Instituto Superior Técnico

Departamento de Engenharia Electrotécnica e de Computadores

Machine Learning

3rd Lab Assignment

Shift: *Sexta 14h*	Group Number: *1*
Number 84053	Name Francisco Raposo de Melo
Number 89213	Name Rodrigo Tavares Rego

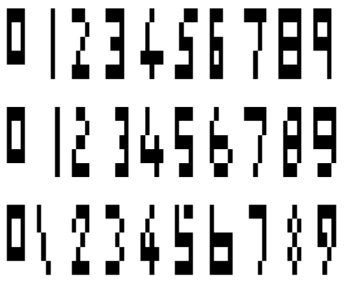
Multilayer perceptrons

This assignment aims at illustrating the applications of neural networks. In the first part we'll train a multilayer perceptron (MLP) for classification and in the second part we will train a MLP for regression.

This assignment requires MatLab's Neural Network Toolbox.

1 Classification

Our classification problem is a pattern recognition one, using supervised learning. Our goal is to classify binary images of the digits from 0 to 9, with 5x5 pixels each. The following figure illustrates some of the digits.



1.1 Data

Data are organized into two matrices, the input matrix X and the target matrix T.

Each column of the input matrix represents a different pattern or digit and will therefore have 25 elements corresponding to the 25 pixels in each image. There are 560 digits in our data set.

Each corresponding column of the target matrix will have 10 elements, with the component that corresponds to the pattern's class equal to 1, and all the other components equal to -1.

Load the data and view the size of inputs X and targets T.

```
load digits
size(X)
size(T)
```

Visualize some of the digits using the function show_digit which was provided.

1.2. Neural Network

We will use a feedforward network with one hidden layer with 15 units. Network training will be performed using the gradient method in batch mode. The cost function will be the mean squared error,

$$C = \frac{1}{KP}\sum_{k=1}^{K}\sum_{i=1}^{P}\left(e_i^k\right)^2,$$

where K is the number of training patterns, P is the number of output components, and e_i^k is the ith component of the output error corresponding to the kth pattern.

```
net = patternnet([15]);
net.performFcn='mse';
```

Both the hidden and the output layer should use the hyperbolic tangent as activation function.

```
net.layers{1}.transferFcn='tansig';
net.layers{2}.transferFcn='tansig';
```

We will use the first 400 patterns for training the neural network and the remaining 160 for testing.

```
net.divideFcn='divideind';
net.divideParam.trainInd=1:400;
net.divideParam.testInd=401:560;
```

1.3. Gradient method with fixed step size parameter and momentum

Train the network using Gradient descent with momentum backpropagation. Initially, set the learning rate to 0.1 and the momentum parameter to 0.9. Choose, as stopping criterion, the cost function reaching a value below 0.05 or the number of iterations reaching 10000.

```
net.trainFcn = 'traingdm'
net.trainParam.lr=0.1; % learning rate
net.trainParam.mc=0.9;% Momentum constant
net.trainParam.show=10000; % # of epochs in display
net.trainParam.epochs=10000;% max epochs
net.trainParam.goal=0.05; % training goal
[net,tr] = train(net,X,T);
```

To see the evolution of the cost function (MSE) during training, click the "Performance" button.

Try to find a parameter set (step size and momentum) that approximately minimizes the number of training epochs of the network. Indicate the values that you obtained.

Step size: *4,25*	Momentum: *0,5*

Determine how many epochs it takes for the desired minimum error to be reached (execute at least five tests and compute the median of the numbers of epochs).

Median of the numbers of epochs: *51*
42, 47, **51**, *55, 66*

1.4. Gradient method with adaptive step sizes and momentum

Next, choose as training method, gradient descent with momentum and adaptive learning rate backpropagation.

```
net.trainFcn = 'traingdx'
```

Train the network using the same initial values of the step size and momentum parameters as before. How many epochs are required to reach the desired minimum error? Make at least five tests and compute the median of the numbers of epochs.

Median of the numbers of epochs: *109*

105, 107, 109*, 113, 117*

Try to approximately find the set of parameters (initial step size and momentum) which minimizes the number of training epochs. Indicate the results that you have obtained (parameter values and median of the numbers of epochs for training). Comment on the sensitivity of the number of training epochs with respect to variations in the parameters, in comparison with the use of a fixed step size parameter. Indicate the main results that led you to your conclusions.

Initial Step Size: *1,25* | ***Momentum:*** *0,8*

Median of the number of epochs: *53*

47, 47, 53*, 66, 78*

With respect to the variations in the parameters, the training method Gradient Descent with ***Fixed Step Size*** *and Momentum is* ***more sensitive*** *to those variations, as we can infer from the tables below (where we fix one of the parameters and increment the other for 3 cases). This happens because the initial step size with the adaptive step size method weights less in the algorithm since the method will change its value gradually in order to converge faster (a little bit regardless of how bad is the initial step size).*

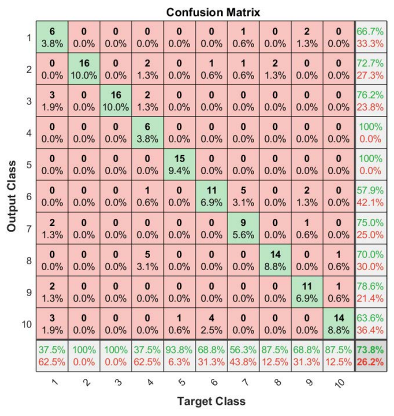
Momentum	**Learning Rate**		
0.9	0.1	0.5	1
Fixed Step Size	2405	402	293
Adaptive Step Size	105	79	62

Learning Rate	**Momentum**		
0.1	0.1	0.5	0.9
Fixed Step Size	2097	1712	2426
Adaptive Step Size	181	125	59

Next, we'll analyze the classification quality in the test set with a confusion matrix. This matrix has 11 rows and 11 columns. The first 10 columns correspond to the target values. The first 10 rows correspond to the classifications assigned by the neural network. Each column of this 10×10 submatrix indicates how the patterns from one class were classified by the network. In the best case (if the network reaches 100% correct classification) this submatrix will be diagonal. The last row of the matrix indicates the percentage of patterns of each class that were correctly classified by the network. The global percentage of correctly classified patterns is at the bottom right cell.

```
x_test=X(:,tr.testInd);
t_test=T(:,tr.testInd);
y_test = net(x_test);
plotconfusion(t_test,y_test);
```

Comment on the values in the confusion matrix you obtained. Is the accuracy the same for every digit? Are the confusions between the different digits what you'd expect?



Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	6 3.8%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.6%	0 0.0%	2 1.3%	0 0.0%	66.7% 33.3%
2	0 0.0%	16 10.0%	0 0.0%	2 1.3%	0 0.0%	1 0.6%	1 0.6%	2 1.3%	0 0.0%	0 0.0%	72.7% 27.3%
3	3 1.9%	0 0.0%	16 10.0%	2 1.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	76.2% 23.8%
4	0 0.0%	0 0.0%	0 0.0%	6 3.8%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	15 9.4%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
6	0 0.0%	0 0.0%	0 0.0%	1 0.6%	0 0.0%	11 6.9%	5 3.1%	0 0.0%	2 1.3%	0 0.0%	57.9% 42.1%
7	2 1.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	9 5.6%	0 0.0%	1 0.6%	0 0.0%	75.0% 25.0%
8	0 0.0%	0 0.0%	0 0.0%	5 3.1%	0 0.0%	0 0.0%	0 0.0%	14 8.8%	0 0.0%	1 0.6%	70.0% 30.0%
9	2 1.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	11 6.9%	1 0.6%	78.6% 21.4%
10	3 1.9%	0 0.0%	0 0.0%	0 0.0%	1 0.6%	4 2.5%	0 0.0%	0 0.0%	0 0.0%	14 8.8%	63.6% 36.4%
	37.5% 62.5%	100% 0.0%	100% 0.0%	37.5% 62.5%	93.8% 6.3%	68.8% 31.3%	56.3% 43.8%	87.5% 12.5%	68.8% 31.3%	87.5% 12.5%	**73.8% 26.2%**

Analysing the confusion matrix, for each class the most frequent output was the correct digit/class. The global percentage of correctly classified patterns is usually higher than 70%.The accuracy is different for every digit.

The confusions between the different digits were expected, because there are digits with very similiar patterns, for example 8 with 0, 9 with 5 and 1 with 7.

Write down the performance values that you obtained:

Training error (mse): 0,0499	Train set Accuracy: 83,3%
Testing error (mse): 0,0579	Test set Accuracy: 76,9%

Which quantity (mean squared error, or global percentage of correct classifications) is best to evaluate the quality of networks, after training, in this problem? Why?

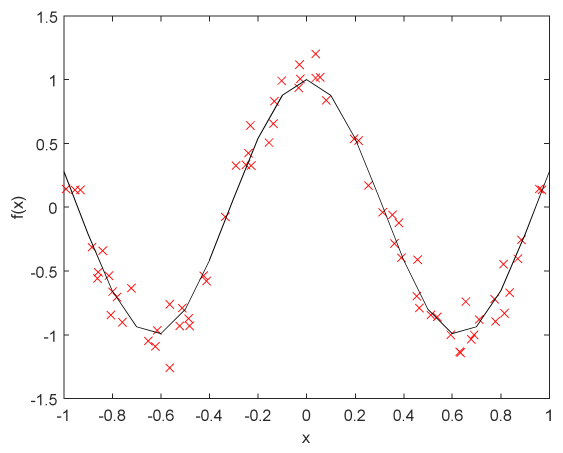
The quality of networks is best evaluated with the global percentage of correct classifications.

The training method of Gradient Descent will try to minimize the MSE until a certain goal is reached with the training set. The MSE of the testing error will accordingly be minimized, however it will end up a little deviated from the original goal. The only way of knowing how much that deviation impacts the classification is by checking the global percentage of correct classifications of the testing set (unbiased), this is why it is the best quantity to evaluate the quality of networks.

2. Regression

The goal of this second part is to estimate a function and to illustrate the use of a validation set.

The function is $f(x) = \cos(5x)$, with $x \in [-1,1]$ for which we have a small number of noisy observations $d = f(x) + \varepsilon$, in which ε is Gaussian noise with a standard deviation of 0.1. The values of x will be used as inputs to the network, and the values of d will be used as targets. The following figure shows the function $f(x)$ ($black\ line$) and the data we will use for training (*red crosses*).



2.1 Data

Load the data in file 'regression_data.mat' and check the size of inputs X and targets T.

2.2 Neural Network

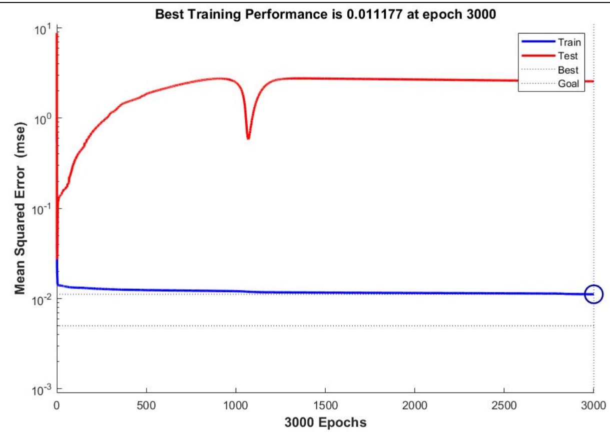
Create a network with a single hidden layer with 20 units. Set the activation of the output layer to linear (in the output layer, the 'tanh' function is more appropriate for classification problems, and the linear function is more appropriate for regression ones). Choose gradient descent using the Levenberg-Marquardt method.

```
net = fitnet(20);
net.layers{2}.transferFcn='purelin';
net.trainFcn = 'trainlm';
```

Choose 'mse' as cost function and, as stopping criterion, the cost function reaching a value below 0.005 or the number of iterations reaching 3000.

2.3 Training without a validation set

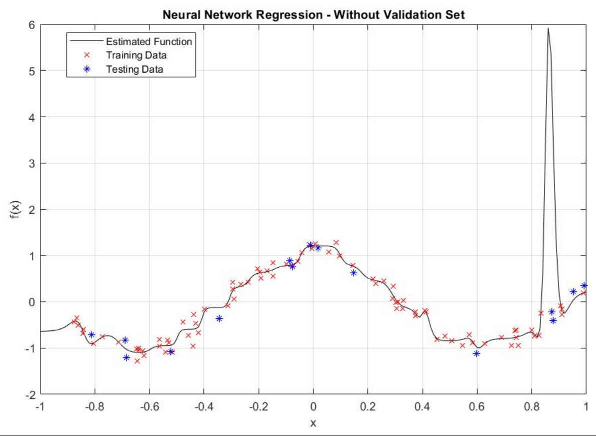
Train the network using the first 85 patterns for training and the last 15 for testing. Click the "Performance" button. Observe the evolution of the cost function for the different sets and comment.



The MSE of the train set doesn't reach the goal in 3000 iterations, yet it decreases. The MSE of the test set decreases abruptly in the begining but then rises and stays apart from the tendency of the training set.

Since the network is biased by the training data and trained with overfitting, the MSE of the test set is likely to stay apart from the goal.

Obtain the estimated function for x between -1 and 1 with a step of 0.01. the Plot the training data, the test data and the estimated function, all in the same figure, and comment.



The estimated function is overfitted, it tries to fit accordingly to all points in the training data. The neural network is trained to overfit all the training data.

It should also be considered that the training data has additive noise of standard deviation 0,1.

2.4 Training with a validation set

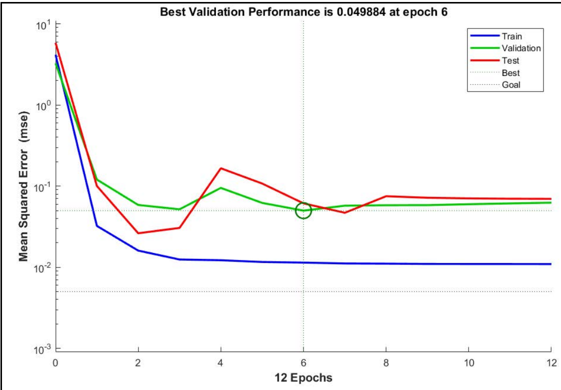
(T) When performing training with a validation set, how are the weights that correspond to the result of the training process chosen? What is the goal?

The goal of performing the training of the network with a validation set is to stop the training before it starts overfitting data and to reduce the bias given by the training data, thus generalizing more the regression.

The validation dataset works as a hybrid: it is training data used by testing, but neither as part of the low-level training nor as part of the final testing.

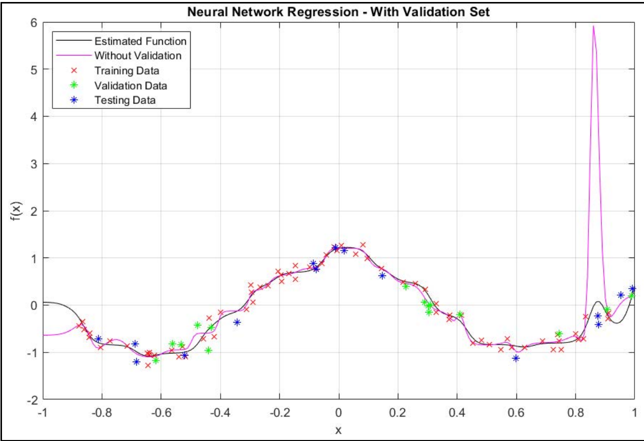
The validation set will work as a monitor of the weights and will stop the training in order to prevent overfitting choosing the weights at the previous iteration before the validation checks fail. The validation checks fail when the mean squarred error is higher than the last time it was checked, therefore this is the stopping criteria for the training of the network.

Train the network using the first 70 patterns for training, the next 15 for validation and the last 15 for testing. Click the "Performance" button. Comment on what is shown in the plot.



The training stopped when the validation increased for 6 iterations (occurred at iteration 12 - 6 validation checks). The result at epoch 6 is reasonable because the final MSE is small, the test set error and the validation set error have similar characteristics and no significant overfitting has occurred by iteration 6, where the last validation performance occurs. Therefore, the weights correspond to the iteration 6.

Obtain the newly estimated function and plot it in the same figure as the function obtained in 2.3. Comment.

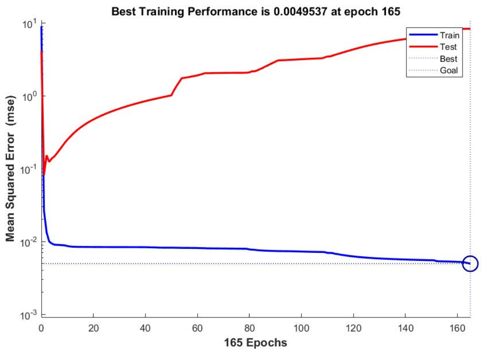


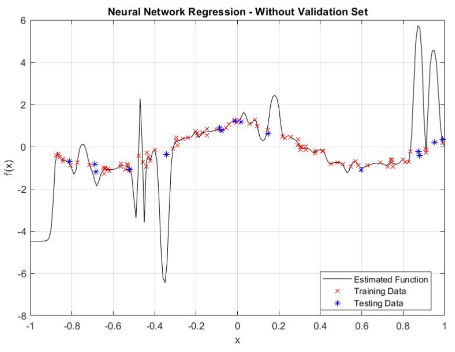
The validation set lessened the network bias by the training set, monitoring the weights of the network. One of the objectives of a validation set, regarding regression, is to diminish the overfitting of data.

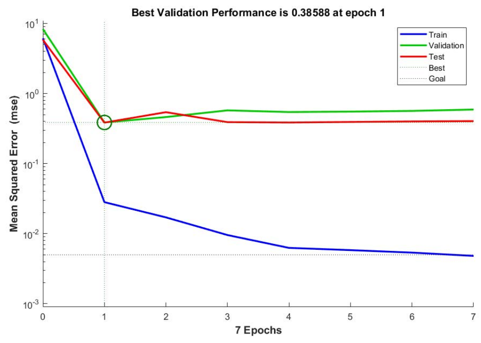
Observing the plot, there's still some overfitting, however, it is less influenced by the training data. It should also be considered that the training, validation and test sets are too small (to save computational effort).

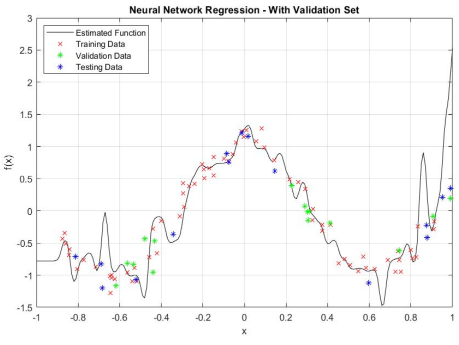
2.5 Effect of the number of units

Repeat 2.3 and 2.4 but now use 50 units in the hidden layer. Plot the functions obtained with and without validation and comment on the results, in comparison with the previous ones.









In comparison to the previous results, the increased number of units in the hidden layer allowed the training to reach the MSE goal, in the case of without validation set, resulting in a higher overfitting of the data.

In general, for both cases the model gets more adaptive, so it can learn smaller details. As a result, the network got more affected by overfitting, decreasing the generalization of the regression.

Since we have a relatively small training set, and a noisy one, it's preferable to keep the number of units in the hidden layer low in order to avoid point-to-point mapping, which is likely to overfit or pick up noise.