Deep Learning Project 1 - report

Maïk Guihard (284922), Alexandre Clivaz (287897), Bruno Da Costa (288525)

May 2021

Introduction

For this project, the objective was to create and compare multiple architectures of deep learning algorithms, in particular the impact of weight sharing and adding an auxiliary loss. Three different architectures were implemented and tested for this task: a simple network with convolutions and linear transformations, a network with weight sharing between the two images, and a combination of weight sharing with an auxilliary loss.
The three networks output a single real value. In order to count the number of errors, this value is rounded to the closest integer in {0,1}, and compared to the target value, which is also in {0,1}. The optimiser used is ADAM. All results cited below were obtained using 25 epochs and η equals to 0.001.

Architectures used

Normal

The initial model first applies three convolutional layers on the input data, one max pooling step and finally three linear classification layers, with ReLU in between each. This model is basic and doesn't use any other method to improve the result.

While satisfying the constraint of having less than 70'000 parameters (57'213 for this model), the model achieved those results: the mean training and testing error rates are respectively 0.03% and 18.98%, and their corresponding variance, estimated on 100 samples, are 0.02% and 2.68%.

Weight-sharing

The weight sharing structure consists of applying convolutions to both images separately, but using the same kernels and the same weights. Then, the resulting tensors are combined and run through a few linear layers to arrive at a single prediction. This structure ensures that both images are processed with the same network before being combined, meaning the network tries to differentiate at least a little each number, which was not the case in the first network.

The best model found uses 54'479 parameters. Over 100 repetitions, the mean training error rate is 1.83% and the mean testing error rate is 15.34%. Their variances are 2.67% and 2.43% respectively. Increasing the number of parameters, by adding channels, bigger or more linear layers, did not improve the results of this architecture above the one used.

Weight-Sharing and Auxiliary Loss

This last architecture, slightly more complex than the precedent, uses a combination of weight-sharing and auxiliary loss. The first part is similar to the weight sharing model, but instead of applying only one set of linear layers, another one is implemented, which has for objective to recognize the initial number on the images. This loss is computed as the cross entropy between the estimated value in range [0...9] of the picture and the real one, and is then multiplied by a ratio < 1. Several variations of layers were tried in order to get the best result. The addition of the two losses is used to calculate the gradient and improve the weights of the network.

The final model uses 69865 parameters. Over 100 samples, the observed results are a mean error of 1.66% on the train set, 13.75% on the test one, and respectively 2.26% and 2.95% for the variances.

Discussion

The results are summarized in Table. 1 and Fig. 1. Even if the mean error during training is the smallest for the normal architecture, a gain is observed, as expected, in the mean error during testing.

Model	Normal	Weight-Sharing	W-S + Auxiliary-loss
Train Error mean	0.03 %	1.83 %	1.66 %
Train Error var	0.02 %	2.67 %	2.26 %
Test Error mean	18.98 %	15.34 %	13.75 %
Test Error var	2.68 %	2.43 %	2.95 %

Table 1: Summary of the results

The biggest performance difference is between the "normal" network and the weight-sharing network. With a similar number of parameters, the latter performs better. This is because instead of passing a big chuck of data (the two images together) to the network, the data, which is half the size (one image), passed through a part of the network separately. Therefore the amount of parameters used in the convolutions is much smaller than with the big chunk of data, which is twice as big. So more parameters are free to be used elsewhere in the network.
Also, as we are dealing with images of numbers, having some processing in parallel on two images probably has a better chance of training the model to get closer to identifying the numbers (if only very roughly) instead of predicting a value from a big chunk of data. This is also useful because if we invert the two numbers (i.e. 1&3 instead of 3&1), a network that has some sense of what each number is will perform better than a simpler one, which would have to learn the whole structure for both a 1&3 and 3&1 images.

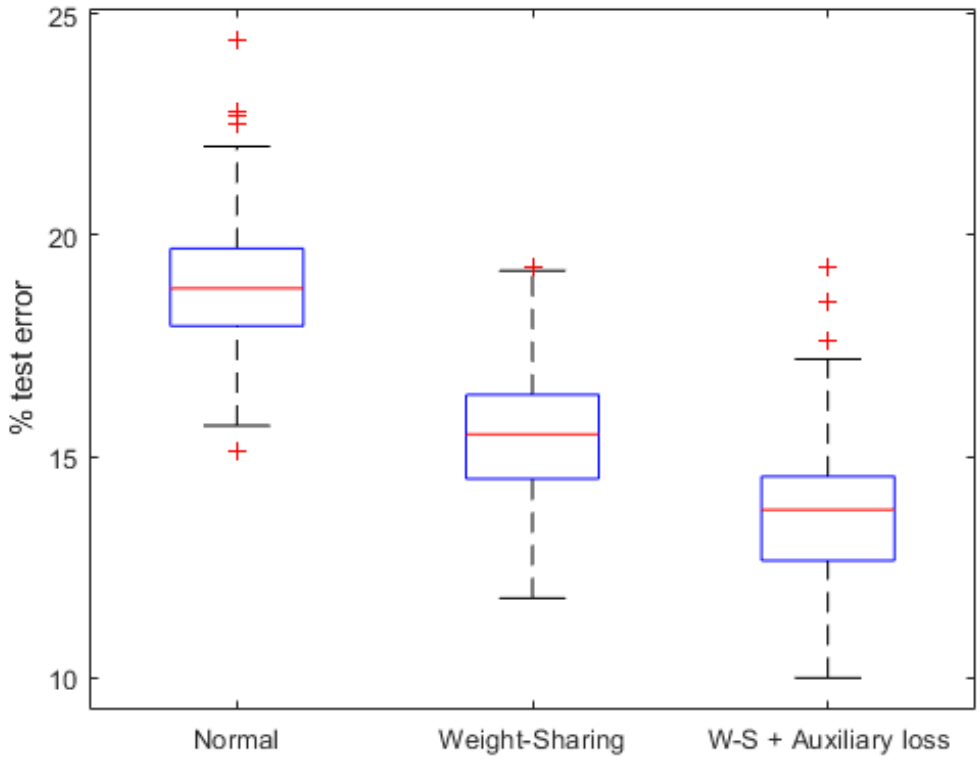

Figure 1: Boxplots of the testing error over 100 tests, for each architecture

Adding feedback to the model through the auxiliary loss with the classes further increases the performance. Again, it gives more incentive to the network to evolve so that it recognizes the numbers, and it is more efficient. However, the loss of the model is a weighted sum between this auxiliary loss and the MSE loss. Theoretically, a network that perfectly identifies the numbers would be perfect. However, the goal here is to output a single value (0 or 1), so all the weight can't be given to the classes during training. Obviously, the output of the network used to classify is given by the prediction, which is the input of the MSE loss, therefore this part needs to be consequent. So there is an equilibrium between aiding the network to recognize numbers, and completely disregarding the actual output of it (which is disastrous for classification). This is why tuning these weights was an important part of the process. With this model, the better-functioning weights for this exercise are 0.3 and 0.7.