Deep Learning Mini Project 1

Authors: Gaia Carparelli, Leonardo Pollina, Hugues Vinzant

May 14, 2019

1 Introduction

This first Deep Learning Mini Project aims at testing different architectures to compare two digits in a greyscale image. It especially focuses on the impact of weight sharing and the use of an auxiliary loss during the training process. The final goal is to implement a Neural Network such that, given two digits, it predicts if the first digit is lower than or equal to the second one. Everything was implemented with `Pytorch` library only.

2 Data

The dataset was generated with the function `generate_pair_sets(N)` which returned the six tensors shown in Table 1.

Name	Tensor dimension	Type	Content
`train_input`	N x 2 x 14 x 14	float32	Images
`train_target`	N	int64	Class to predict {0,1}
`train_classes`	N x 2	int64	Classes of the two digit {0,...,9}
`test_input`	N x 2 x 14 x 14	float32	Images
`test_target`	N	int64	Class to predict {0,1}
`test_classes`	N x 2	int64	Classes of the two digit {0,...,9}

Table 1: Data shape and content.

In particular, notice that the `target` refers to the binary classification task (i.e. the first digit is less or equal to the second one) and it represents the main target label. On the other hand, `classes` contains the additional information related to the digit classification itself. Plus, training and test sizes were fixed at N = 1000.

3 Models and Methods

The methods and models implemented and compared in the frame of this Mini Project are described hereafter.

3.1 Neural Networks

In order to investigate the impact of architecture's complexity on the model performance three main architectures were explored: Shallow, Multi Layer Perceptron (MLP) and Convolutional Networks. For each architecture, the effect of auxiliary loss and of weight sharing was analyzed. A total of sixteen combinations were implemented.

3.1.1 Parameters

Neural Networks having several hyper-parameters, some of them were fixed based on theory and already existing evidence [1]. Thus, *ReLU* activation function was preferred over others (*e.g. Sigmoid, Tanh*) as it is theoretically the best and the most widely used [2]. In the output layer, the *SoftMax* activation function was selected, giving us a value between 0 and 1 for each class interpretable as the probability of belonging to that class. The binary cross-entropy loss and the optimizer Adam were also chosen. Moreover, in order to speed up the computational time, mini batches with fixed size of 100 samples were employed. The epochs number was fixed at 25, this value was then validated for the best and final model. The other parameters such as the learning rate, the number of hidden neural units and lambda (indicating how much the auxiliary loss is taken into account) were tuned.

3.1.2 Auxiliary loss

Since information about `classes` was provided for each image, the prediction of single digit was additionally performed and its loss was taken into account as a penalty term in order to strengthen the training phase of the neural network. Also this loss was computed by the Cross-Entropy loss. The rationale was to push the model to perform the main binary classification taking into account also the loss of linked to another task such the classification of the single digit. The final loss will thus correspond to the following expression:

$$loss_{tot} = loss_{main} + \lambda * (loss_{aux_1} + loss_{aux_2}) \quad (1)$$

3.1.3 Weight Sharing

In order to lower the computational cost and since the use of two different weight matrices or activation maps (resulting from the application of Convolutional layers) for the two digits does not really make sense, weight sharing was implemented. Using the same mapping for the two images should help the model to generalize better by reducing the risk of overfitting.

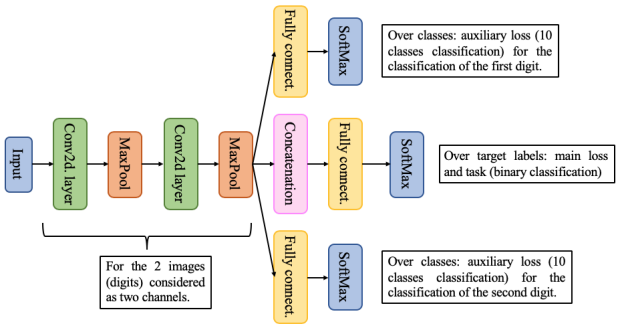

Figure 1: Scheme illustrating the principle of weight sharing, i.e. the fact that the first layers are used for both images, as well as the one of auxiliary losses. This structure corresponds to the first Convolutional architecture tested in the frame of this project.

3.1.4 Shallow

First of all, a Shallow model was tested. With "Shallow" we refer to the simplest neural network architecture consisting of an input layer, one hidden layer and an output layer (performing the binary classification task). When exploring contribution of the auxiliary loss two additional output layers are inserted in order to predict the single digit class. By taking into account the different cases (both with and without auxiliary loss and both with and without weight sharing), four shallow models were thus implemented. This method can be considered as being a sort of baseline in the frame of our Mini Project.

3.1.5 MLP

Our Multi Layer Perceptron model is very similar to the Shallow model but having two hidden layers instead of one. The leading idea for the implementation of such model is that it has been proved that a deeper model (even if here it corresponds to the addition of only one layer) could perform better. This is likely to be due to a more accurate feature extraction from the input data. Also in this case the four combinations (with and without auxiliary loss and with and without weight sharing) were implemented in order to detect the best combination.

3.1.6 Convolutional

Two different architectures were tested for the Convolutional Networks to test again the impact of depth. The first structure, a "more shallow one", was composed by two convolutional layers, being considered as the feature extraction part, and by two final fully connected layers, considerable as the classifier itself. A max-pooling operation was applied after each convolutional layer (with a kernel size of 2 and a stride of 2, meaning no overlap). In the second model two additional convolutional layers were added. In order to avoid a too strong dimensionality reduction of the input, a single max-pooling operation was applied after the second convolutional layer.
For each of the two architectures the four combinations (with and without auxiliary loss and with and without weight sharing) were tested.

3.2 Hyperparameters tuning

In order to tune the hyperparameters (learning rate, number of hidden units and lambda) of each model a *Grid Search* was performed. In this way, every possible combination was tested. Values were taken in the following lists: lambdas = [0.25, 0.5, 0.75, 1], etas = [10^{-5}, 10^{-3}, 10^{-1}], hidden units = [50, 100, 200, 300].
In order to get an unbiased estimation of the performance, the input dataset was split in two using a 70-30 ratio. 70% were used for training, while the remaining 30% were employed as validation set. The tuning of hyperparameters was performed over this validation set.
Finally, the model was trained using the best hyperparameters on the whole 1000s samples of training set. The unbiased estimation of the model performance was obtained using the untouched test dataset. This operation was performed 30 times re-loading the data every time and randomly initializing the weights.

4 Results

Results of this Mini Project are described hereafter.
In Table 2 the mean accuracy and the standard deviation for the sixteen different optimized models are shown.

Neural Network Architecture	**Accuracy**	**Std**
Shallow_NOsharing_NOaux	0.7188	0.0584
Shallow_sharing_NOaux	0.7755	0.0114
Shallow_NOsharing_aux	0.7604	0.0403
Shallow_sharing_aux	0.7751	0.0096
MLP_NOsharing_NOaux	0.8028	0.0130
MLP_sharing_NOaux	0.7963	0.0190
MLP_NOsharing_aux	0.8050	0.0094
MLP_sharing_aux	0.8122	0.0168
Conv1_NOsharing_NOaux	0.7978	0.0136
Conv1_sharing_NOaux	0.8043	0.0174
Conv1_NOsharing_aux	0.8043	0.0117
Conv1_sharing_aux	0.8154	0.0143
Conv2_NOsharing_NOaux	0.8148	0.0147
Conv2_sharing_NOaux	0.8300	0.0120
Conv2_NOsharing_aux	0.8205	0.0127
Conv2_sharing_aux	**0.8319**	**0.0111**

Table 2: The mean performance and the standard deviation for each model tested over 30 repetitions. Notice that at each repetition the dataset was reinitialized randomly.

The model giving the best mean accuracy was the Deep Convolutional Neural Network with weight sharing and the addition of auxiliary loss (labeled as Conv2_sharing_aux). It has a mean accuracy of 83.19% and a std of 0.0111. This model had as best hyperparameters lambda = 0.5, learning rate = 0.001 and 100 hidden units. The overall architecture of this best model is described in Table 3.

Layer	
Conv_2d	1 x 16 x 5 x 5
Conv_2d	16 x 32 x 3 x 3
Conv_2d	32 x 64 x 3 x 3
Max_Pooling	2 x 2 with stride = 2
Conv_2d	64 x 128 x 2 x 2
Fully_connect_Aux1	512 x 10
Fully_connect_Aux2	512 x 10
Concatenation	//
Fully_connect_Main	1024 x 100
Fully_connect_Main	100 x 2

Table 3: Detailed structure of the best model, which corresponds to the second deep architecture implemented with auxiliary losses and weight sharing. Notice that the concatenation refers to the fact that we put together the activation maps from the two different images. The separated block in the middle refers to the auxiliary loss.

Then, the validation and training performance for in-

creasing number of epochs (from 0 to 50 epochs) was analyzed for the best model. A plateau was reached around 20 epochs as shown in Figure 2.

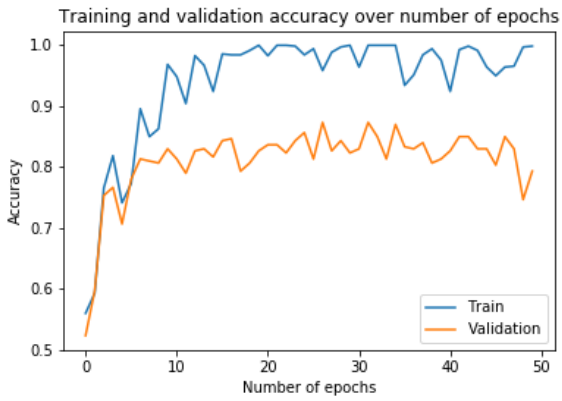

Figure 2: Evolution of both training and validation sets accuracy depending on the number of epochs used for training.

In order to compare the effect of weight sharing and auxiliary loss, the four types of architecture for the best model were compared using boxplots as shown in figure 3.

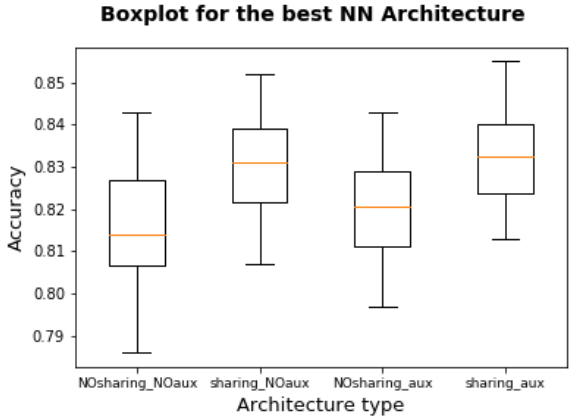

Figure 3: Boxplot for the four different architecture types of the best Neural Network Model.

5 Discussion

This project aims to analyze different neural network architectures. In particular by comparing the role of depth, weight sharing and auxiliary losses in improving the performance of the classification task. Networks containing only Fully Connected layers were compared with ones also containing Convolutional layers.

As expected, the Convolutional and the MLP architectures (in all their combinations) performed better than the Shallow ones. This reinforces the hypothesis than the deeper the model the better the performance.

By taking into account a general trend, we can also state that, if MLP and first Convolutional structure models are showing really close performances, the second Convolutional architectures outperformed all the others. This phenomenon can be due to two main factors: the depth and the use of Convolutional layers. These are in fact known to be used as the common strategy for tasks of image classification. A combination of both factors is also possible. In particular, we can speculate that the leading factor is the level of depth, since the performances corresponding to the second Convolutional architecture (the deeper one) are clearly higher with respect to the first Convolutional architecture (the simpler one). However, in order to validate the hypothesis that Convolution layers are effectively better than the Fully Connected layers we should have tested also a deeper architecture for the MLP network type: in this case it would have been clear the effect of having a MLP versus a Convolutional Deep Neural Network.

Regarding the best model (corresponding to the the deeper Neural Network containing four Convolutional layers), the low standard deviation indicates a low variance in the model, no matter what the initial dataset or the weight initialization are.

For what concerns the use of the auxiliary losses, we can see from Table 2 that the addition of a loss term (depending on the ability of the model to well classify the digit) has proved to be useful in all cases, with the only exception of the Shallow architecture. Indeed, the mean accuracy of the models corresponding to the *_aux* option proves to be better than the corresponding architecture with the *_NOaux* option. We can speculate that the addition of such term helps the model to learn the "right" features, i.e. the ones based on the semantics of the digit and thus linked to the classification of the single digit, instead of image-specific and possibly misleading features, which could also lead to overfitting.

Moreover, the use of weight sharing also seems to improve the performance of the corresponding model (the one with the *_NOsharing* option). As mentioned above, this technique allows us to reduce the computational cost. However, it can also reduce the risk of overfitting. Indeed, the fact of training the same activation map on both images means that the weights are learned over a double number of images. All these observations should be further validated by statistical-tests, such as t-tests, in order to assess if the differences found are significant or not.

In conclusion, we can affirm that our model benefits from the use of both weight sharing and auxiliary losses. Of course, there are several aspects that could be further analyzed in order to improve the performance of our model. One natural idea would be to make the grid search more dense to sharpen the optimization of the hyper-parameters. In particular, a more accurate estimation of eta (learning rate), of the number of hidden units and of lambda could be achieved. The number of epochs does not look to need more tuning. Indeed, as shown in Figure 2 the model does not take advantage of additional training after ≈ 20 epochs. Thus, a number of epochs equal to 25 seems to be acceptable. Of course, this aspect should be validated in the case of even more complex structures.

Another interesting aspect would be to analyze the impact of the activation function and thus to try for example to use *Sigmoid*, *Tanh* or the more direct evolution of *ReLU*, *Leaky-ReLU*. Finally, overfitting could be further reduced by the addition of *DropOut* layers with different dropping probabilities, another hyper-parameter to tune.

References

[1] Nielson, M. (2015). "Neural Networks and Deep Learning", Determination Press.

[2] LeCun, Y; Bengio, Y; Hinton, G (2015). "Deep learning". *Nature*, 521 (7553): 436–444.