Handwritten digit comparison with different architectures of neural networks

Aurélien ORGIAZZI, Gaspard VILLA, Leonard KARSUNKY
EPFL, EE-559, Deep Learning

***Abstract*—The objective of this project is to test different architectures for our deep neural network to compare two digits visible in a two-channel image. The following report will show the importance of using an auxiliary loss and the principle of weight sharing for the architecture of our problem. It will also show that batch normalization and dropout regularization can increase the efficiency and accuracy of our model.**

I. INTRODUCTION

In the subject of this project, we will study the efficiency of different architectures of Neural Networks. The main method to test our different architectures is to compare two handwritten digits and see which is higher than the other one. For that, we will use the MNIST handwritten digit database and the format of the input will be a tensor of dimension $N \times 2 \times 14 \times 14$. The size of this tensor represents the number N of pairs of digits with 14×14 gray-scale pixels that we will use as input. The target will be a tensor of size N that will tell us if the first digit is lesser or equal to the other one with target classes $\{0, 1\}$. On the other hand, we will also have access to the class prediction of each digit. All of these three tensors will be given by the function `generate_pair_sets(N)` defined in the given python file `dlc_practical_prologue.py`. After getting our test set and train set, we will run them on our different architectures using (or not) weight sharing and auxiliary loss. These architectures will be presented in the first section of this report. After that, we will look at the results we obtained to compare these different architectures and see which is more efficient for this problem.

II. PRESENTATION OF DIFFERENT ARCHITECTURES

For this project, we will use different architectures principally based on the use (or not) of the weight sharing and an auxiliary loss. To begin with, the goal of our network is to compare two digits. Thus, the first part consists in digit recognition and the second part consists in comparing them. For the first part, we chose to implement either a multi-layer perceptron or a convolutional neural network. Without using weight sharing, each digit goes through a different network for its identification (MLP or CNN), and with weight sharing, only one network is used to identify the two digits. Then, for the second part with the comparison of the two digits, we have implemented a MLP with two hidden layers. To train the model, we have also tested the use of an auxiliary loss, which means that instead of considering only the target result of the comparison, the loss takes into account the knowledge of the classes of the two digits. Then, we will present in details the different architectures used to solve this problem.

Using no weight sharing and no auxiliary loss (`NoWS_NoAL`). In this architecture, we do not use auxiliary loss, which means that the only thing which is used to compute the loss is the result of the comparison between the two digits and not the classes of the two digit. In addition, there is no weight sharing in this architecture. Therefore, each of the two digits goes through a different network to identify the class before going through the final comparison network. The two networks to identify the digits have exactly the same structure, however their parameters, computed during training, are clearly different.

Using auxiliary loss without weight sharing (`NoWS_AL`). For this architecture, the same structure as above is kept to identify the digit and compare them, but this time, the loss computed uses the target classes of the two digits. Thus, the loss used for the backward pass is the sum of the loss of the result of the comparison and the loss of the classes of the two digits. We chose to put a coefficient (equal to 10) in front of the two losses computed with the target classes of the digit. Indeed, we have reasoned that these losses have a greater importance because the task consisting to identify a digit is much more difficult than to compare two digits (only $10^2 = 100$ different possibilities).

Using weight sharing without auxiliary loss (`NoWS_AL`). This time, we used weight sharing in the network. Therefore, the network used to classify the two digits is the same, and so share the same parameters. But the architecture does not use an auxiliary loss, which means that only the target result of the comparison is used to compute the loss.

Using weight sharing and auxiliary loss (`NoWS_AL`). Finally, for this architecture, the auxiliary loss described above is used as well as the weight sharing.

Using batch normalization (`B`). To make our deep neural network less sensitive to the change in the distribution of inputs to layers in the network, we implemented a technique called batch normalization which standardizes the inputs to our network. After some extensive research on the topic, it is said to apply batch normalization before the activation

functions (ReLU) of each prior layer instead of to the inputs directly, since ReLU is a non-Gaussian distribution (for Tanh, it would be wiser after the activation functions). Indeed, we see a much lower variance of final test error when using batch normalization this way.

Using dropout (D). Another deep regularization technique we implemented is dropout, in which we removed units from our neural network at random during the forward pass on each sample, and put them all back during test. This method works well to address over-fitting in our neural network. Unlike batch normalization, we applied dropout after the activation of each layer. We found the optimal dropout parameter to be $p = 0.1$, meaning we keep 90% of the parameters.

Using both batch normalization and dropout (BD). By combining both batch normalization and dropout, we can benefit from both regularization techniques at the same time, using batch normalization to standardize the inputs for each layer and dropout to simulate having a large number of different network architectures using a single model. Again, batch normalization is used before and dropout is used after the activation functions of each layer, respectively.

All of the above architectures were designed and implemented for the first part of our deep neural network, i.e. for the classification of our two digits. In the second part of our network, the goal is to compare the two digits and return the result. Here we used a simple MLP with two hidden layers. A MLP is sufficient for this network since it is just the comparison of two digits and a more complex convolutional network would not be necessary for that kind of simple problem.

III. Results

For the presentation of the results, each architecture will be tested over 20 rounds (i.e. 20 identical models with different initializations) that includes 20 different test sets, each containing 1000 samples from MNIST, to make sure that the mean error represents a good average of the efficiency of the model. Then, the models will be trained over 50 epochs, instead of 25 epochs.

First of all, we will compare for a MLP network the four principal architectures that are: NoWS_NoAL, NoWS_AL, WS_NoAL and WS_AL. The obtained results are presented in the figure (1). We can clearly see that the architecture with no weight sharing and no auxiliary loss is the worst one in terms of test error. The next worst architecture is the one with weight sharing but still no auxiliary loss. And finally, the architectures with auxiliary loss gives the best results with better prediction whether there is or not weight sharing. This reveals that in the case of MLP, the use of an auxiliary loss is very important and the addition of weight sharing is useful principally when there is no auxiliary loss. When there is an auxiliary loss, the improvement of adding weight sharing is quite small (difference of $\approx 0.8\%$, see Table I).

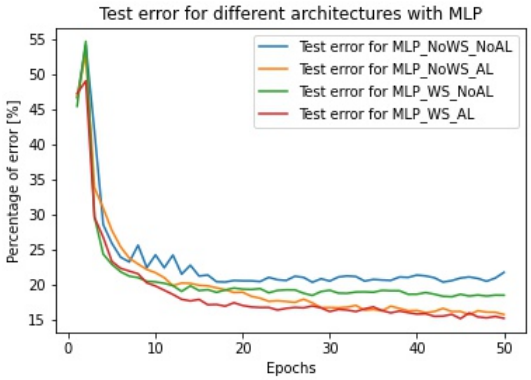

Figure 1: Test error with MLP network and different architectures.

The second experience was made on a convolutional neural network and we have done the same thing as previously, i.e. we have the curves of test error for the four principal architectures introduced before. The obtained results presented in the figure (2) shows similar results than the one obtained for MLP in the previous figure (1). In fact, the add of only weight sharing or auxiliary loss respectively improve the prediction for our problem. And when the both are adding together we obtain the best results.

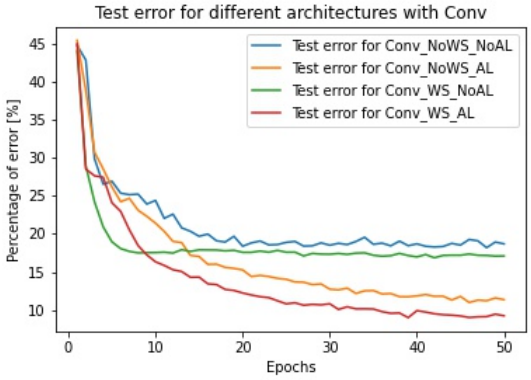

Figure 2: Test error with convolutional network and different architectures.

Now, we take the best architecture for MLP and Conv and we will add batch-normalization and dropout to them and see how it improves the results. For MLP, the results presented in the figure (3) shows us that the add of dropout (only) gives an improve on the final test error of 4%. But the batch-normalization gives better results with an

improve of almost 8%. And, we can see that in the case of MLP, when we combine batch-normalization and dropout, the test error is almost the same than if we only had batch-normalization. So, skipping some units with dropout for training generally improves the performance when there is no batch-normalization. Finally, for the case of MLP, batch-normalization is preferable instead of dropout, and combining the two seems to be useless for improving the results.

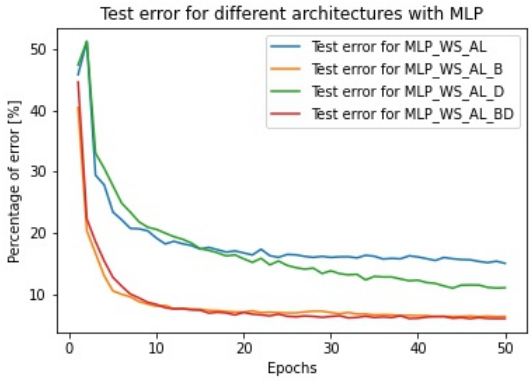

Figure 3: Test error with MLP network and batch-normalization and dropout.

Now, we will compare the add of batch-normalization and dropout for the case of convolutional network. In the figure (4), we can observe similar results than previously with an increasing of the performance for adding dropout or batch-normalization, with a better one for the last. But, the combination of dropout and batch-normalization gives worst results than the ones obtained with batch-normalization only. But, since the difference of performance is less than 1% (see table I), we can just say that when there is batch-normalization, the add of dropout doesn't influence the performance.

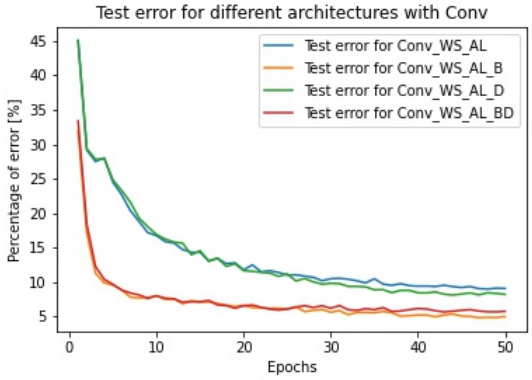

Figure 4: Test error with convolutional network and batch-normalization and dropout.

Finally, we will compare the convolutional network and the MLP with their best architectures. The results are presented in the figure (5) and we can see that with the best architecture for convolutional network, that is with weight sharing, auxiliary loss and batch-normalization, we obtain a better performance than the one obtained with MLP's best architecture that is with weight sharing, auxiliary loss, batch-normalization and dropout.

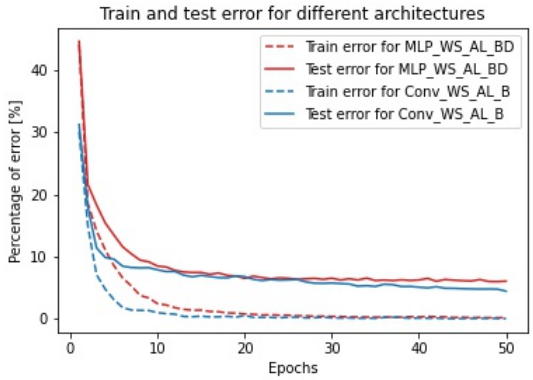

Figure 5: Train and test error with convolutional network and MLP.

All the final test error of the different architectures for MLP and convolutional network are resumed in the Table I that follows. The corresponding standard deviation of each of the test error are also displayed in the table. We can observe the improvements given by the different architectures and that with an adapted one, we can obtain the best performance with an error of 4.42% and a standard deviation of 0.62%.

Architectures	Test error [%]	Architectures	Test error [%]
MLP_NoWS_NoAL	21.85 ± 2.25	Conv_NoWS_NoAL	18.68 ± 1.30
MLP_NoWS_AL	15.83 ± 1.50	Conv_NoWS_AL	11.36 ± 1.33
MLP_WS_NoAL	18.59 ± 1.22	Conv_WS_NoAL	17.09 ± 1.33
MLP_WS_AL	15.06 ± 1.47	Conv_WS_AL	9.07 ± 1.49
MLP_WS_AL_B	6.41 ± 0.85	Conv_WS_AL_B	4.42 ± 0.62
MLP_WS_AL_D	11.08 ± 1.33	Conv_WS_AL_D	8.22 ± 1.47
MLP_WS_AL_BD	6.05 ± 0.69	Conv_WS_AL_BD	5.76 ± 0.89

Table I: Test error for each architecture with the corresponding standard deviation.

IV. Conclusion

To conclude this report, we can say that the weight sharing and the auxiliary loss plays an important role in the performance of a MLP or a convolutional network for comparing two digits visible in a two-channel image. Also, the combination of the two gives better results than using only one of them. And, finally we have seen that the batch-normalization and the dropout can be good options to increase the performance.