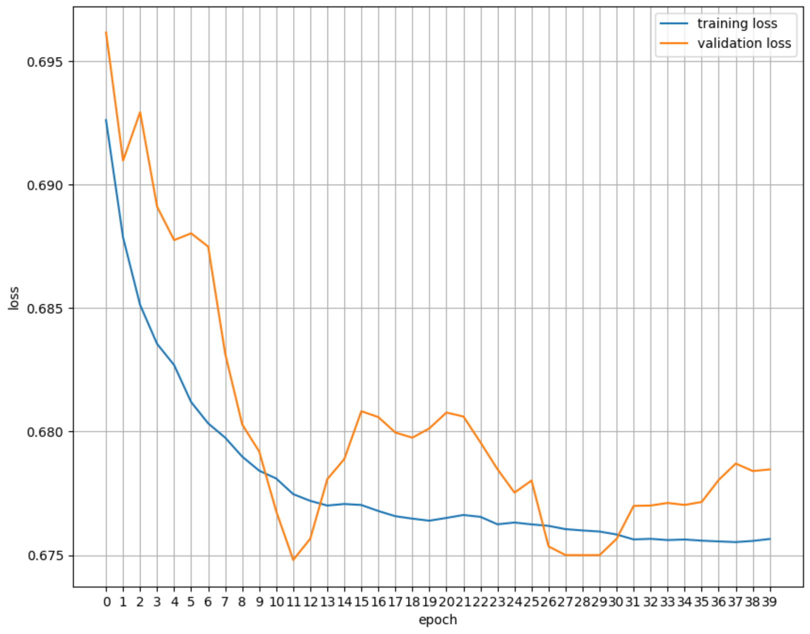

Figure 9: loss function for lambda=0, n_epochs=40, n_batch=100, eta=0.1

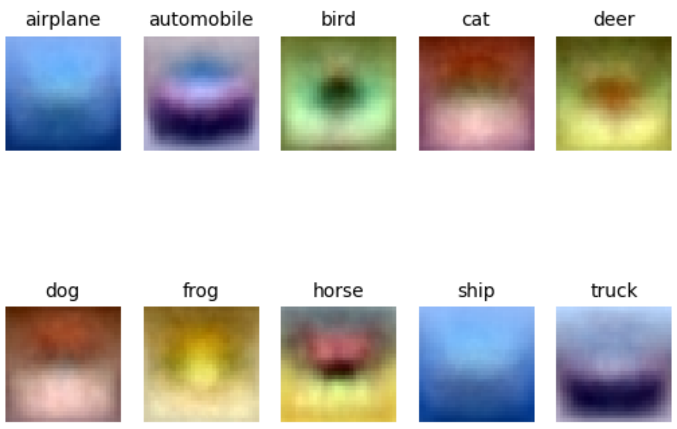

Figure 10: class templates for lambda=0, n_epochs=40, n_batch=100, eta=0.1

2. lambda=0, n_epochs=40, n_batch=100, eta=0.001

which yielded the following accuracies:

- - training set accuracy: 0.3031
- - validation set accuracy: 0.3040
- - test set accuracy: 0.3088

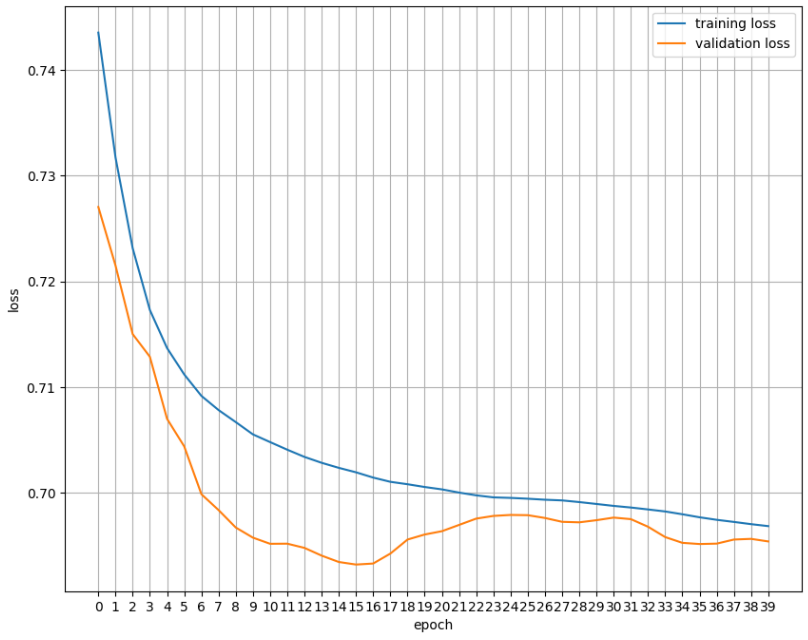

Figure 11: loss function for lambda=0, n_epochs=40, n_batch=100, eta=0.001

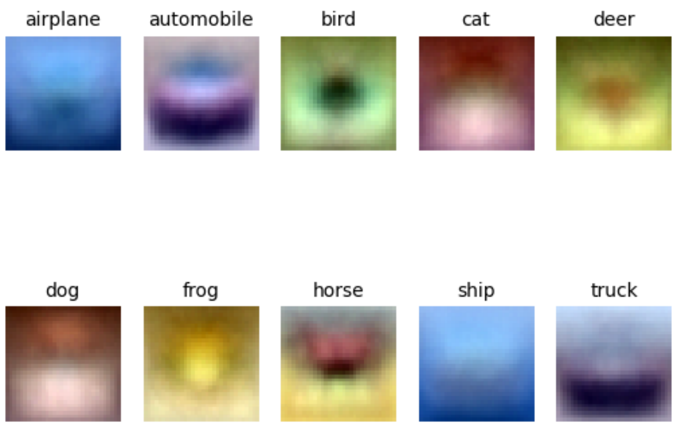

Figure 12: class templates for lambda=0, n_epochs=40, n_batch=100, eta=0.001

3. **lambda=0.1, n_epochs=40, n_batch=100, eta=0.001**

   which yielded the following accuracies:

- - training set accuracy: 0.3016
- - validation set accuracy: 0.3060
- - test set accuracy: 0.3090

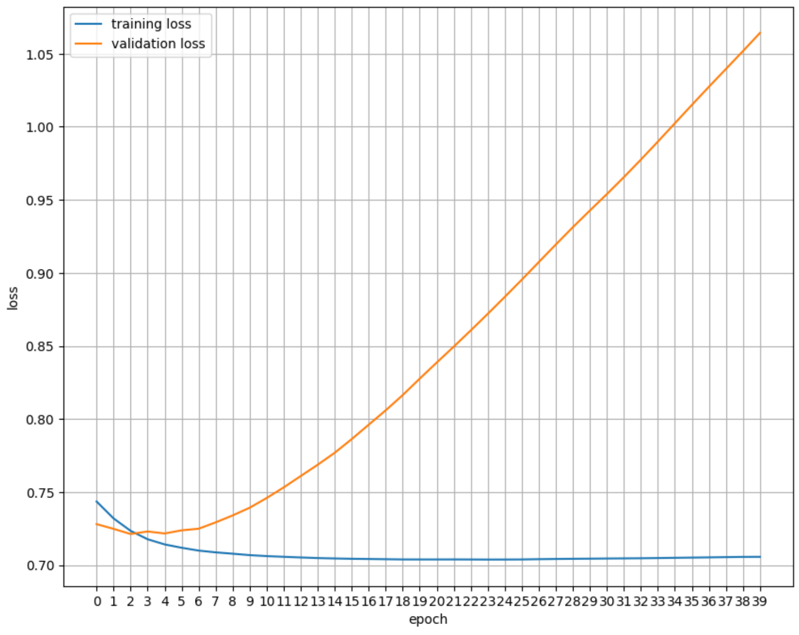

Figure 13: loss function for lambda=0.1, n_epochs=40, n_batch=100, eta=0.001

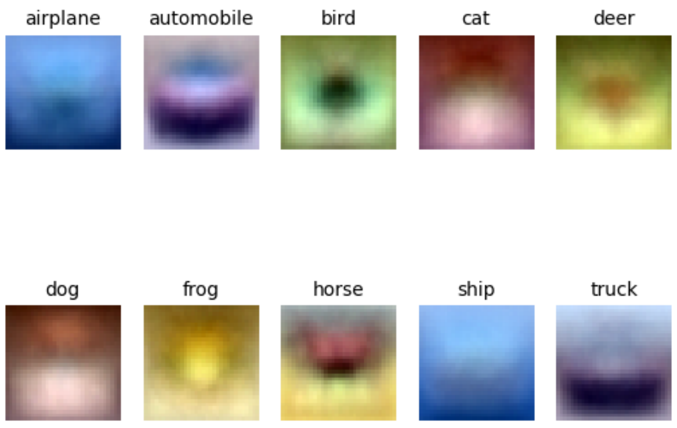

Figure 14: class templates for lambda=0.1, n_epochs=40, n_batch=100, eta=0.001

4. lambda=1, n_epochs=40, n_batch=100, eta=0.001

which yielded the following accuracies:

- - training set accuracy: 0.2929
- - validation set accuracy: 0.3060
- - test set accuracy: 0.3004

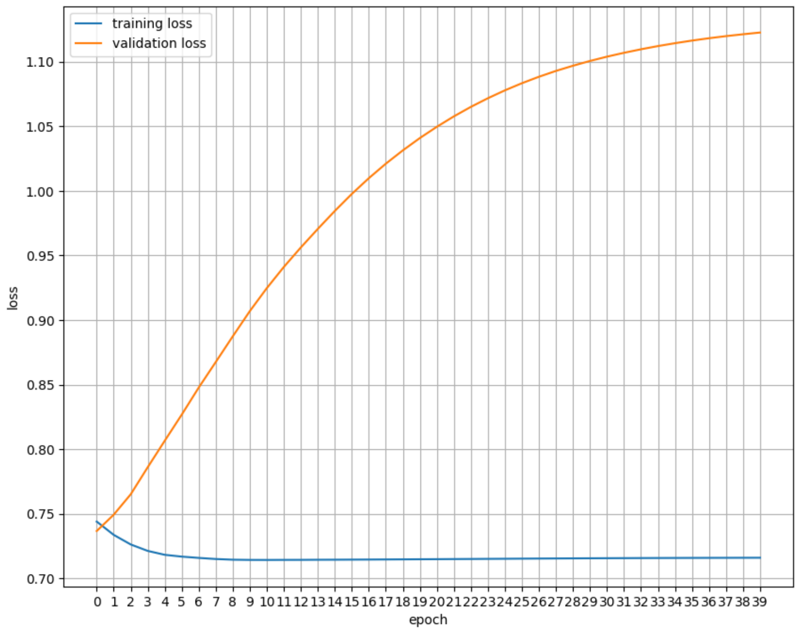

Figure 15: loss function for lambda=0.1, n_epochs=40, n_batch=100, eta=0.001

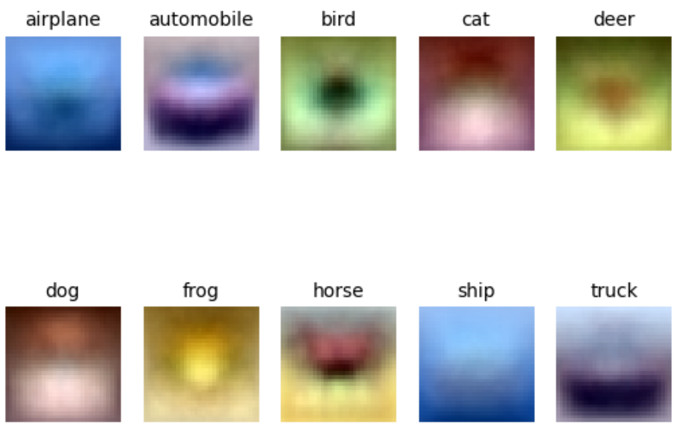

Figure 16: class templates for lambda=0.1, n_epochs=40, n_batch=100, eta=0.001

As can be seen from the SVM results and when comparing them to the cross-entropy results, I did something wrong in my coding. After countless hours of debugging, I am still unable to locate where the issue arises. The computation of the gradients is incorrect for the SVM_loss-method.

Assuming that I had implemented the function for computing the gradients correctly, I would have expected the SVM classifier to suffer from similar issues as the classifier based on the cross-entropy loss, namely that it too would be unstable with a too high learning rate and that it too would be unable to learn properly with a too high lambda.