Ex. 2 NN and Deep Learning

Federico Bottaro

January 19, 2020

Abstract

In this document we are going to explore the field of the hand written digit analyzing the MNIST dataset that provide a collection of hand written number from 0 to 9. In particular we are going to learn this dataset using a feed forward neural network developped using pytorch.

1 Project development

In this document we are going to build a feed forward neural network to learn and predict data that come from the MNIST dataset. This dataset contain picture of dimension $28 * 28$ so each number is represented by 784 values in a greyscale. Figure 1 show two examples of data.

(A) (B)

Figure 1: *Some numbers contained in the MNIST dataset*

Firs of all we open and prepare the data before working on them. To open the data in ".mat" format we use a simple function from scipy:

```
mat = scipy.io.loadmat('./MNIST.mat')
mat.keys()
X = mat['input_images']
Y= mat['output_labels']
```

and then we split this dataset with the respective labels into train ant test set (80 - 20 split) using a random shuffle. To conclude the preparation of the date we convert the list into tensor object for pytorch.

Now we can move to the topic of the document and describe how we build the network. First of all we evaluate the performance considering the structure of the network and we decide to preserve a sort of symmetry in the number of neurons that we try. The input layer has 784 neurons and we try to evaluate the accuracy on the test set in two case:

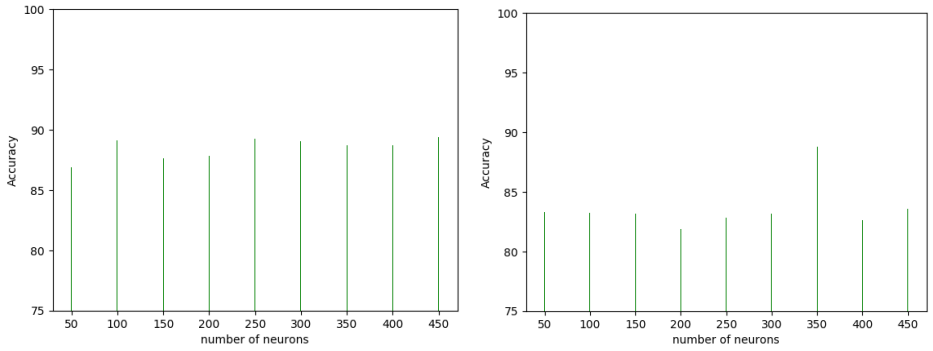
- Just one hidden layer with 256 neurons and the output layer with 10 neurons
- Two hidden layer with respectively 256 and 64 neurons.

We choose those value because we want to reflect the fact that the learning of the image is in an abstract way compressed by passing from a 28×28 immage to a 16×6 to a 8×8 . In this first tuning for the structure we apply a relu function between the layer , we choose

an Adam optimizer with a Cross entropy loss using a learning rate of 0.001 and train for 700 epochs (all parameters that becomes from the literature except for the learning rate that we set ad hoc looking at the graphic of the loss function for the training and the test set). With this simple structure we get the fallowing result:

$$accuracy_{1h} = 97.05\% \qquad accuracy_{2h} = 96.95\%$$

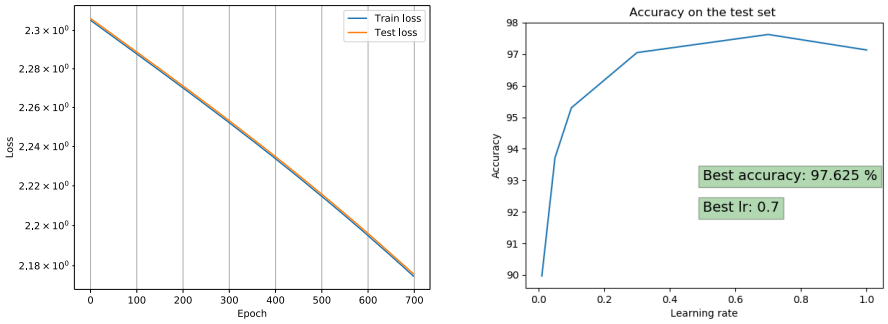
We can see that the score don't change soo much and is already pretty good so we choose to fix the structure of the neural network with a single hidden layer and we proceed to the tuning o the other parameter. Notice that also other number of neurons was evaluated as we can see from figure 2 but all the values of accuracy are bellow the one reported above.



(A) *Accuracy with one hidden layer as function of number on neurons* (B) *Accuracy with two hidden layer as function of number on neurons*

Figure 2: *Tuning for the structure of the neural network*

First of all we try to move from an Adam optimizer to a SGD. Without changing any other parameter we get that using a learning rate of 0.001 the accuracy was about 53% but looking at the graph of the loss we have the behavior shown in figure 3 a. This shape indicate that the learning is not completed with this combination of epochs and learning rate. After try different combination we have that set the epoch to 3000 is enought to guarantee the convergence. Figure 3 b show how the accuracy change as function of the learning rate.



(A) *Loss with SGD optimizer* (B)

Figure 3: *Some number contained in the MNIST dataset*

2 Training

So we have that the best accuracy is 97.45% that correspond to the fallowing set up:

- Structure of the network: input layer with 784 neurons, one hidden layer with 256 neurons and an output layer with 10 neurons that correspond to the various class
- Loss: crossentropy that we find in literature to be one of the most appropriate to this type of tasks
- Optimizer: SGD, that take longer time respect to Adam to learn but has a little bit more accuracy.
- Learning rate: 0.7

So we can now move to the real learning. Looking at the graphic of the loss for the training and test set we have that with 3000 epochs the algorithm seems to converge but the slope of this function still decrease so we try to set the number of epochs to 5000. Figure 4 show the loss function in this set up. As we can see set the epochs to 5000 don't give a very important gain in term of test error but the accuracy grows a little bit and we reach an occuracy on the test set of 97.625%.

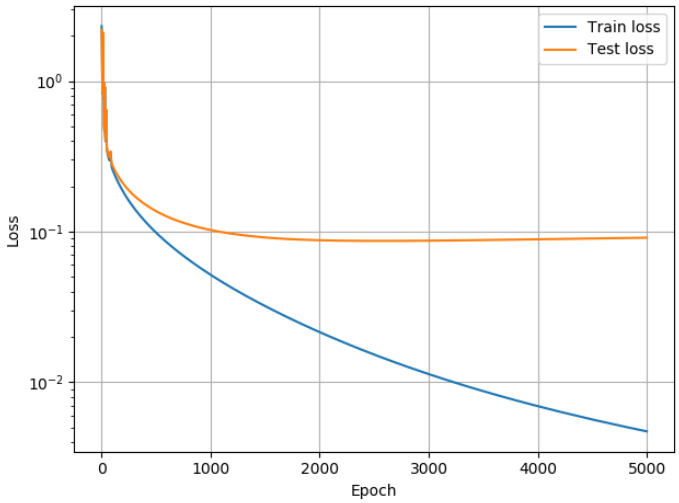
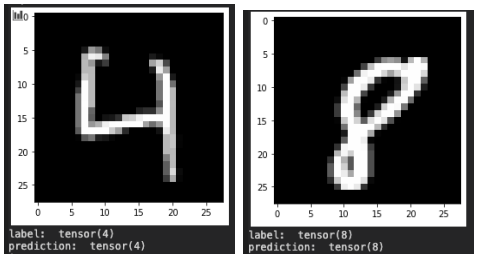


Figure 4: *Loss function for the final structure of the network*

3 Results

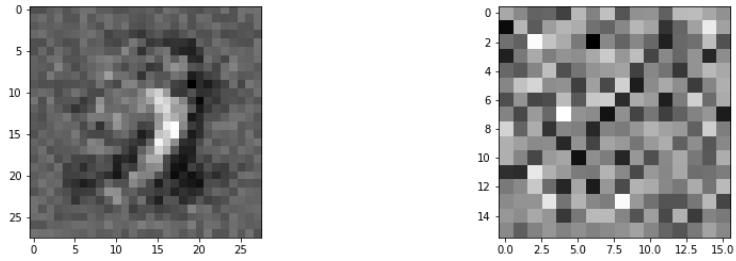
Once that the network is trained we can save the parameter using specifics pytorch function and load in another script that provide in few seconds a value of accuracy for an arbitrary hand written ".mat" file. Figure 5 Show the output of the script for some examples. That script also provide the accuracy on the analyzed file.



(A) (B)

Figure 5: *Output of the script*

Another interesting features that can be visualized is the receptive field of the network. We have two layer before the output layer so we can display in figure 6 an example of a receptive field from the first layer and an example from the second one. We can comment noting that in the first layer we can distinguish more or less a shape (in the first figure we can see something like the number 2) while in the second one we can't recognize any shape.



(A) *Receptive field from the first layer* (B) *receptive field from the second layer*

Figure 6: *Output of the script*

4 Extra

Just for comparison a convolutional neural network was also implemented without requiring any specific tuning about the parameter. For this specific task is possible to find some structure of a network in the literature and we have replicated this structure using pytorch. With this structure (detail of the code can be found the attachments) we found an accuracy for the same test set of 92.21% that is certainly an under estimated accuracy for a convulutional neural network and it can be improved by studying the space of the parameters but it can be a good number for make a comparison with our FF network. So we can conclude that for a feed forward neural network we find a good structure.