Deep Learning (EE553) - Project 1: Classification, weight sharing, auxiliary losses

Hugo Grall Lucas, Théophile Schenker and Costa Georgantas
Deep Learning EE553, EPFL, Switzerland

***Abstract*— The problem studied involves classification of pairs of input images showing digits, depending on which one is the lowest. A common global architecture was defined, and different strategies were explored. The use of weight sharing and an auxiliary loss was shown to improve performance significantly. For the second stage of the network (comparison between the digits), a human-designed relation was compared to trained layers, yielding similar results. The best results on the test set are around 6% error (mean).**

I. INTRODUCTION

This first project - *Classification, weight sharing, auxiliary losses* - is part of the EPFL (EE553) course *Deep learning*. The purpose of this work is to explore and test some of the main concepts and tools of deep learning through a practical programming exercise using the well known *PyTorch* framework. The stated problem is the classification of pairs of digit images, in order to assess whether the first number is smaller or equal to the second one. The standard MNIST[1] dataset is used to generate pairs of input images. Labels are available for the class of each image (the digit) as well as for the target for each pair of image.

First, the different implemented architectures are described. Then, results are shown and analyzed to assess the interest of using weight sharing and auxiliary losses, as well as the effect of the metric choice on the classification performances. After that, a comparison is made between a fully trained network and a more guided one, where the last part is hard-coded. The main results are finally summarized in a table, before a concluding discussion.

II. ARCHITECTURE

As shown in Figure 1, different techniques were explored to solve the problem described above. All of the approaches use a separate path for each image at the beginning, ending in a 10-dimensional space (implicitly being the 10 digits). Then the two paths merge to combine the results for both images into the final result (telling which digit is the lowest).

The first part with separated paths uses a combination of three convolutionnary layers, bringing each image in a single dimensional space with 256 channels, followed by two linear layers. Rectified linear units are used at the end of most of the layers. The weights in this part can either be the same for both paths, or different, as explored in Section III.

[1] http://yann.lecun.com/exdb/mnist/

The second part can either be performed using additional layers (*Model A*), or it can be designed manually (*Model B*), assuming a relation between the output of previous layer and the probability that an image belongs to each class (digit).

Both models can be used either only with a loss on the target, or also with an auxiliary loss on the class for each image. For *Model B*, as the last part is fixed, it is even possible to use only the class information. MSELoss as well as CrossEntropyLoss can be used. A standard Stochastic Gradient Descent (SGD) step is used for the whole project.

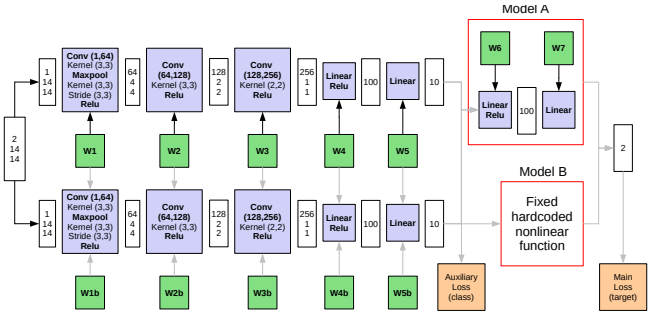

Fig. 1. Graphical description of the different architectures explored in this project. Different gray arrows were used, depending on the model and options that are selected.

III. WEIGHT SHARING

The aim of this section is to evaluate the performance improvements offered by a weight sharing strategy. Indeed the nature of the input samples is suitable for such a technique and it seems advantageous to take this symmetry into account. To recall, the weight sharing mechanism consists of using the same weights *W1*, *W2*, *W3*, *W4* and *W5* (see Figure 1) for both paths.

As introduced in Section II, two *Model B* architectures with and without weight sharing mechanism were compared ten times, with each time shuffled train and test samples.

TABLE I
SET OF PARAMETERS USED DURING THE TRAINING STEP.

Parameters				
Loss	**Batch size**	**Label**	η	**epochs**
Mean squared error (MSE)	100	Target only	1e-1	25

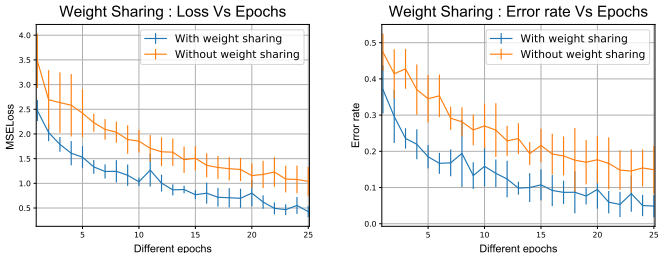

Fig. 2. Train results, with and without weight sharing - mean and standard deviation computed over ten runs. **[Left:]** Evolution of the train loss (on the targets). **[Right:]** Evolution of the error rate during the train.

Figure 2 summarizes the results obtained during the train. The left plot shows the evolution of the loss over the epochs. In spite of large standard deviations, it can be concluded that enabling weight sharing leads to quicker learning and better final performance. The same observation can be made from the right plot that shows the evolution of the error rate in training. The test results are also better using weight sharing, as can be seen on Table IV. Therefore, all the next sections use weight sharing.

IV. AUXILIARY LOSSES

For a human, it seems natural to recognize each digit independently before doing the comparison to conclude which one is the lowest. With the developed architecture, it is also possible for the network to separate each digit at the end of the first stage (separated paths, where features of each picture are extracted) but another strategy could be used as well. Using an auxiliary loss on the class forces the network to separate each digit, making this an objective.

Model A (see Section II) is used in this section, both with and without an auxiliary loss. The goal is to study the induced performance change. When the auxiliary loss is enabled, the total loss is simply a non-weighted sum of a loss on the classes and a loss on the targets.

TABLE II

SET OF PARAMETERS USED DURING THE TRAINING STEP.

Parameters				
Loss	**Batch size**	**weight sharing**	η	**epochs**
Cross Entropy	100	True	1e-1	25

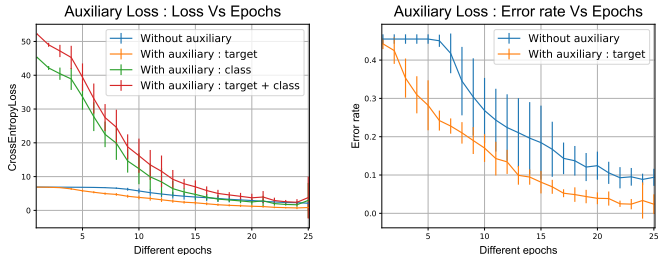

Fig. 3. Train results, with and without auxiliary loss - mean and standard deviation computed over ten runs. **[Left:]** Evolution of the train losses. The loss used is the red one (sum of a target loss [orange] and a class loss [green]). **[Right:]** Evolution of the error rates.

The left plot of Figure 3 shows that the magnitude of the loss on the targets is much smaller than the magnitude of the loss on the classes. With a weighted sum, this could be balanced, for example to give more importance to the target objective.

On the right plot, one sees that the performance improvement is significant. Guiding the network with the auxiliary class results in a faster learning at the beginning, and also a much smaller final error. The improvement is significant as well for the test results (see Table IV), where only 6.6% error remains using weight sharing and an auxiliary loss.

V. ENGINEERED PROBLEM

As introduced in Section II, the second stage (comparison between digits) can be defined using two linear layers (and a ReLU) that are trained, or it can also be hard-coded by interpreting the output of the first stage as a quantity linked with the probability for one image to belong to each class. The relation between the output of the first stage and this probability is defined as the following:

$$P_{class=i} = normalize_j\left([output(i) - min_j(output(j))]^6\right) \quad \text{(V.1)}$$

where j loops to each output of the 10 dimensional space and i is the class for which the probability is evaluated.

An interesting parameter in this formula is the power used to transform the output into a probability. Its value was chosen experimentally.

The probability for the first digit to be lesser or equal to the second one can then simply be deduced from the sum of the probability of each possible combination where this happens.

TABLE III

SET OF PARAMETERS USED DURING THE TRAINING STEP.

Parameters		
Batch size	η	**epochs**
100	1e-1	25

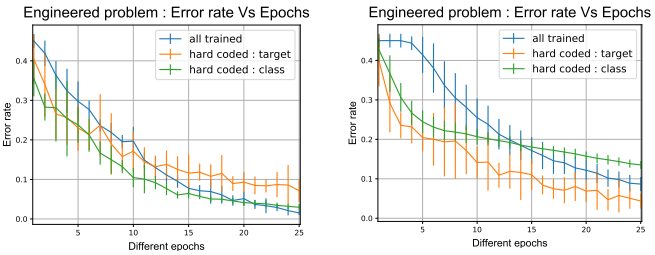

Fig. 4. Train error comparison for different strategies: *Model A* with an auxiliary loss (blue), *Model B* trained with a loss on the target only (orange) and *Model B* trained with a loss on the class only (green) - mean and standard deviation computed over ten runs. **[Left:]** CrossEntropyLoss **[Right:]** MSELoss

Figure 4 shows that lower train error rates can be reached using CrossEntropyLoss for this problem. This is confirmed also for the test error rates on Table IV.

Using CrossEntropyLoss, *Model A* and *Model B* are both able to reach interesting results. In all these cases, the network is strongly encouraged to separate the classes at the end of the first stage. Training the two layers of *Model A* for

the second stage does not improve the results significantly, compared to the engineered hard-coded version trained on the classes, which even has slightly better results on the test set with 5.6% error (Table IV). With only two linear layers and one ReLU, the second stage of *Model A* cannot be as nonlinear as the one of *Model B* (power 6). This might limit its performance, and more layers could be added in a further study.

Interestingly, using MSELoss, the best train errors are obtained by training the hard-coded version on the targets (less than 5% error at the end). However this is not verified for the test set (16.5% error). This difference indicates a clear overfit.

VI. RESULTS SUMMARY

TABLE IV

SUMMARY OF THE RESULTS OBTAINED WITH THE TEST SAMPLE.

Model	**Error rate mean [%]**	**Error rate std [%]**
Model B, weight sharing, trained on targets only (MSELoss)	16.4	4.6
Model B, no weight sharing, trained on targets only (MSELoss)	24.9	4.5
Model A, weight sharing, no auxiliary loss (CrossEntropyLoss)	17.5	1.7
Model A, weight sharing, auxiliary loss (CrossEntropyLoss)	**6.6**	1.1
Model A, weight sharing, auxiliary loss (MSELoss)	14	1.1
Model B, weight sharing, trained on targets only (MSELoss)	16.5	2.9
Model B, weight sharing, trained on targets only (CrossEntropyLoss)	16.5	3.3
Model B, weight sharing, trained on classes only (MSELoss)	14.2	2.2
Model B, weight sharing, trained on classes only (CrossEntropyLoss)	**5.6**	1.0

VII. CONCLUSION

The benefits of using weight sharing and an auxiliary loss for this problem were confirmed, with significant impacts both on the train and test performances. With around 6% test error, the two best explored methods used CrossEntropyLoss, one with a classical auxiliary loss, the other trained on the classes with a human-designed class-to-target stage, involving a probability interpretation. MSELoss was accompanied by good train results in the case of *Model B* trained on the targets, but the less good test results indicated an overfit.

In the architecture used, 10 dimensions remained at the end of the first stage for each branch, allowing the network to use a onehot encoding for the classes and even enforcing it in some cases (auxiliary loss, or *Model B* trained on the classes). In a further study, it would be interesting to try reducing this number, to see how the network can find more compact ways to store the information needed to solve the problem.

Finally, designing a network to solve this problem and evaluating different strategies was motivating and helped us getting familiar with *PyTorch*.