Ecole Polytechnique Fédérale de Lausanne

EE-559 - Deep learning
Mini-projects

Project 1: Classification, weight sharing, auxiliary losses

Nikitas Papadopoulos - 262824

Nicolas Kieffer - 270647

Sarah Fleury - 262329

May 2020

Robotics Master 2 - Summer semester

1. Introduction

The goal of this project is to test different architectures to compare two digits visible in a two-channel image. It aims at showing in particular the impact of weight sharing, and of the use of an auxiliary loss to help the training of the main objective. In order to do that, we had to implement a deep learning neural network that predicts for 1000 pairs of 14 x 14 grayscale images if the first digit is lesser or equal to the second. We chose to compare four architectures by plotting the error rate performed by each one of them.

2. Theory

At first, we considered implementing two types of neural network: a fully connected (FC) one and a convolutional (CNN) one. However, we quickly dropped the implementation of the FC one, as a little research made us realize that the CNN network would yield much better results for image classification, because of their translation invariant characteristics. Therefore, it was on the latter that we dedicated our time.
Much like most kinds of artificial neural networks, a convolutional neural network has an input layer, an output layer and various hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers. The convolutional layers simulate some of the actions that occur in the human visual cortex, which is why the CNN is extensively used for image classification. The CNN takes as input a tensor which becomes a feature map after passing through the convolutional layer(s). Then, the pooling layer groups together neuron clusters into one neuron, thereby reducing the dimensions of the data. Finally, the fully-connected layers classify the data.

3. Implementation

The implementation of our Convolutional Neural Network was inspired by the practical number 4. The layers that we implemented follow this order: a convolutional layer, a pooling layer, a second convolutional layer, a second pooling layer, and two fully-connected layers. In order to implement the choice to use weight sharing or not, all layers were duplicated inside the neural network. If one chooses to share weights in his model, he simply inputs `weight_sharing` when calling the `train_model` function, which will then run the second image in the same layers as the first image, thereby reusing the adjusted weights used for the first image, on the second image. If, however, one inputs `no_weight_sharing`, the second image will pass from a new set of layers, and the weights will not be shared. The results for these different architectures are discussed in section 4. The auxiliary loss is implemented in a similar manner: the user enters `auxiliary` or `no_auxiliary` in the train model, and the loss is either calculated only on the outcome (comparison) or as a sum of the losses over the two images and the outcome.
Once we set the main structure, the next step was choosing the values of the hyper-parameters. We had to tune them in order to achieve satisfying results while having approximately 70 000 parameters in total. More precisely, we obtain a model with 72 554 parameters. The learning rate and the number of hidden layers are other two hyperparameters that had a major influence on our results, and that needed fine-tuning. In the end, we settled for 150 hidden layers, a learning rate of $1\ 10^{-3}$, and used a kernel size of 3 for both the convolutional and the pooling layers. We choose a stride and padding of 2 for the convolutional layer, and a stride of 2 for the pooling layer. The following table shows, for the Test error based on the outcome (prediction of the comparison between the two digits), different results of the mean and the standard deviation with different values of the learning rate, with 150 hidden layers.

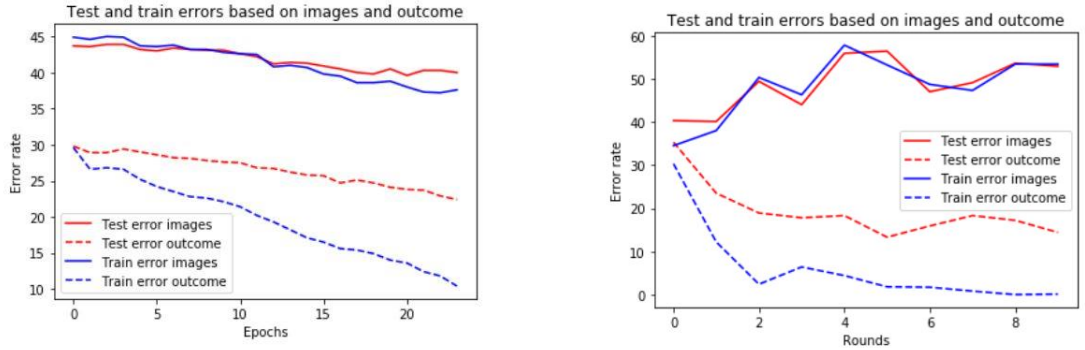
The images are loaded from the `generate_pair_sets` function, and they are then sent in the `train_model` function. This latter uses the Cross Entropy Loss criterion for the loss calculation, and

Learning rate	Mean	Standard dev.
0.005	29.440	7.222
0.003	12.310	3.615
0.001	12.100	3.381

a Stochastic Gradient Descent optimizer, and loops over the number of epochs (25), creating tables for the test and train errors by calling the `compute_error_rate` function. This last function computes the error rate, in percentage, of either the images (with option `images`) or the comparison prediction (with option `outcome`). After the 25 epochs, the values are printed for the current round, and the error rate is plotted. This is repeated over 10 rounds, after which the mean and standard deviation is given for the training and testing error rates over the images classification or the comparison.

4. Results/Performance

As mentioned earlier, we tested different architectures for the image classification with our convolutional neural network, activating or not the weight sharing and the auxiliary loss. First, we are going to analyze the results of the convolutional network that doesn't use neither weight sharing nor auxiliary loss. We have plotted the testing and training errors of both the images classification and the outcome predictions, over 10 rounds (one round = 25 epochs).



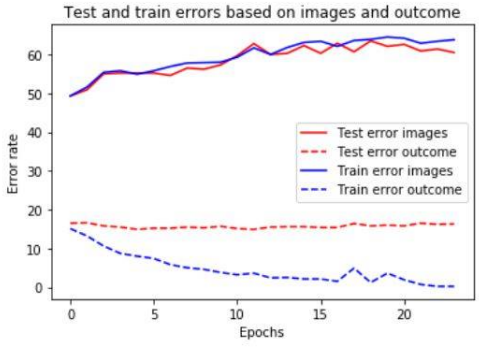
(a) Error rate of a random round

(b) Error rate over 10 rounds

Figure 1: Error rate of architecture 1 (neither weight sharing nor auxiliary loss

We can see by this first graph that this architecture does not yield great results. Indeed, the classification results are very poor, only half of the time does the prediction match reality, which lets us believe that the classification of the digits is totally random. However, the outcome curves give acceptable results, as the test error finds itself to be in the 15-20% mark. That being said, an overfitting can be spotted beginning at the 5th round mark, with the train error approaching zero and an increase in the test error.

Now, let's take a look at the network, but this time by adding the weight sharing attribute. We observe on figure 2 that the addition of weight sharing contributed on the outcome curve: it is able to achieve good results starting from the first round, as opposed to the first architecture that needed more rounds to display satisfactory results. This is due to the fact that the model can re-use the same weights that it found for the first image, on the second image, therefore better comparing the images and bringing the standard deviation down for the training and testing of the outcome.

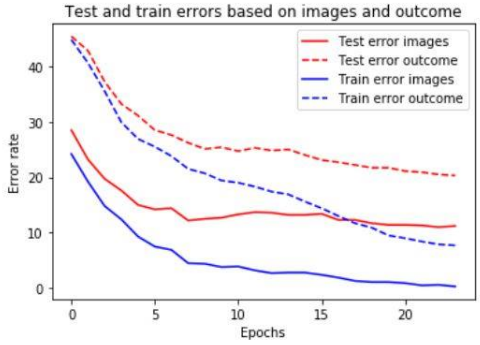


(a) Error rate of a random round

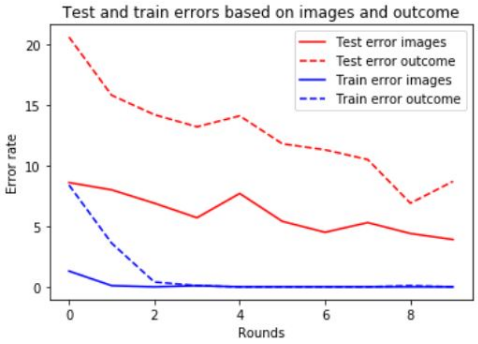
(b) Error rate over 10 rounds

Figure 2: Error rate of architecture 2 (weight sharing but no auxiliary loss)

Next, we take off the weight sharing but turn on the auxiliary loss feature.



(a) Error rate of a random round

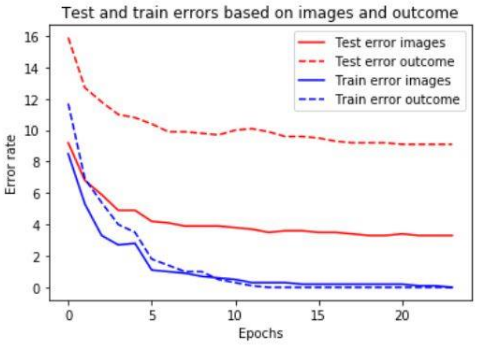


(b) Error rate over 10 rounds

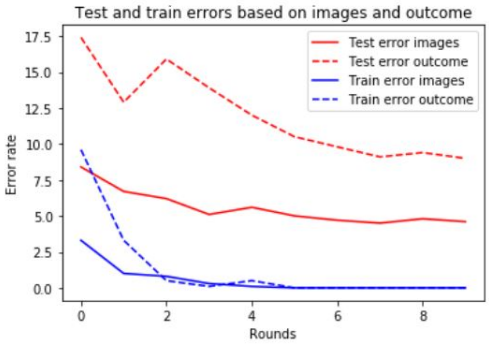
Figure 3: Error rate of architecture 3 (no weight sharing but auxiliary loss)

Thanks to this change, we observe a dramatic improvement in the images training and testing curves: they are able to classify the images correctly 90%+ of the time during training and about 80-90% of the time during testing. This is due to the fact that we added the loss of the images in the loss calculation, therefore forcing the model to optimize the image classification as well, as opposed to only the digit comparison. The model overfits after the second round, as it falls to nearly 0% of misclassified images, although it is able to keep a good percentage of correctly classified images in testing.

Finally, let's analyze the full model, implemented with both the weight sharing and auxiliary loss attributes on figure 4.
As predicted, this architecture yields the best results. The improvement mainly happens for the outcome test and train curves, as the second image can now be classified more accurately thanks to the weight sharing feature.
In the table 1, we can observe the mean and standard deviation for error rates based on outcome for the different architectures used. In the table 2, it is the same except that it concerns error rates based on images. We can deduce from it that the weight sharing feature mainly acts on the standard deviation, and the auxiliary loss on the mean over the rounds.



(a) Error rate of a random round



(b) Error rate over 10 rounds

Figure 4: Error rate of architecture 4 (weight sharing and auxiliary loss)

Architecture	**Mean**	**Standard deviation**
no WS, no AL	48.87	±5.68
WS, no AL	60.79	±3.47
no WS, AL	6.31	±2.07
WS + AL	5.17	±1.26

Table 1: Mean and st. dev. for test error rates based on images

Architecture	**Mean**	**Standard deviation**
no WS, no AL	19.29	±5.95
WS, no AL	17.63	±2.45
no WS, AL	12.71	±3.66
WS + AL	11.99	±2.83

Table 2: Mean and st. dev. for test error rates based on outcome

5. Conclusion and possible improvements

In conclusion, we were able to test different architectures and assess the importance of weight sharing and auxiliary loss in a convolutional neural net, as seen by the results in section 4. However, the performance could have been further improved by implementing one or more other features. For example, a learning rate decay, which drops the learning rate by a factor every few epochs, would have resulted in smoother curves as the model would have adapted better to the training set. Another parameter that could have been chosen differently is the optimizer: we decided to use a SGD optimizer, but others could have been implemented, such as Root Mean Square Propagation or Adaptive Moment Estimation. For a future project in neural network, all of these options will be carefully considered, thanks to this mini-project in which we were able to learn and put to use all of these different concepts.