Mini Project 1 - EE-559

Deep learning architecture benchmark on MNIST images

William Cappelletti, MA, Charles Dufour, MA, Fanny Huguelet, MA

Spring semester 2019

1 Introduction

We test the performance of different Neural Network architectures in recognizing which of two handwritten digits is greater. We use the python-based deep learning platform *PyTorch* to create and train different networks; we start from simple models, trying different activation functions and regularization techniques, to end with deeper and more elaborate structures, such as residual and siamese networks. Finally, we discuss their performance on the task, their convergence speed and their training time.

In order to confront the strengths and limitations of each model, we create various architectures as sub-classes of `torch.nn.Module`, which are then trained on 1000 pairs of images from the MNIST database [2] and tested on as many. These pairs of handwritten digits are grey-scale images stored as two channels of a $[2 \times 14 \times 14]$ Tensor (the dimension correspond, in the order, to channels, height and width). For each pair we know its *class*, i.e if the first digit is greater than the second, and what the two digits are, i.e the *labels* of the images.

2 The architectures

We start with a basic Fully Connected Neural Network (FCNN) with an input layer of 392 nodes (one for each pixel of the two 14×14 grayscale images), two hidden layers of 1000 nodes each and a final layer of two nodes, for the classification of the bigger digit. On top of this basic layout, we implement a FCNN with dropout training and two FCNN with batch normalization after each hidden layer, one of which is trained with dropout.

We then implement two different Convolutional neural networks, the first one with three convolutional layers and a final fully connected layer to predict the two classes. Its kernels sizes are respectively 3, 5 and 5 and the number of channels goes from 2 (the two images) to 8 to 16 to 16. The second network has four convolutional layers, of which the first three are as in the former model, while the last one aggregates the 16 channels into two, corresponding to the prediction classes. Using these same layouts we then implement two other networks with batch normalization after the first three convolutional layers. All the previously cited basic model were tested with 3 different activation functions after each layer, more precisely using ReLu, Tanh and LeakyReLU.

After that, we move to more elaborate structures. Firstly, we implement a Residual Neural Network (ResNet), whose basic block takes as input a Tensor of size $[12 \times 14 \times 14]$, applies a first convolution with a kernel dimension of 5 and a 2-padding (to keep the same height and width), then applies ReLU and one additional convolution with the same properties as before. Finally, the block adds to the latter output the input tensor and applies ReLU. The ResNet takes as input the two images, then applies a first convolution with a kernel size of 1 and 12 output channels. After that it applies 6 of the previous blocks, then averages over the channels, applies ReLU and, finally, applies a fully connected layer which predicts the two classes.

The second advanced model we implement is based on weight-sharing. It is a Siamese Network, since it splits the two input images and makes them pass, separately, through the same sub-network which outputs a Tensor of size $[1 \times 10 \times 1]$. These two Tensors are then concatenated and passed through a fully connected layer, which predicts the two final classes. In principle the sub-network could be any structure taking a $[1 \times 14 \times 14]$-tensor and outputting a $[1 \times 10 \times 1]$, therefore, we test this model a first time with a simple convolutional network (Sequentially: convolution, max-pooling, ReLU, convolution, max-pooling, ReLU, reshaping, an hidden fully connected layer with 100 nodes, ReLU, a final fully connected layer to output on ten classes). Then, we test the siamese network using the same ResNet as before, with the slight change in the output classes of the ResNet, now being 10 instead of 2.

Thanks to our choice of the siamese sub-network output size, we can interpret it as if the model tries to learn in a first moment the two digits and then uses them to infer which is the bigger one. Pushing further this idea, we can use an auxiliary loss to suggest to the sub-network to actually predict the digits while we train it (for precise information about the training we refer to the implementation at [1]). We test these two networks also using the auxiliary loss during training.

3 Benchmark

For each basic architecture we experiment its behaviour by testing it 15 times, with both data and weights initialization randomly chosen. The performances of the residual and siamese networks are tested only 10 times, since they require more time to train, with randomized data and initial weights as well. We ran all the experiments using a GPU nvdia GTX970m, and we reproduced those using the basic models also on a VM with one single core and no GPU.

Each network is trained on 50 epochs, using the Adam optimizer and computing the gradient on batches of 250 pairs, using the Cross Entropy loss function. The learning rates are shared only by similar models, in order to grant convergence, therefore the basic FCNN uses 1e-3, the fully connected nets trained with dropout 2e-4, all the convolutional ones use 4e-4 and both the ResNet and the Siamese architectures use 5e-3. All models are then tested on 1000 new pairs and their accuracy scores are reported in Figure 1.

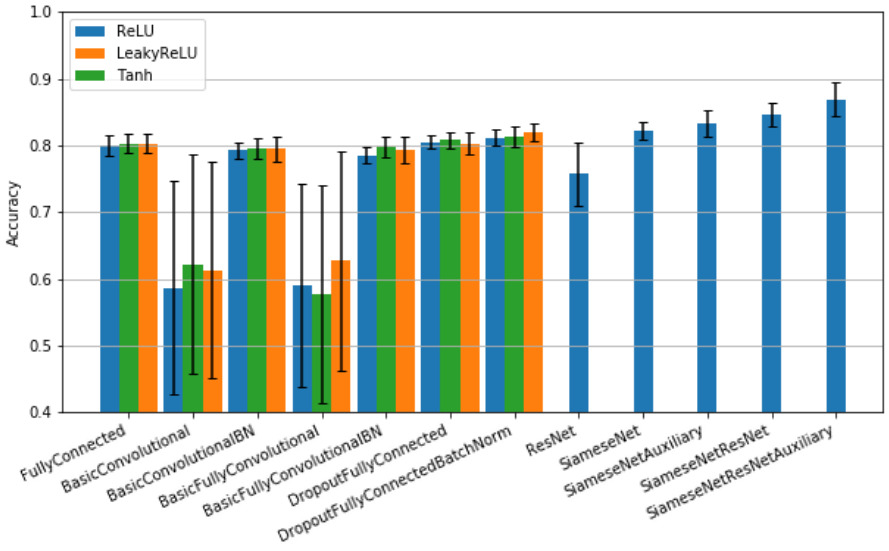

Figure 1: Accuracy scores on test sets.

We see that the mean accuracy of the two basic convolutional designs is low with each activation function (0.5865 ± 0.3186 for the convolutional with final FC layer and 0.5903 ± 0.3058). The best models are the most elaborate ones, in particular the Siamese net with a Residual sub-network trained with an auxiliary loss (0.8693 ± 0.05042).

The training times for 50 epochs explode with the complexity of the model, and thus with its number of parameters, as we can see in Figure 2. All the simple models can be trained in less than 1.2sec, while the best performing one is also the slowest (SiameseResNet with auxiliary loss trains in (114.9 ± 1.4696)sec).

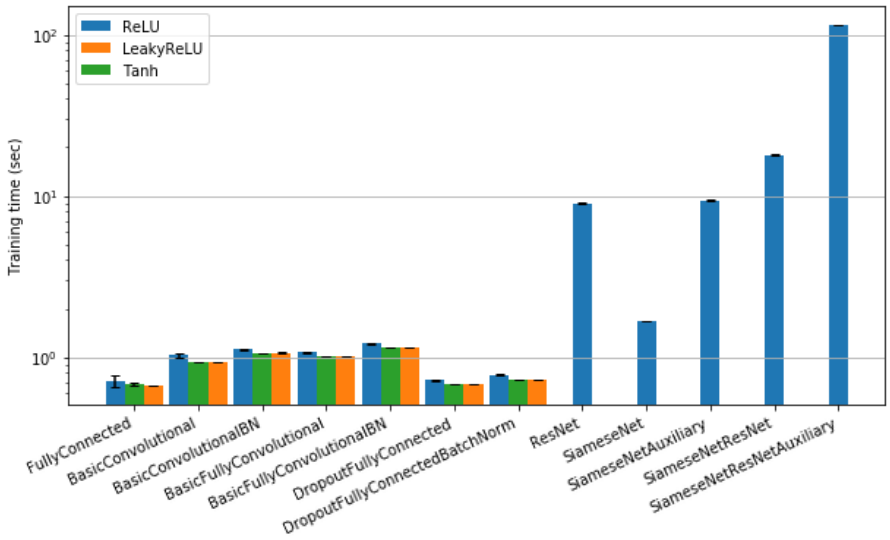

Figure 2: Training times for 50 epochs on GPU (nvidia GTX970m).

4 Discussion

As we would expect the accuracy and the training time scale accordingly to the number of parameters. The convolutional models perform poorly due to the fact that their learning is very susceptible to weight initialization. As we observed that during some of the experiences it diverged or kept oscillating away from the minimum. This can be solved by changing the learning rate or, as we can see, by using batch normalization, which we observed speeding up the convergence. We observed the same on the FCNN, which were boosted even more by dropout training, although this slowed down the computations (as we can see in Figure 2). The residual network performs worse than expected, again due to a limitation of our benchmarking setup; we do not tune its learning rate and with the one cited here 50 epochs are not enough for it to minimize the loss.

All the implementation, along with the full data we gathered, are available in the GitHub [1].

References

[1] Cappelletti, W. and Dufour, C. and Huguelet, F. *Deep-learning-mini-projects*, (2019), GitHub repository, `https://github.com/dufourc1/Deep-Learning-mini-projects`

[2] LeCun, Y. and Cortes, C. *MNIST handwritten digit database*, (2010).