Neural Network and deep learning: homework 2

of Tommaso Tabarelli

Abstract

In this homework we are asked to do modify a given python script in order to train some neural networks. The goal is to learn main features of pytorch package using it to tune neural network parameters while trying to learn hand written digits in MNIST dataset.

Code development

First of all we had to undestand how the code works and how the Neuran Network (NN) model is implemented using pytorch.

Then we had to modify it to:

- Load data from a *.mat* MNIST file (in the given script data are generated using a second order polynomial and classified using two classes: above or below the polynomial curve);
- Train a neural network on the loaded data classifying images using digits as labels/classes;
- Look for a *good* set of parameters, i.e. parameters which makes the network classify images with *sufficient* precision the digits.

Moreover, cross validation, early stopping and the use of different activation functions are suggested.

My choice was to work with jupyter notebook since I find it more familiar and cleaner; anyway a python script was finally developed as homework requests.

Data file is opened using *scipy.io.loadmat* and then X and Y are initialized using *input_images* column and *output_labels* colomn respectively. In X are stored the 60000 images, which are represented by (fletten) arrays of 784 pixels having float values in the [0,1] interval; in the Y instead are stored the corresponding labels. After been loaded, data and labels are shuffled (using the same random seed) and then split in *train* and *test* sets having 50000 items and 10000 items respectively.

The code was modified to allow NN to be feeded with input array of dimension 784 and to return an array of 10 scaled float values representing the probabilities that the input image shows a certain digit. To make it able to be trained on a classification task a *softmax* operation was added to the output layer. It was also modified to work with a generic activation function. Furthermore, to let the NN return also the guessed digit (i.e. the digit that have the higher probability) the *additional output* part of *forward* method was slightly modified.

To prepare the model to be fit on data, *sklearn* and *skorch* libraries are imported and the NN model is inserted in *skorch.NeuralNetClassifier* method, with all parameters to initilize it (the parameters are inserted only conventiolally to be able to handle immediately the *NeuralNetClassifier* object, for example to train it before using it in a grid search procedure).

To implement an *EarlyStopping* procedure a proper skorch module is imported. The early stopping was chosen to have *patience* equal to 3 (arbitrary choice to try to consider both stopping when validation error starts to increase and a possibility to improve in a small period after the "local" minima of the validation error).

To train on data, a cross validation tecnique was chosen. The *GridSearch* method in sklearn implement automatically the cross validation. Setting the *cv* parameter to 5 we are asking it to do a cross validation splitting the train set in 5 pieces using one of them as validation and the union of the reamining four as train set.

The parameters used to train the networks are the following:

- Learning rate: [0.1, 0.01, 0.001];
- Nh1 (first hidden layer number of neurons): [24,48,96];
- Nh2 (second hidden layer number of neurons): [12,24,48];
- Maximum nuber of epochs: [500]. This was chosen to be large thanks to the presence of Early Stopping procedure, which indeed almost always stopped the training procedure before the 200th epoch was reached;
- Activation function: [Sigmoid, ReLU, Tanh] (all already implemented functions in the module *torch.nn*).

GridSearch procedure results were the following:

- Learing rate : 0.001
- Nh1 : 96
- Nh2 : 24
- Activation function : Tanh

This NN was trained again using the train dataset (the 50000 images chosen as train set at beginning) and splitting it into 5 pieces to use the same procedure as before.

Results

After the train, the network was used to classify the test set images obtaining $97,11\%$ accuracy. Net parameters and weights were saved to a file and retrieved to check that the saved files contained all needed informations.

After this, the receptive fields of some hidden neurons were checked using the following procedure: the weights of first hidden layer neurons were reshaped in a 28×28 array and plotted using *plt.imshow* function; for the second hidden layer neurons the *first-to-second-layer* matrix weights were multiplied with the *input-to-first-layer* matrix weights so that the result would be a 784 array, which then was reshaped as before in a 28×28 matrix and finally plotted.

$$W_{i\to H1} \in M(96,784) \quad \Rightarrow \quad \text{Take one row and reshape it}$$

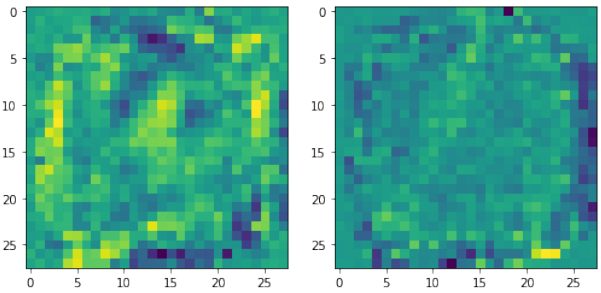
$$W_{H1\to H2} \in M(24,96) \quad \Rightarrow \quad W_{H1\to H2} \cdot W_{i\to H1} \in M(24,784) \quad \Rightarrow \quad \text{Reshape one row}$$

To try to have a clearer idea of what was going on in the network my choice was to try to visualize also the receptive field of the output neurons. To do it, the same reasoning as before was applied:

$$W_{H2\to \text{out}} \cdot W_{H1\to H2} \cdot W_{i\to H1} \in M(10,784) \quad \Rightarrow \quad \text{Reshape one row}$$

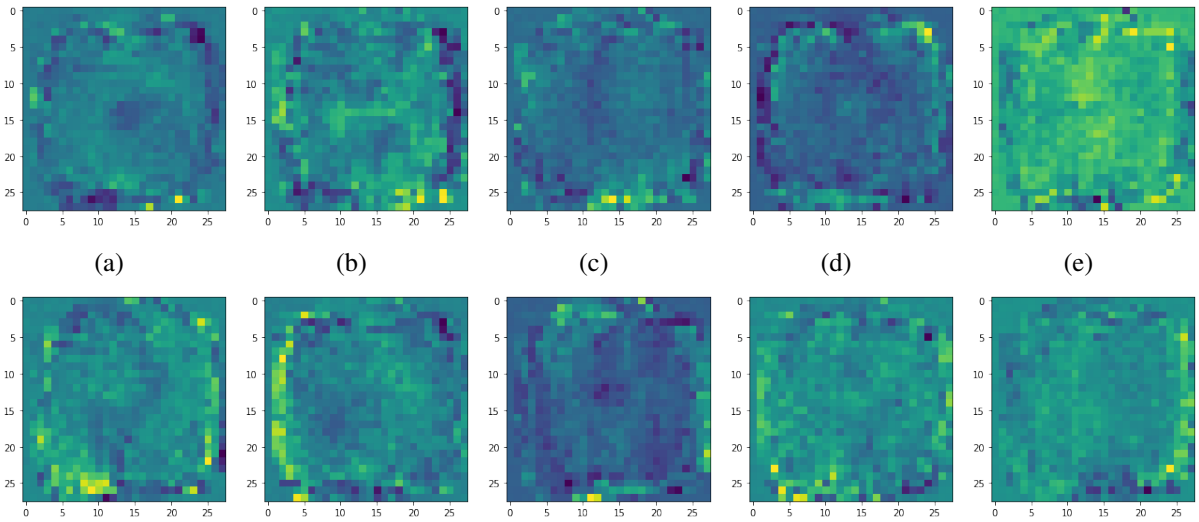
Given that the final network was trained many times due to some imperfections of the code, the different results were very interestings. Since the network is not a convolutional one, we expect that the receptive fields of the output neurons can be quite "weird" and may not represent the digits as we expect. Indeed this happened only once and I decided to save that result and to report it in the appendix.

As one can notice, there are no clear shapes even in the receptive fields of the output layer. Nevertheless, the network achieves a 97% accuracy on the test set. This can be due to the fact that, as



(a) (b)

Figure 1: (a) Receptive field of the 10th neuron in the first hidden layer. (b) Receptive field of the 9th neuron in the second hidden layer.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 2: Receptive fields of the output neurons of the last trained network. (a) represents the receptive field for the "0" digit, (b) that of "1" digit, (c) that of "2", and so on.

aforementioned, the network is not a *convolutional* network but it is a *feed forward* network, so it may represent the digits in ways that we can not undestand (not having proper "filters" analyzing some pattern in the images).

Ahother thing to stress is the fact that the receptive fields are evaluated in a linear way acting a matrix multiplication using the weights matrices, while the "real" evaluations are done using the activation functions in the network. Having this in mind, one should know that the results can be easily misleading or misunderstood.

Comments

This exercise was useful to undestand how to deal with pythorc's main features, how to implemet and train a neural network using pythorc and how to interface pythorc with *sklearn* to automatize even more the learning, having the cross validation and the early stopping already implemented in the libraries.

Appedix

As aforementioned, some interesting results were find in a model train. They are reported here and briefly discussed.

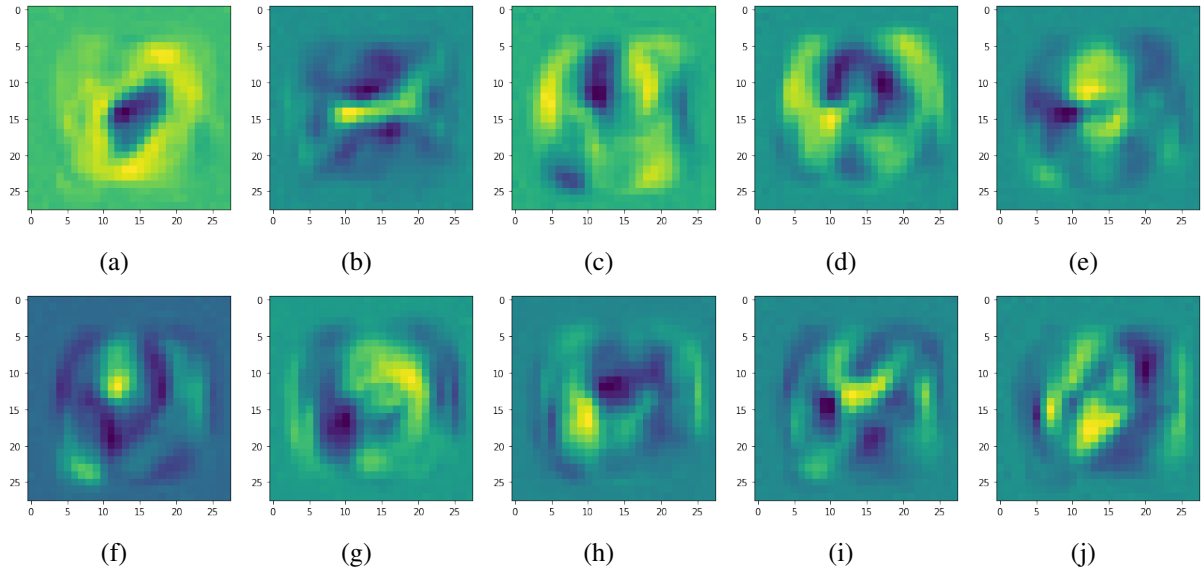


Figure 3: Receptive fields of the output neurons of an "intermediate" trained network. (a) represents the receptive field for the "0" digit, (b) that of "1" digit, (c) that of "2", and so on.

As one can notice, some receptive fields are quite "straight forward" and they can be guessed even without knowing the digit they represent (for example, "0", "1", "3" and "8" receptive fields are quite clear and understandable; of course, one must have in mind that the images are turned and modified).

These results seems to me a bit strange because what had been trained was not a convolutional Neural Network but it was a feed forward one. It seems that even if the model is simple and there are no spatial correlations highlighted in the model (as it should be in a convolutional network instead) because the images are passed as flatten arrays, in this case the model was "learning" the "natural" pattern of some digits; so it may learn itself that there are some kind of "typical patterns" in the data in a very complex way.