Neural Network and deep learning: homework 1
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 involving parameters tuning and handling while trying to learn a unknowk function.

Code development

First of all we had to undestand how the code works and how the Neuran Network (NN) model is implemented.
Then we had to modify it to:

- Load data from a text file (in the given script data are generated using a second order polynomial);
- Train a neural network on the loaded data;
- Look for a *good* set of parameters, i.e. parameters which makes the error small.

Moreover, cross validation, regularization, 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 the homework asks.
The train and test files are opened and read using *pandas*; using them, x and y vector are created both for train and test data.
The code was modified to make NN able to work with a generic activation function; 2 parameters were added to NN constructor, the activation function (which should be a callable object, a function indeed) and its derivative (which should be coherent with respect to the activation function). In the code, ReLU and ELU was implemented from scratch and NN was trained using sigmoid (already implemented in given script), ReLU and ELU.
To avoid overfitting, a regularization term was implemented. For this scope another class called *Network_reg* was created (copying most features from the previous NN class). A new updating rule was implemented, using *L2 regularization* and the following considerations:

$$L = \frac{1}{2}(Y - \hat{Y}) + \lambda ||\beta||^2$$

Where here β is the vector of *all* weights, thus the concatenation of W_0, W_{h1} and W_{h2} flattened. The update rule turns out to be (in the second and third equation the vector components indexes are omitted):

$$\left(\frac{\partial L}{\partial (W_0)_i}\right) = 2\lambda (W_0)_i + \left(\frac{\partial L}{\partial Y}\frac{\partial Y}{\partial (W_0)_i}\right)$$

$$\left(\frac{\partial L}{\partial W_{h1}}\right) = 2\lambda W_{h1} + \left(\frac{\partial L}{\partial Y}\frac{\partial Y}{\partial Z_2}\frac{\partial Z_2}{\partial H_2}\frac{\partial H_2}{\partial W_{h2}}\right)$$

$$\left(\frac{\partial L}{\partial W_{h2}}\right) = 2\lambda W_{h2} + \left(\frac{\partial L}{\partial Y}\frac{\partial Y}{\partial Z_2}\frac{\partial Z_2}{\partial H_2}\frac{\partial H_2}{\partial Z_1}\frac{\partial Z_1}{\partial H_1}\frac{\partial H_1}{\partial X}\right)$$

To train on data, a cross validation tecnique was chosen. Train test was split into 6 subsets of 20 elements each and training was implemented to use 1 subset as validation set and the union of the other as train set.

An *early stopping* algorithm was also implemented in order to avoid overfitting. It had been developed in a simple way: after a given number of epochs (which is an arbitrary choice basing on previous runs of the code), the first time the validation error gets bigger makes the algorithm stop training.

The parameters used to train the networks are the following:

- Nh1,Nh2 = [(50,50), (100,100), (200,200)]. Conventionally, to avoid wasting too much time in training, the networks were built using the same number of neurons for both layers;
- Activation functions = [sigmoid, ReLU, ELU].
- Learning rate was chosen to be adaptable and to be let "decay" during training. Values were:
  - 0.2 $\rightarrow$ 0.001 for sigmoid
  - 0.001 $\rightarrow$ 0.00001 for ReLU and ELU (they had smaller values to avoid high errors, indeed in first trainings when using LR:0.2 $\rightarrow$ 0.001 the error became *nan* in the second iteration and training was not successful).
- Minimum number of epochs was chosen to be:
  - 2000 for the sigmoid;
  - 5000 for ReLU and ELU.
- Regularization: "L2" regularization was chosen, with $\lambda = 0.01$ (higher λ parameters, such as 1 or 0.1, lead to mistakes due to weights becoming very large during the update).

The validation errors of all training were saved and the "best network" was chosen to be that with smallest average validation error. It turned out that *ELU* network with $Nh1 = Nh2 = 50$ neurons had the best performances (avg validation error = 0.75085).
This NN was trained again using the whole train dataset, chosing (arbitrarily) a number of epochs of 3000 (since there was no validation set no early stopping was possible).

Results

The image above shows the resulting function and the train data.

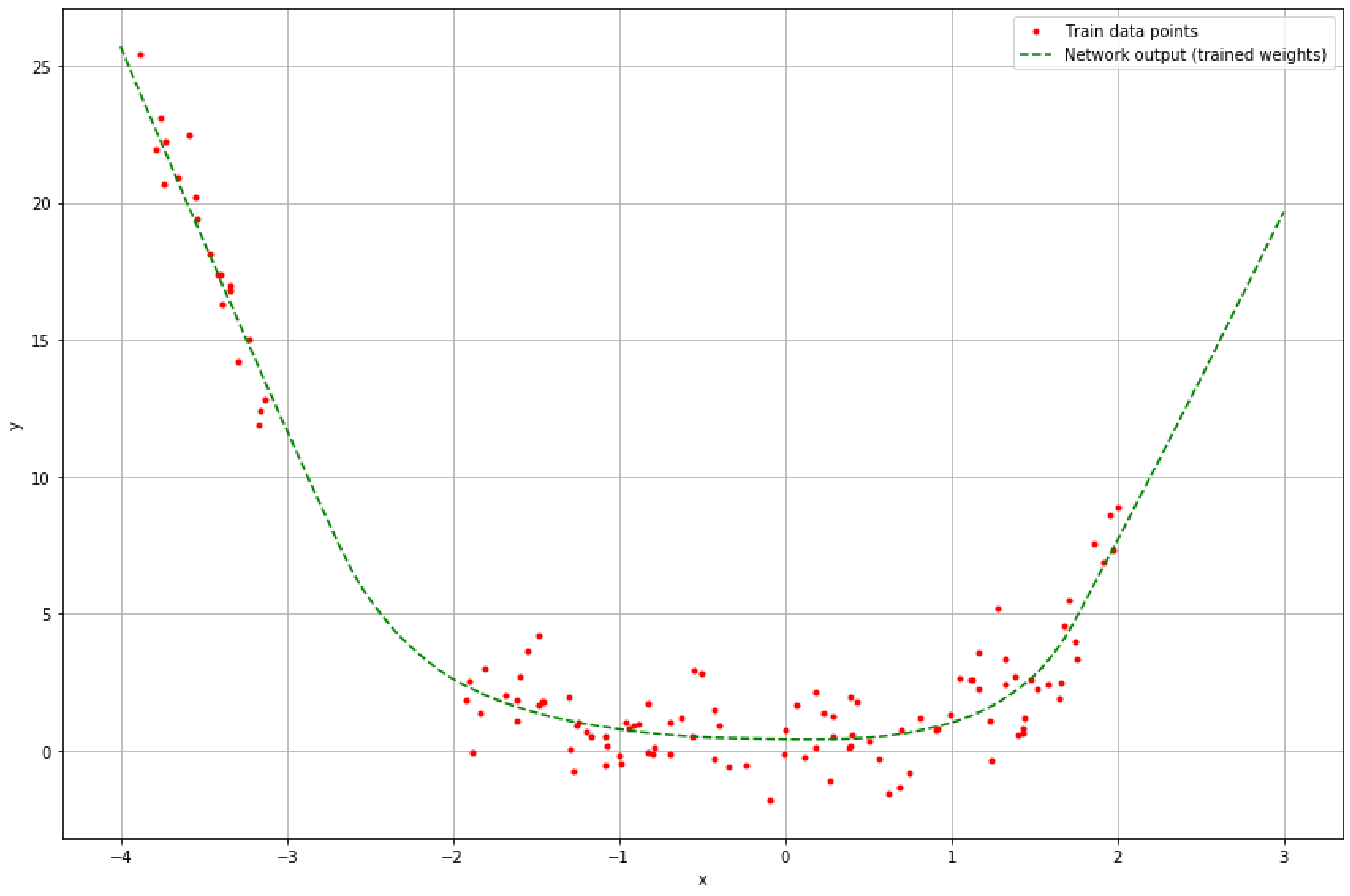

Figure 1: Function of the ELU network after training.

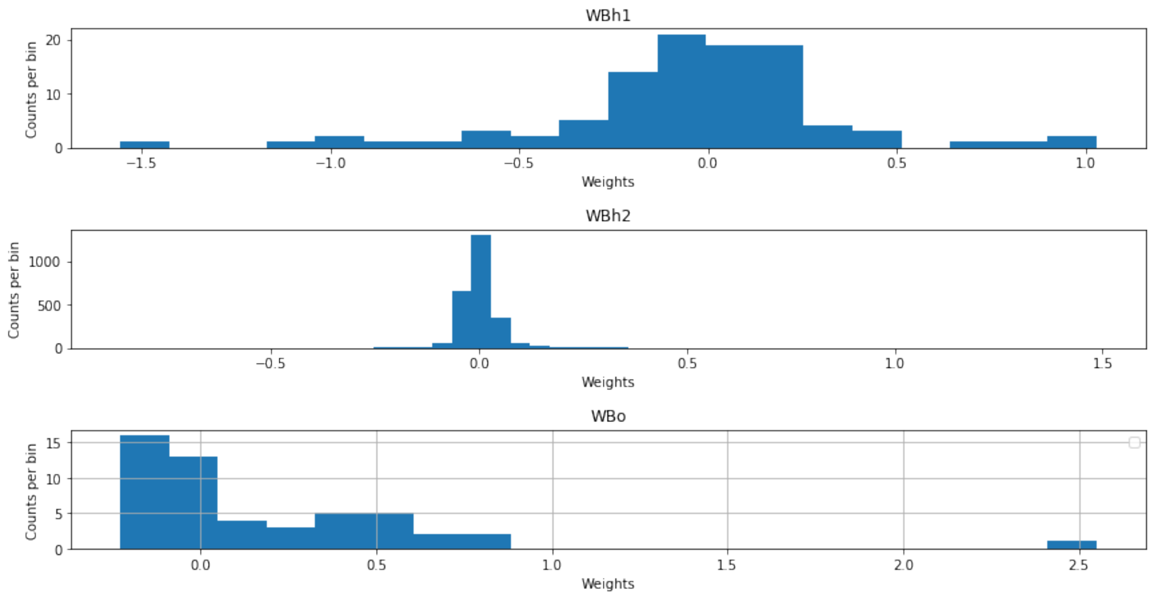

Figure 2: Distribution of the network weights after training.

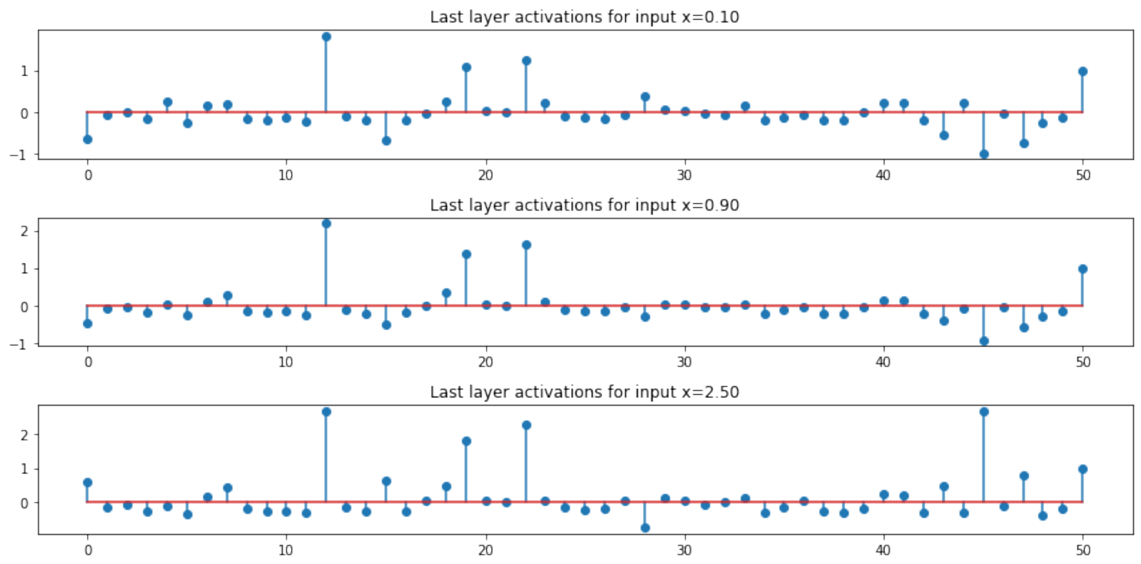

Figure 3: Neurons activations in some random cases.

From the first image one can notice that the network function is plausible: it tends to guess all points with almost no trace of overfit. One may ask if there is underfit because it is quite easy to recognize what seems a sinusoidal pattern which instead is not noticed by the network: this is a question which is hard to answer because we do not know the original function (the "sinusoidal pattern" may be only a strange pattern in noise and in that case we should claim our net is good because it does not notice it).
From the second image we notice that the weights are almost all near 0. This may be explained claiming that the network is made by 2 layers of 50 neurons each and they can represents too many parameters to fit the apparently simple data distribution we gave it. Indeed, looking at neurons activation in the 3rd image we can notice that few neurons activate and they are usually the same for different inputs.

The final test score of the tained network is: 0.759978.

Comments

This exercise was useful to undestand how to implement a neural network "by hand" and how to deal with it, how to avoid overfitting and which strategies one can adopt to reduce it. Also, it was useful to understand that the parameters tuning can be very computationally and time expensive, so analysis strategies should be chosen in advance and clearly motivated.
To improve the program one can train also different activation functions and different number of neurons in the layers. Also, a better early stopping can be discussed (for example, checking an increasing trend in validation error using 10 points instead of 1...).