Assignment 1 Bonus Point - Report

Pietro Alovisi

Report

In the nest subsection I am going to try some techniques to improve the performance of the network.

The code is basically the same as the one of the first par of the assignment,only some small changes are performed or other statistics during the training are saved. The functions relative to the SVM loss function have the suffix *SVM* in their name.

Full Dataset

I loaded 40000 images in the training data, and us 10000 for validation, and then I compared the results obtained on the test data with the results obtained in the first part of the assignment.

λ	eta	batches	epochs	Restricted Dataset	Full Dataset
0	0.1	100	40	$23.1\% \pm 4\%$	$26.8\% \pm 4.9\%$
0	0.01	100	40	$36.8\% \pm 1\%$	$38.5\% \pm 0.2\%$
0.1	0.01	100	40	$34.3\% \pm 0.6\%$	$34.17\% \pm 0.01\%$
1	0.01	100	40	$21.5\% \pm 0.5\%$	$24.55\% \pm 0.01\%$

From the loss diagram there is no evidence of overfitting, which means that the model is simple enaugh to not overfit with such a hgih dimensional data. By looking at the performance we can see that generally more data increases the performances, but more importantly it decreases the variance of the results we get, stabilizing the model.

Decaying Learning Rate

To try to boost the performance I also implemented the dacaying of the learning rate by a factor of 10 after each epoch. Then I computed several models by varying the initial learning rate and reported the performance as reported in Figure 1.

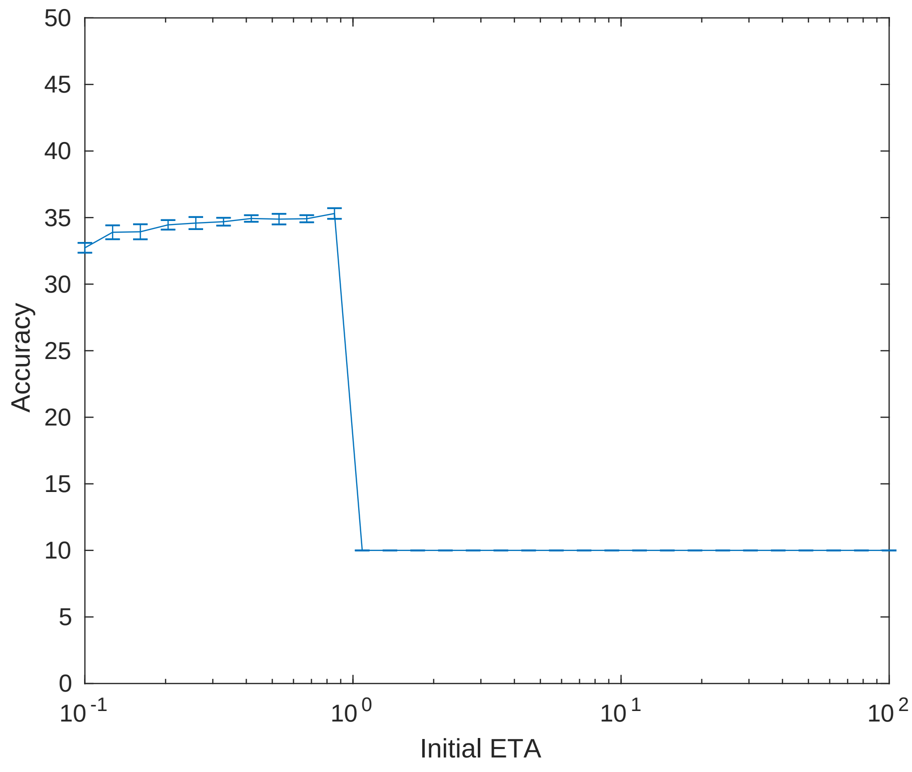

Figure 1: Performance and its standard deviation related to the starting learning rate.. Parameters used are $\lambda = 0$, size of a batch is 100, and 40 epochs

We can see that after a certain initialization of the learning rate the model performs completely randomly, this is probably because a high learning rate at the beginning moves the parametres too far away in first update, then the algorithm cannot fix it anymore.

With the parmeters I have set I cannot get a better result than the one I get without the decay.

Maybe a better way to decay the learning rate would be by choosing a higher decaying factor, in this way the decrease is more smooth.

Xavier Initialization

For this part I choose to pich the following parameters for the network: $\lambda = 0.1$, $eta = 0.01$, 40 epochs and batch size of 100, so that I can have in the gradient also the part relative to the regularization term, and I can also relate to the other results.

For this network the Xavier initialiation means that the parameters are:

$$w_{ij} = \mathcal{N}\left(0, \frac{1}{\sqrt{n_{in}}}\right) = \mathcal{N}\left(0, \frac{1}{\sqrt{3072}}\right)$$

Where $1/\sqrt{n_{in}}$ refers to the standard deviation and not the variance of the distribution.

I computed the mean and the standard deviation of the gradient for different multiples of that variance and plotted them in Figure 2 along with the performance of the network.

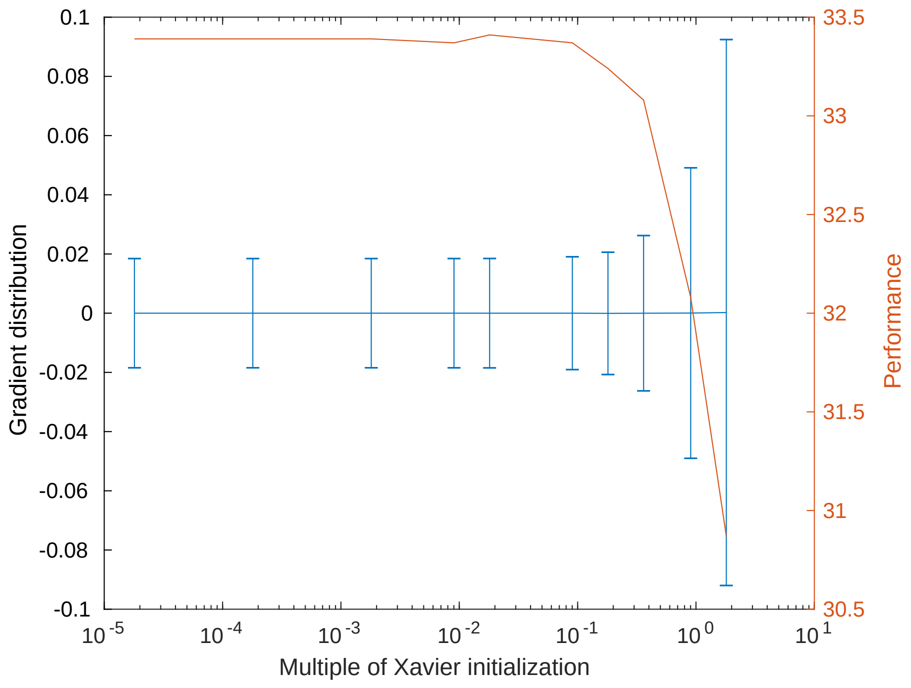

Figure 2: Distribution of the gradients and performance of the network as multiple of the Xavier initialization's standard deviation.

As we can see from the figure this initialization does not give us a big bump in performance, and also the effect of huge or small gradients is only relevant for multi-layer networks.

On the other hand the standard deviation of the performance on the test set is 0.033 which is smaller than the one obtained in the first part of the assignment with the same parameters, which let us conclude that the Xaver Initialization has a stabilizing effect on the network.

SVM Loss Function

To implement the network the only thing to change are the function that evaluate the network, that compute the cost and the computation of the gradients.

After having implemented the network, the first evaluation I have done is against the same parameters I used in the cross-entropy network, with only 10000 training samples. Here are the results:

λ	eta	batches	epochs	Cross-Entropy loss	SVM loss function
0	0.1	100	40	$23.1\% \pm 4\%$	$25.18\% \pm 3.0\%$
0	0.01	100	40	$36.8\% \pm 1\%$	$28.46\% \pm 2.8\%$
0.1	0.01	100	40	$34.3\% \pm 0.6\%$	$27.4\% \pm 1.87\%$
1	0.01	100	40	$21.5\% \pm 0.5\%$	$17.93\% \pm 1.8\%$

The network performs worse on almost every aspect, but as we can see from the loss plot of Figure 3 these parameters don't actually fit for this network, so to get the most of the network I tried other parametrs. By lowering the learning rate I can get smoother loss curves, as in Figure **??**.

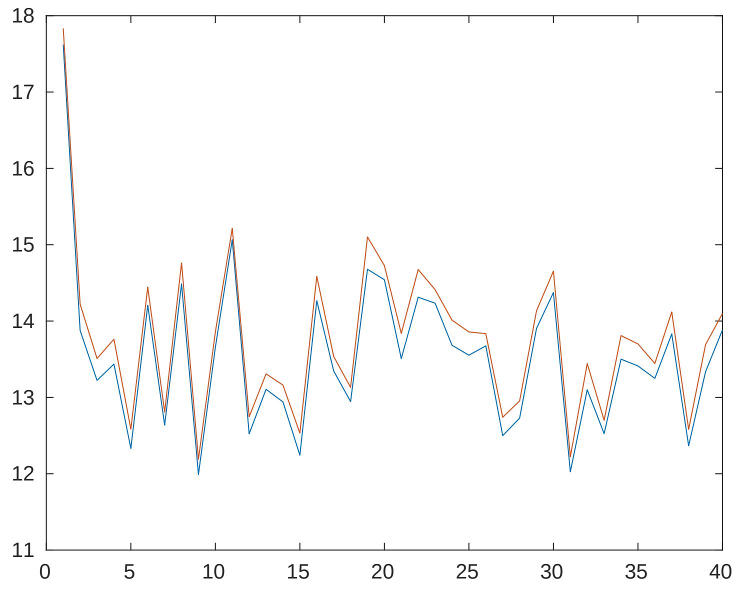

Figure 3: Loss function for the fourth set of parameter in the table.

λ	eta	batches	epochs	Initialization σ	SVM loss function
0.001	0.001	100	100	0.1	$34.0\% \pm 0.4\%$
0.001	0.001	100	200	0.1	$34.4\% \pm 0.26\%$
0.01	0.001	100	100	0.1	$34.34\% \pm 0.4\%$
0.01	0.01	100	100	0.5	$29.6\% \pm 2.3\%$

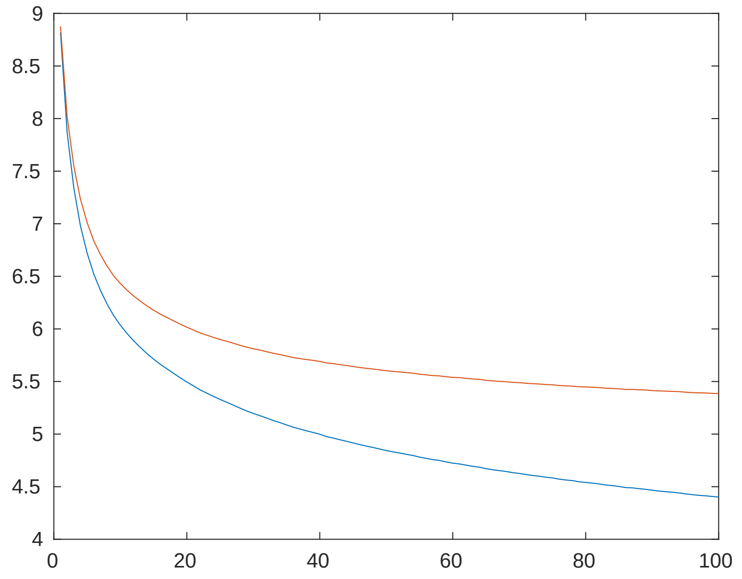

Figure 4: Loss function for SVM loss function with $eta = 0.001$.

As we can see there are no improvements in the final accuracy of the network if we choose a SVM loss function. This might be because the network is too simple to get an accuracy more than 38/40%.

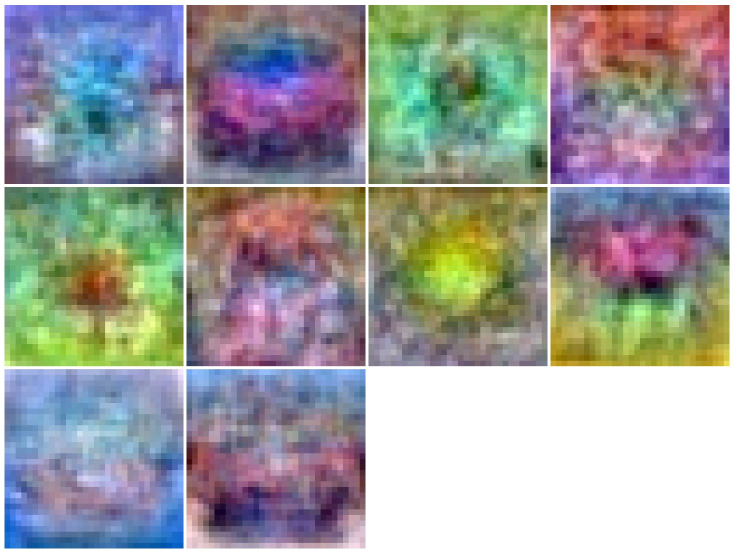

Figure 5: Learned representation by the network with the SVM loss function.