Homework 3

Ana Santos, 84364 Inês Branco, 81506 Vânia Nunes, 85235

April 2019

The link to the code: https://drive.google.com/file/d/1vqvNrZs5v-PNK9hwzZ-2ShSvCO08gKSh/view?usp=sharing

1 Problem 1

In the following text we considered the following notation for the sub-questions:
a)linear unit with quadratic error as loss function
b)sigmoid unit with quadratic error as loss function
c)sigmoid unit with cross entropy as loss function

As required in the problem, we used the gradient descent algorithm to train our data. The input corresponds to the 4 features of the iris flower: Sepal Length, Sepal Width, Petal Length and Petal Width. The present purpose the Irish-flower data set' training is to be able to classify a flower in the category Iris-Virginica (output=1) or non Iris-Virginica (output=1). In total, we have 150 input vectors. There were 3 cases of combinations of activation and loss functions First, it was computed the net for each case:

$$net = \sum_{j=1}^{N} w_j x_{kj} \tag{1}$$

For both cases a) and b), it was computed a quadratic error as loss function:

$$E = \frac{1}{2}\sum_{k=1}^{N}(t_k - o_k)^2, \qquad o_k = \sigma\left(\sum w_i x_i\right) \tag{2}$$

For c), a cross entropy error was used:

$$E = -\sum\left(t \ln o + (1-t)\ln(1-o)\right), \quad o = \sigma\left(\sum w_i x_i\right) \tag{3}$$

A loop was then performed in a number n of epochs. For each epoch, another loop over each point of the data set used for training and test was executed. The expected output for each point was computed, o_k, using the corresponding activation function for each case:

$$\sigma(x) = 1, \qquad for a) \tag{4}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}, \qquad for\ b)\ and\ c) \tag{5}$$

To update the weights, we used the following equations:

$$w^{new} = w^{old} + \Delta w, \quad \Delta w = -\eta \frac{\partial E}{\partial w_i} \tag{6}$$

where $\frac{\partial E}{\partial w_i}$ takes the form:

$$\frac{\partial E}{\partial w_i} = -\sum (t - \sigma(net))\sigma(net)(1-\sigma(net))x_i, \qquad for\ a)\ and\ b) \tag{7}$$

$$\frac{\partial E}{\partial w_i} = -\sum\left(\frac{t}{\sigma(net)} - \frac{1-t}{1-\sigma(net)}\right)\sigma(net)(1-\sigma(net))x for\ c) \tag{8}$$

In each loop over the data set, it is found a value of Δw to update the weights. Over the various epochs, various values of Δw are computed in order to update the vector w_k for each point.

In order to study the convergence rate, we plotted the variation of the error function in function of initialization parameters such as the learning parameter, η, the initialization vector of weight, and the number of epochs. The results are shown hereafter.

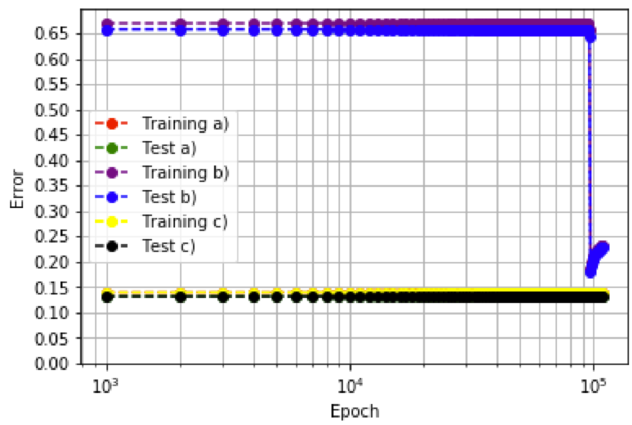

Figura 1: Plot of the variation of the average error (over all points of the data set) with the number of epochs.

Observing the plot, we realize that for both cases a) and c) the error has already converged in the epoch number 1000, whereas for the b) case only between the epoch number 9×10^4 and 10^5 the error achieves a minimum. In order to find out which method has the best convergence rate, we have repeated the plot in figure 1 for the cases a) and c), with a lower range of the value of epoch (see the code). They both converge approximately at an epoch of 100.
We conclude therefore that the convergence rate for a sigmoid unit with quadratic error is much slower than for the other two cases.

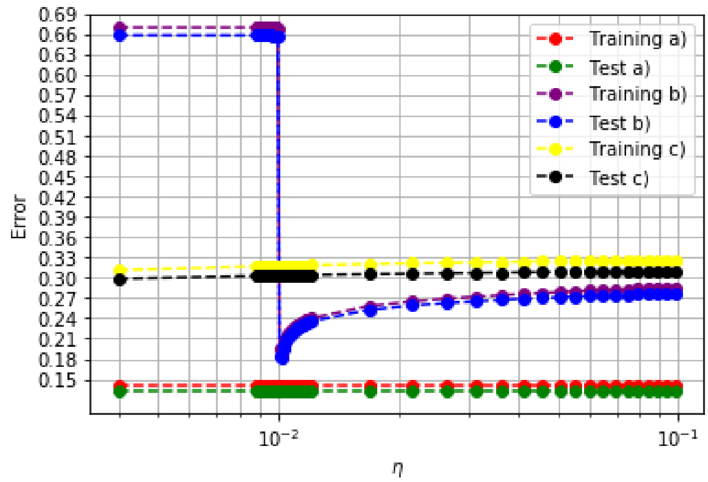

Figura 2: Plot of the variation of the average error (over all points of the data set) with the learning rate, η. Epochs=100000

Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The lower the value, the slower we travel along the downward slope. If the learning parameter is too high, the gradient descent can overshoot the minimum and fail to converge. It's observable that for the cases a) and c) the error appears to remains almost constant for the values of η presented, whereas for b) a minimum is achieved for a value of η close but bigger than 0.01. From this point on, the error continues to increase until it levels to values close to 0.30.

Once again, to find a minimal value for the learning parameter, we re-plotted the figure 2, for only the cases a) and c) (see code). With a better resolution, we observe that, for c), the error is minimum in 0.009, whereas for a) is not observable.

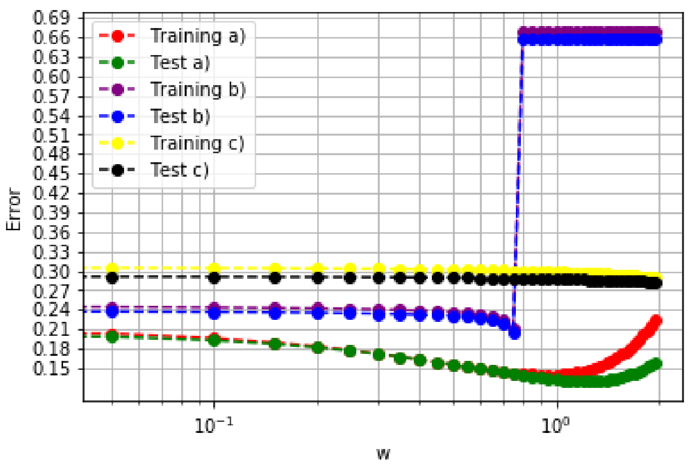

Figura 3: Plot of the variation of the average error (over all points of the data set) with epoch

For testing the impact of the initial weight vector, we studied the error variation with this parameter. The weight vector was initialized with all its elements with the same values. In the above plot, we see that in the case b), there is a convergence of the error for an initial weight of 7.5×10^{-1}; in the case a), the minimum occurs in 1; and in case c), the error is minimum in 0 (see extra plots in the end of code). For high values of epochs used, for the cases a) and c) the final values of the weight is independent of initial values of the weight and learning parameter. We've therefore computed, this final weights using less number of epochs (see plots in the end of the code).
Having already in mind the optimal parameters already found, for each method, it was computed the test and training error:
a) final error train= [0.14518316] final error test= 0.130
final error train= 0.132 b)final error train= 0.266 final error train= 0.259
c)final error train= 0.262 final error test= 0.253
The best method revealed to be the one for case a). In addition it must be refered that the observed behaviour of training and test data was similar, which indicates there was not over-fitting of the results, and the trained algorithm was was well applied to a generalized test data. Since one third of the output values are one and two thirds are 0, when the error function of b) is 0.66 it means that the output of the trained neuron is always 1, so the weights are all positive. The same argument can be made for 0.33, the output is always 0, the weigts are all negative

2 Problem 2

To process the data, we decided to convert all the input parameters into numbers from 0 to one. We also saw that some of the data was missing, in total 9 data points out of 277 had some information missing, so we decided to delete them. We divided the data into testing and training data about 75% of data went into training and the rest into testing.

For all the graphs plotted we used one hidden layer, with linear activation function and 2 inputs(similar to the class). We used 32 batch points and either 500 or 1000 epochs depending of how long it took to have smaller losses.

We tested this neural networks for the three activation functions (sigmoid, linear and relu), for the two regularization (with the weights squared or with their absolute value), and for four different values for the regularization parameter (0.001,0.01,0.1,1).

We made sure that all other initial parameters such as the initial weights were the same for all the plots.

By looking at the graphs (in the code) we can see that using the mean square loss function, the neural networks that use the leacky relu function converge the faster. For higher values of the regularization parameter the convergence seems to be faster for all functions (either activation, loss or regularization).

By looking at the graphs (in the code) we can see that using the cross entropy loss function, the convergence seems to be slower for all activation functions, and seems to be better for the sigmoid activation function (if the regularization uses the aif the regularization uses the squares). Once again the higher the regularization parameter seem to be better.

The values for the test and training error don't diverge greatly so we assume that there wasn't over-fitting.

Since the mean-square error works better for linear and relu functions and the cross entropy for sigmoid function, the results are what was expected.