Instituto Superior Técnico

Departamento de Engenharia Electrotécnica e de Computadores

Machine Learning

3rd Lab Assignment

Shift: Thursday Group Number: 5

Number 80966 Name Francisco Manuel Neves Moreira de Azevedo

Number 81356 Name Duarte Miguel Ferreira Dias

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

(T) Describe how the minimization of a function through the gradient method, with fixed step size parameter and with momentum, is performed. Be precise. Use equations when appropriate.

This is the algorithm presented in the lecture slides that minimizes the cost with a gradient method with fixed step size and momentum.

Set $t = 1$ and $\Delta w_{ij}(0) = 0$. Repeat step 1 until stopping criteria is met

1. For $k = 1, \ldots, n$, perform steps 1.1 through 1.6
   - 1.1 propagate forward: apply the training pattern x^k to the perceptron and compute the variables z_i and outputs $\hat{y}^k$
   - 1.2 compute the cost derivatives: $\frac{\partial L^k}{\partial \hat{y}_j^k}$
   - 1.3 propagate backwards: apply $\frac{\partial L^k}{\partial \hat{y}_j^k}$ to the inputs of backpropagation network and compute its internal variables ϵ_j
   - 1.4 compute the gradient components: compute the variables $\frac{\partial L^k}{\partial w_{ij}} = z_i \epsilon_j$
   - 1.5 Apply momentum: set $\Delta w_{ij}(t) = -\eta z_i \epsilon_j + \alpha \Delta w_{ij}(t-1)$
   - 1.6 Update the weights: set $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$

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 0.1 or the number of iteration 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=0.1; % 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 training time of the network. Indicate the values that you obtained.

Step size: 3 Momentum: 0.6 NOTA: Resultados para goal = 0.1

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: Median {45,46,62, 68, 77} = 62

NOTA: Resultados para goal = 0.1

1.4. Gradient method with adaptive step sizes and momentum

(T) Describe how the minimization of a function through the gradient method with adaptive step sizes and momentum is performed. Be precise. Use equations when appropriate.

The gradient method with adaptive step sizes and momentum is a twist to the algorithm already explained in 1.3. In this case the step size η (eta) is not fixed. So the algorithm is the same but there's a difference in the line:

1.5 Apply momentum: set $\Delta w_{ij}(t) = -\eta^{(t)} z_i \epsilon_j + \alpha \Delta w_{ij}(t-1)$

Given that η(t) is now an adaptive step. The step size update changes according to:

$$\eta_{ij}^{(t)} = \begin{cases} u\eta_{ij}^{(t-1)} & \text{if } \frac{\partial J}{\partial w_{ij}}\left(w^{(t)}\right) \cdot \frac{\partial J}{\partial w_{ij}}\left(w^{(t-1)}\right) > 0 \\ d\eta_{ij}^{(t-1)} & \text{otherwise} \end{cases}$$

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: Median {49,71,72,79,95} = 72

NOTA: Resultados para goal = 0.1

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.

We found that the parameters: step size = 3 and momentum = 0.8 grant a Median {28,36,44,44,91} = 44 which is approximately the set that minimizes the training epochs the most

When training the neural net with fixed step size = 3 and 'traingd' we noticed fluctuations of the number of epochs with results between ~60 epochs and others ~180 epochs whilst with adaptive step sizes the results were contained in a smaller intervall. (and momentum)

In terms of sensitivity to parameters the adaptive step sizes and momentum method was very sensitive given that the number of epochs changed significantly when we change the parameters slightly.
For example when using step size = 3 and momentum = 0.7 (instead of 0.8) the median of 5 sets of 5 tests (each set computes median of 5 tests, then the median of 5 sets) was 75 which is almost double of the number of epochs needed with step size = 3 and momentum = 0.8. We conclude that this method produces results that are very sensitive to its parameters.

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 errors what you'd expect?

In this comment we assume that the digit "0" is represented as "1" in the confusion matrix, "1" as "2" and so on untill "9" that is represented as "10".
We notice, analyzing column by column, that the digit "0" is classified as "1" 25% of the times and as a "9" 13% of the times.

The digit "8" is wrongly classified as a "6" 25% of the times and as a "9" with a chance of 13%.

We suspect that the digit "0" is missclassified as a "1" because a very thin "0" resembles a "1".
We postulate that due to the high similarity of the numbers "8" and "6" or "9" the neural has lower accuracy classifying the digit "8"

The errors are not the same for every digit and the neural network is highly accurate (>90%) guessing the digits "4" and "5" but are less accurate guessing the digits "0" and "8"

Write down the performance values that you obtained:

Training error: 0.0499 Test set Accuracy: 79.4% NOTA: Resultados para goal = 0.05

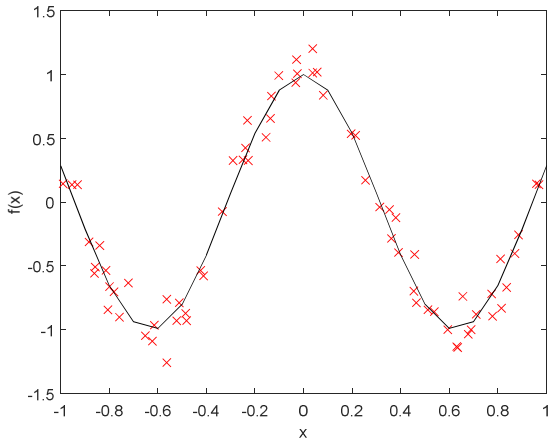
Which quantity (mean squared error, or global percentage of correct classifications) is best to evaluate the quality of networks, after training, in this problem? Why?

We think that the global percentage of correct classification is better to evaluate the network after training. The mean squared error (MSE) is best used while training to determine a stopping point. However when the training is over the MSE value is not as expressive as the global percentage. Having an MSE = 10 and then improving it to and MSE = 1 does not mean the accuracy will be 10x better. Although having a smaller MSE generally translates to better accuracy it is not a good metric to use to evaluate the quality of the network. The global percentage of correct classification translates the quality of the network better.

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 40 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).

```
net = fitnet(40);
net.layers{2}.transferFcn='purelin';
```

Choose 'mse' as cost function and, as stopping criterion, the cost function reaching a value below 0.005 or the number of iterations reaching 10000.

2.3 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?

A validation set is a set of examples used to tune the parameters of the network. In this case we use the validation set to determinate a stopping point for the algorithm.

The weights are chosen in order to minimize both the validation set Mean Squared Error (MSE) and the training set MSE. It's important for validation set MSE to decrease with the number of epochs to validate the parameters chosen.

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.

In this plot we can analyze the different performance regarding the training, validation and testing sets in function of the used cost function (mean squared error in this case).

The plot also indicates what is the best validation performance. In our case it was 0.037139 for the first iteration (epoch).

Plot the training data, the test data and the estimated function, all in the same figure and comment.

In this plot we can observe the elements from the set that were used, respectively, for training, validation and testing purposes. By comparing the estimated function to the data from the training set it is possible to calculate the obtained error deviation which is also represented in the obtained plot. For a more in depth study of the obtained errors there is a derived plot in the lower section of the output window where we can easily see the obtained error for each individual element from the training set.

2.4 Training without a validation set

Repeat the previous item but do not use a validation set this time. Observe the evolution of the cost function for the different sets and comment. Is there any sing of overfitting?

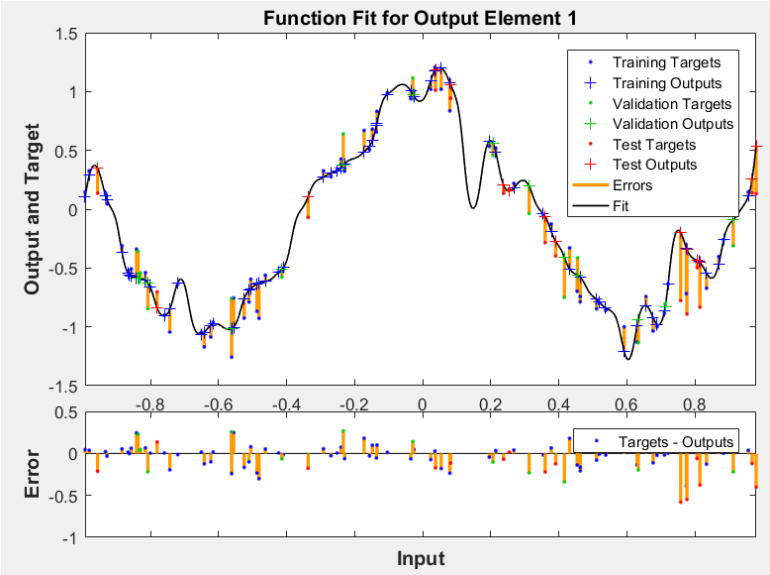
The neural network GUI (graphical user interface) from matlab does not allow us to set the validation set to 0% so a script was written to make this possible and to output the trained neural network and its parameters.

When comparing the obtained plot in this section to the one from section 2.3 we can rapidly observe that this new plot is a better fit than the one obtained with a validation set. There is a clear case of overfitting specially in the [0, 0.2] interval. In this case we have a better fit (given that the error is between [-0.4; 0.2] in this case and around [-0.6; 0.4] in 2.3). This means our fit is better (less error) but there's a tradeoff: it doesnt generalize well, i.e. its specific to the data provided. This is obvious due to the overfitting in [0; 0.2] interval in the X axis.

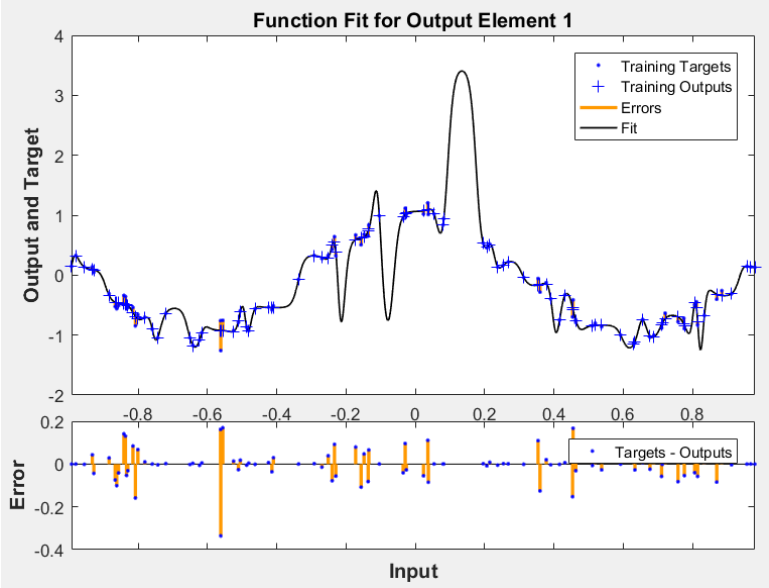
Plot the estimated function on the same picture as 2.3. Compare the two estimated function and comment.

As was said in the last question the plot that corresponds to no validation set overfits the data. Its is obvious in the [0;0.2] interval or in x = -0 .2 in the X axis. This means the fit is too specific to this data and if any other point was added to the dataset it would probably increase the error (or LS cost) by a lot.

In the other hand the plot corresponding to the data trained with validation set, although having higher error, has a curve that follows the data but is not overly specific to it. If new points were added the error would probably not increase as much as with a fit trained with no validation set.



With validation set



Without validation set