Università degli Studi di Padova

Dipartimento di Fisica e Astronomia “Galileo Galilei”

Corso di Laurea in Physics of Data

# Week 3 Information Theory and Computation

Author

Marco Agnolon

Anno accademico 2019-20120

## The neural network

The aim of this exercise was to implement a neural network that is able to make fit a two-dimensional data. It was given a training set and a test set composed each of two columns of data, respectively x and y.
In figure 1 they are shown.

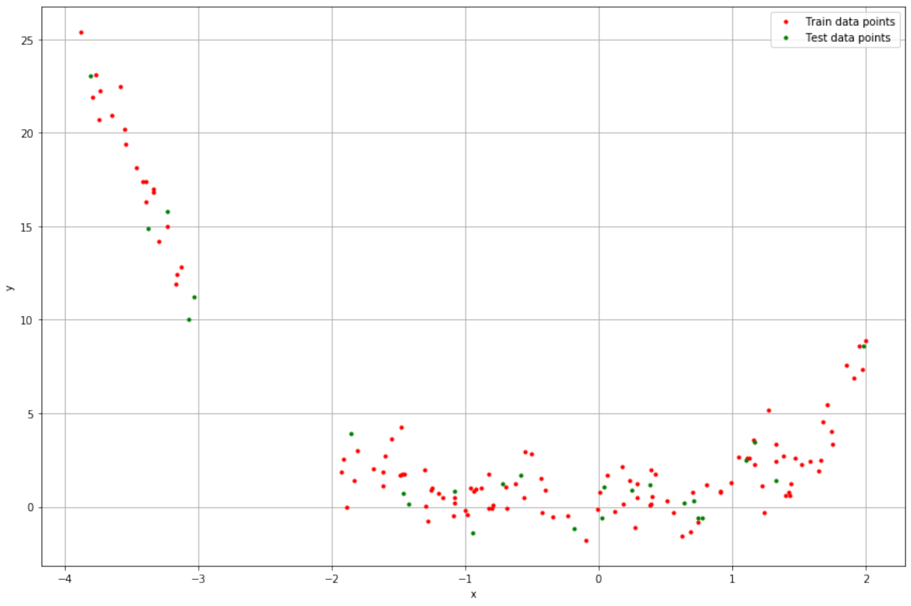

Figure 1: Dataset.

Then the network was created, a two hidden layers network with one input neuron and one output neuron. The weights were initialized with the Xavier initialization as one may see in figure 2. This figure will be important later as a comparison when the weights will have been updated and theirs distributions will have changed.

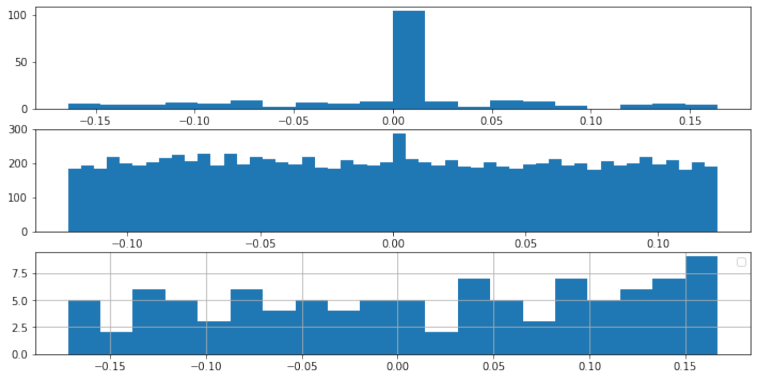

Figure 2: Initial weights distribution between the four layers

Therefore, the network has to be trained and this was done using the training set.
First one has to set the hyper-parameters that work better. There could have been many of them:

- Activation function of the neurons
- Number of neurons of the first layer (Nh1)
- Number of neurons of the second layer (Nh2)
- Number of epochs
- Learning rate

## Activation function

It was decided to try 5 different activation functions:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Tanh: $\sigma(x) = tanh(x)$
- ReLU: $\sigma(x) = max(0, x)$
- Leaky ReLU: $\sigma(x) = max(0.1 \cdot x, x)$
- ELU: $\sigma(x) = \begin{cases} 0.5 \cdot (e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$

The parameter 0.5 for the ELU function was chosen arbitrarily.
In figure the activation functions are plotted together with their derivatives.

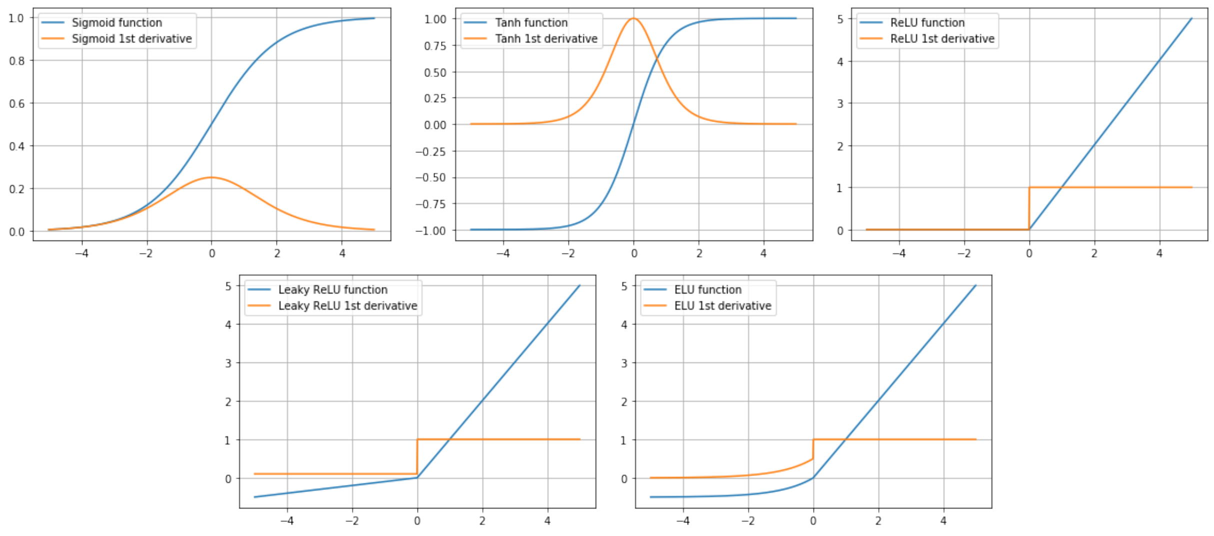

Figure 3: Activation functions

## Random search 3-fold cross validation

For what concerns the choice of the activation function and the number of neurons, in the first attempt it was implemented a Grid search but, since it was supposed that the choice of the activation function would not be so influential in the final error, it was preferred to use a random search, in order to span as widely as possible the plane where the other two hyper-parameters reside.

So it is decided that the range where the number of neurons of the two hidden layers must be draw out. So at every step the program samples randomly two number from that range and an activation function. Then it does all the computation with these hyper-parameters fixed.
The scope of the random search is to find the best set of hyper-parameters between the tried ones.

In order to do this in a robust way it is implemented a 3-fold cross validation, it means that the train set is divided in three equal folds and the network with fixed hyper-parameters is trained over two of the three folds per time. The remaining fold is used as a validation set.
At every epochs the weights are update minimizing the error over each point of the two folds chosen for training and then the network makes the prediction over the third fold (the validation one), finally the error of these predictions is calculated. This procedure is iterated till the early stopping (that will be described later) is satisfied. Once the stopping condition is reached, the minimum error is retrieved from the list containing all the errors on the validation set. Also the number of epochs done is saved. Now all this stuff is repeated exchanging the folds that will be the training set and the validation set for the three times requested.

Therefore, finally three minimum validation errors and three max number of epochs are obtained. The three errors are averaged, instead between the three number of epochs the greater one is selected.

Thus, since this work is done for all the chosen set of hyper-parameters the average error is the fundamental result of all this computation, in fact is the number that allows to compare them. The best set is the one that has the lower average error among the three folds.
Also a L2-regularization factor was firstly included but it turned out to perform worse.

### Early stopping

Now it is important to analyze the fourth parameter of the list, i.e. the number of epochs of training for each training set. In fact if this number is too high the network will overfit the data, because, since a neural network can, in principle, implement any functions, if it is left training for too long it will adapt its weight in order to fit perfectly the training sample and as a consequence very badly the test ones.
The best thing to do, instead of choosing a arbitrary maximum number of epochs, is to implement an algorithm that is able to stop the network from training if it realizes that the net is starting to overfit the data.
The selected algorithm (Lutz Prechelt, *Early Stopping | but when?*, Fakultat fur Informatik; Universitat Karlsruhe D-76128 Karlsruhe; Germany, 1998) is based on the generalization error, i.e. the validation error. It stops the algorithm when the validation error at the last epoch is $\alpha$ times greater then the minimum error found till that epoch. In formulas:

$$GL_{\alpha}(t) = \frac{E(t)}{E_{min}} - 1 > \alpha$$

The parameter $\alpha$ represents the minimum percentage of the minimum error that is not allow to exceed. It is chosen accordingly to the necessity, the greater the value, the greater the goodness of the found minimum but the stop condition will arise later.

### ADAMS

If one wants to use different activation functions the choice of the correct learning rate could be very tricky. Therefore, Instead of using a fixed learning rate, or a decaying one it was used an algorithm that can change adaptively. ADAMS is a good solution and it is implemented inside the class:

```python
def ADAMS(self, W, g, m, s, t, b1=0.9, b2=0.99, e=10**(-8), lr=10**(-3)):
m = b1*m + (1-b1)*g
s = b2*s + (1-b2)*g**2
m_avg = m/(1-b1**(t+1))
s_avg = s/(1-b2**(t+1))
W = W - lr*m_avg/((s_avg)**(1/2)+e)

return W, g, m, s
```

where W represents the weights to be updated and the parameters m, s are set initially equal to zero as global variables. The variable g represents the gradient.
This algorithm allows to have better results in finding the right solution in a less time because it considers also the curvature of the loss function.
Moreover also the decay learning rate was used in combinations with ADAMS because it gave better results.

## Results

Many random search were performed here the results of one of those are shown.
The network was trained over 15 different sets: Figure 4 shows the results for the random sets selected (here the activation functions are not considered).

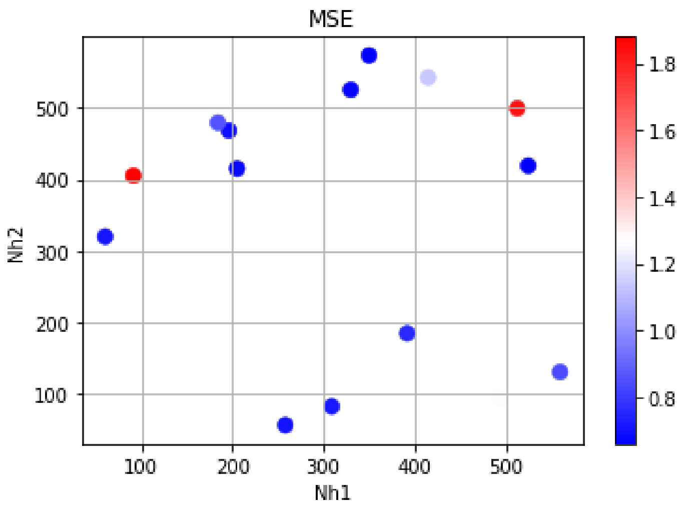

Figure 4: Each point is a couple of numbers of hidden neurons and its color represents the average error over the three validation sets

The best results in this random search was obtained using:

- Activation function: Leaky ReLU
- Number of neurons of the first layer: 350
- Number of neurons of the second layer: 573

The network with these parameters was trained over all the training point without leaving anyone as validation set. The training was stopped using the maximum number of epochs, between the three training, saved before.
Then the net is used to perform the prediction over the test set (that has never been used before), that represents the goodness of the found result.
The test loss is: 0.577

Figure 5 shows the fitting function over both training and test set.

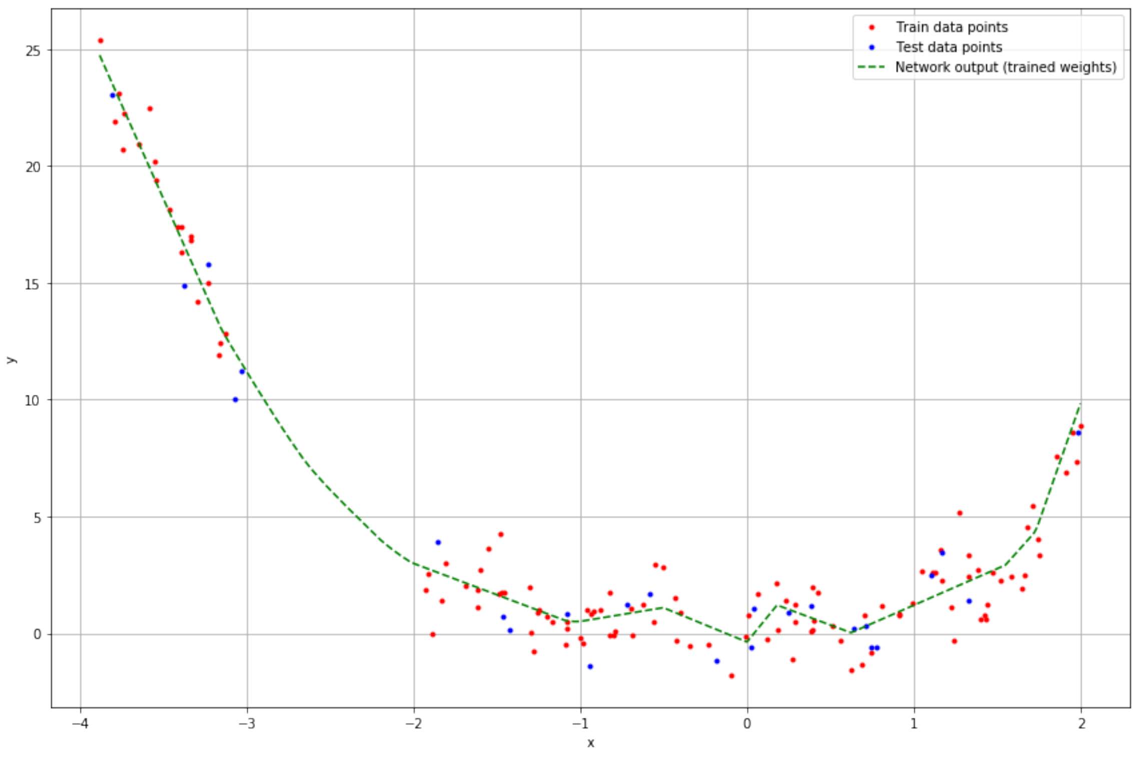

Figure 5: Training, test set and function

The distribution of the weights now is very sharped around 0 as one may see in Figure 6. This could be compared with the initial distributions that were very broad, almost uniform (Figure 2).

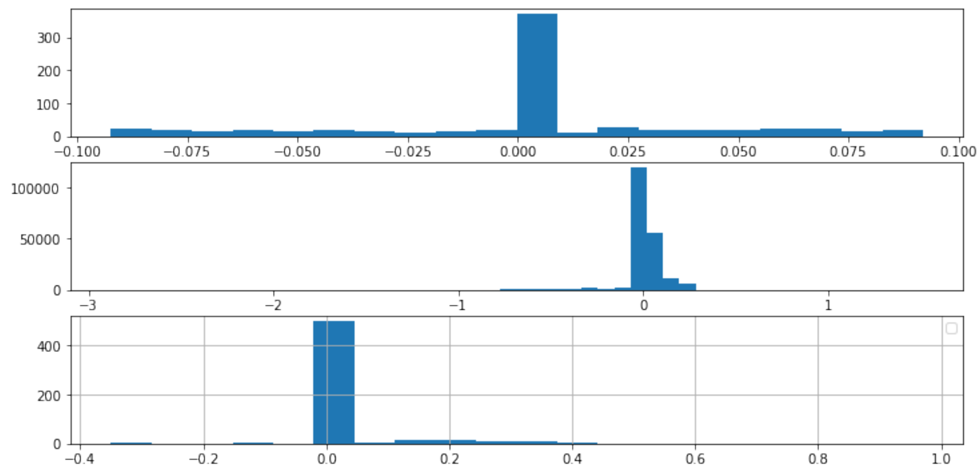

Figure 6: Final weights distribution between the four layers

Finally Figure 7 shows the activation of the neurons in the last layer.

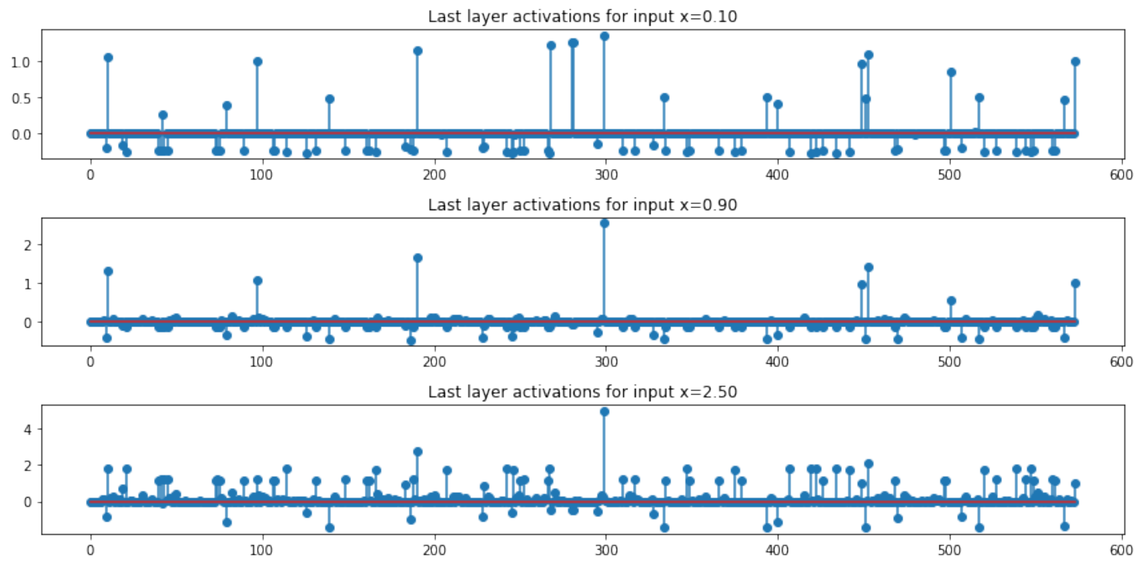

Figure 7: analyze activations

This was the first good result, but the numbers of neurons chosen was supposed to be too big. Why? Because the distribution of the weights is very sharp around zero (Figure 6): this means that the main part of the weights is useless and its contribution is almost zero. Furthermore the activation of the neurons (Figure 7) shows that there are few neurons that spikes differently for different initial conditions, but there are a lot of them that remain around zero.
This is probably due to the fact that there are too many neurons and many of them become useless. So it was tried to used a different range of numbers (between 10 and 100).
The results are better:

- Activation function: tanh
- Number of neurons of the first layer: 87
- Number of neurons of the second layer: 31

The test loss is: 0.5494

Figure 8 shows the trend of the errors.

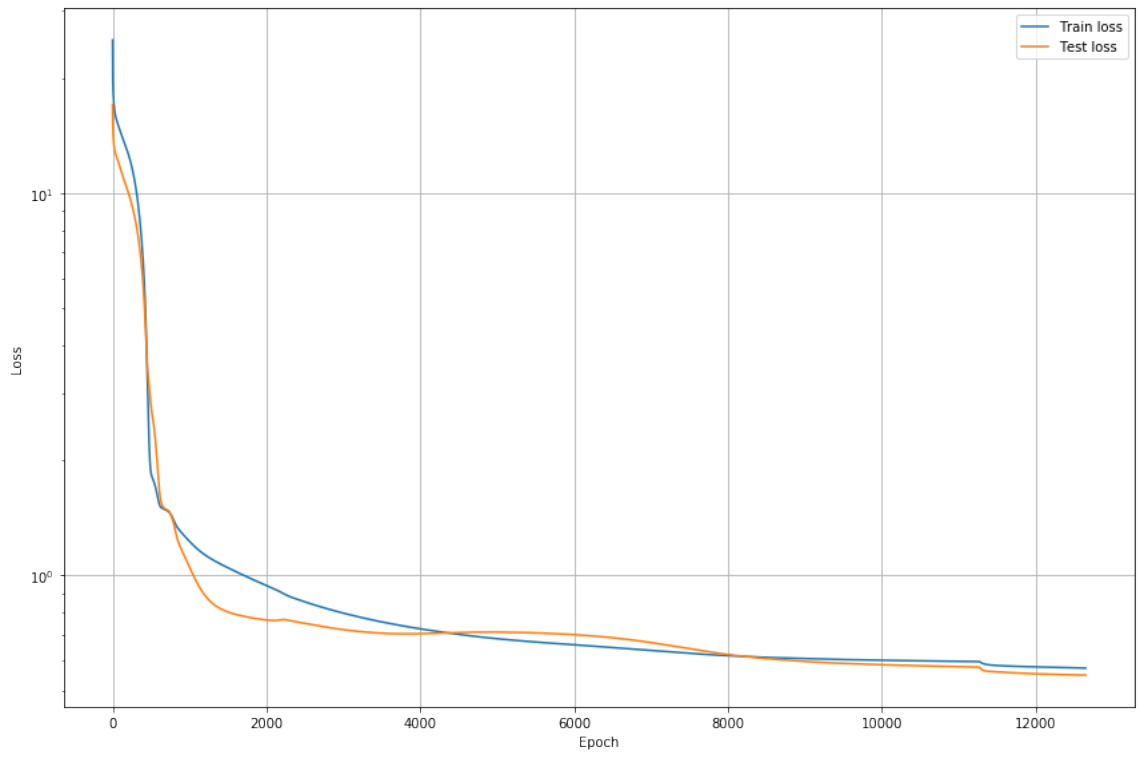

Figure 8: analyze activations

Figure 9 shows the fitting function over both training and test set.

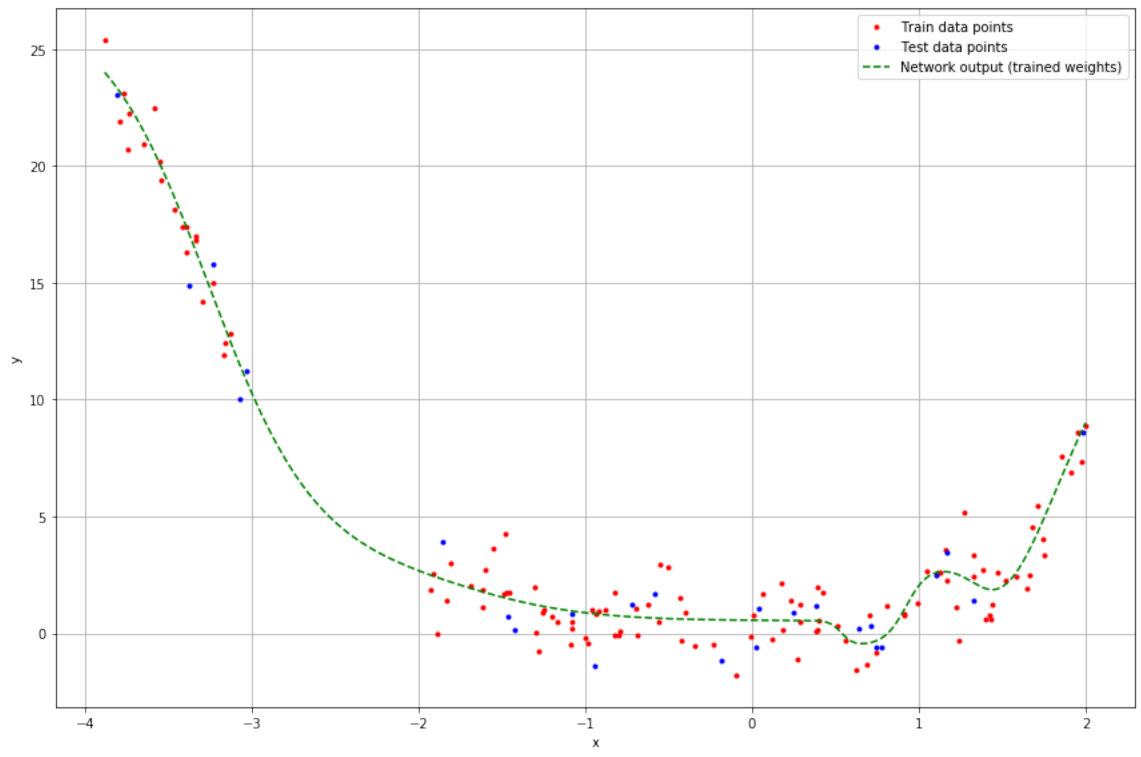

Figure 9: Training, test set and function

Now the distribution of the weights is not more very sharped around 0 but is more broad Figure 10. This could be compared with the initial distributions of the previous trained model.

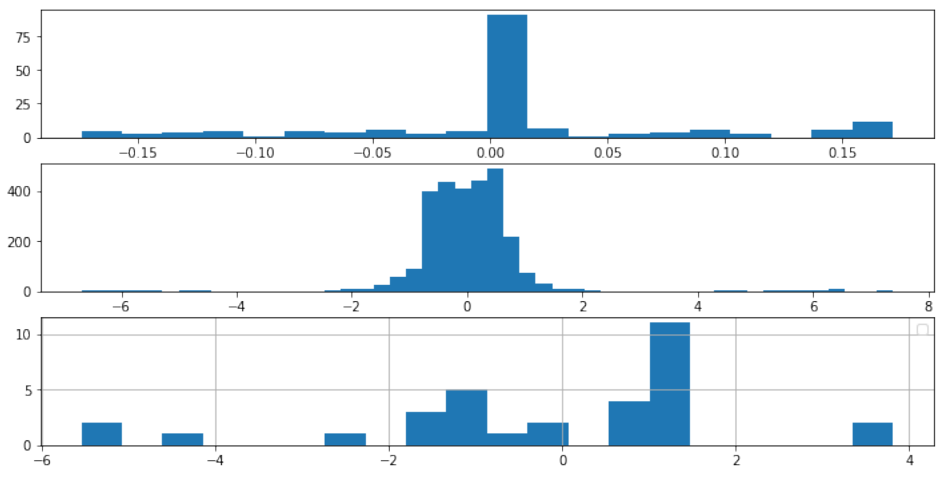

Figure 10: Final weights distribution between the four layers

Finally Figure 11 shows the activation of the neurons in the last layer.

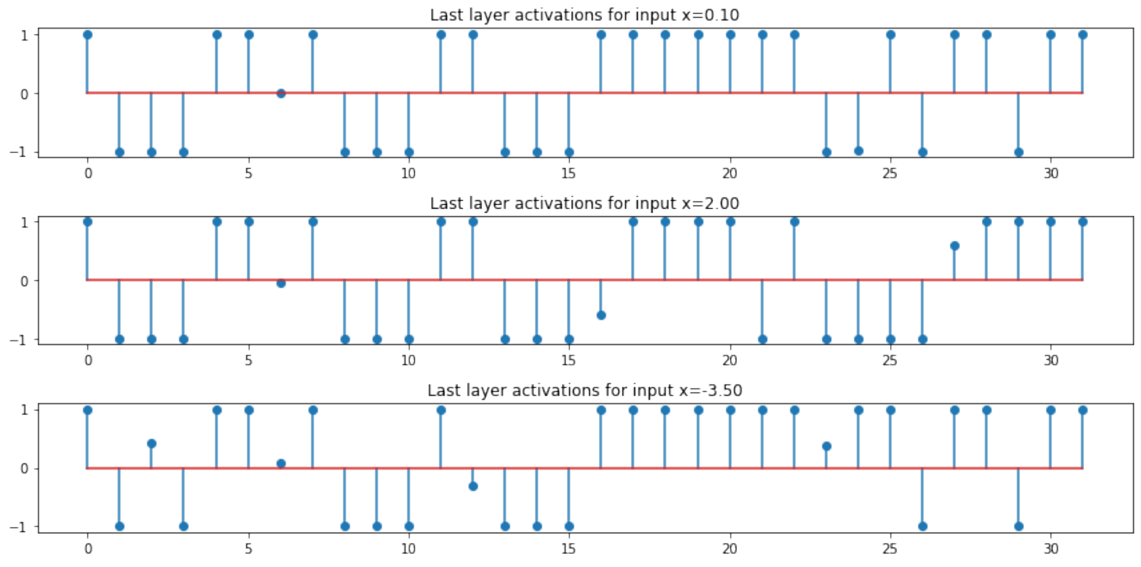

Figure 11: analyze activations

From Figure 11 one can see that every neuron does its job and none but one of them is zero, so everyone codes a bit of information. When changing the input some neurons change specially when different zones of the domain are considered.

## Discussion

Let us discuss the results in the light of the choices made. The cross validation is very useful to prevent that the parameters will depend on the particular validation set. It makes the decision more robust because it will depend on three different validation sets. It was chosen to use only three sets because a greater number will seriously increase the time of computation.
The early stopping condition is fundamental to prevent the overfitting and it is based on the validation set. So, when the training over the all training set is performed (and there is no validation set to use since the test set is untouchable till the very end) one needs a good number of epochs to use. This is chosen as the maximum number of epochs found in the three validation set.
The algorithm for updating the learning rate, i.e. ADAMS, was chosen because it was noticed that using a simple decaying learning rate, in procedure of selection of the parameters, many of them present a very big error (around 30). This was a signal that the network was underfitting. It was not learning almost anything for many of the parameters chosen. Therefore in order to equalize the results from all the sets and comparing them rightfully a good choice of learning rate is necessary. ADAMS solves this problem and let

the algorithm find a good minimum of the loss function and let all the errors be similar as one may see in Figure 4.

The differences between the two best models are very large. The second one seems to have all the characteristics to be more generalizable and better prevent the overfitting. However the first one shows a sinusoidal trend in the middle whereas the second one seems not able to capture this feature. On the other hand the second one shows an oscillatory behavior near the right edge of the domain, where the first model is just a combinations of two linear function. Other attempts have shown that also the first model with a larger number of epochs is able to have an step also in the right edge but this ended with a larger error on the test set.
The second model seems to be a continuous function, the first one a sum of linear functions.
What is the right model? Nobody knows. The considerations of oscillatory behaviors are only subjective. The only things one can use to understand which model is better is are the numerical error and the weights distributions graphs. They both indicate a strong preference for the second model but only a larger number of data will allows to pronounce the real sentence.