Instituto Superior Técnico

Departamento de Engenharia Electrotécnica e de Computadores

Machine Learning

3rd Lab Assignment

Shift: 4ª feira Group Number: 8

Number 79069 Name Bruno Gonçalves

Number Name Afonso Costa

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,

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

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.

Com o método do gradiente procuramos minimizar uma função de custo iterativamente, proporcionalmente ao gradiente negativo dessa mesma função. Neste caso, procuramos otimizar os pesos do MLP para que o peso função de custo (minimum square errors) seja o mínimo possível. Assim, os pesos são atualizados segundo a equação $w^{n+1} = w^n - \Delta w^n$, sendo que $\Delta w^n = -\eta \nabla E$ onde ∇E é o gradiente da função de custo dos pesos a minimizar e η a *learning rate* ou o passo dado em cada iteração em direção ao mínimo da função. Neste caso, com um *fixed step size*, η mantém-se constante ao longo do treino.

Quanto à utilização de *momentum*, neste caso $\Delta w^n = -\eta \nabla E + \alpha \Delta w^{n-1}$, onde $0 \leq \alpha < 1$. Assim, a utilização desta técnica introduz alguma "inércia" no movimento em direção ao mínimo, uma vez que a atualização dos pesos em iterações seguintes depende da iteração anterior. Desta forma consegue-se prevenir oscilações, aumentando a velocidade com que se descem "ravinas". Quanto maior o parâmetro α, mais o efeito desta técnica é pronunciado.

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 training time of the network. Indicate the values that you obtained.

Step size: 10 Momentum: 0.7

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: 35
Values: 27 28 31 34 34 35 36 40 41 38 60

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.

Quando utilizamos a técnica de aceleração de *adaptive step sizes*, utilizamos *learning rates* individuais para casa peso, e ajustamo-las em cada iteração do algoritmo dependendo do que se passou nas iterações anteriores. Por exemplo, se em duas iterações seguidas resultaram em atualizações de um peso na mesma direção, então o *learning rate* deverá aumentar. Caso contrário, se duas atualizações foram em sentidos contrários, então o *step size* deverá diminuir. Este método rege-se pelas seguintes equações:

$$\eta_{ji}^{n} = \begin{cases} \eta_{ji}^{n-1} u & \text{if } \left(\frac{\partial E}{\partial w_{ji}}\right)^{n} \text{ and } \left(\frac{\partial E}{\partial w_{ji}}\right)^{n-1} \text{ have the same sign} \\ \eta_{ji}^{n-1} d & \text{if } \left(\frac{\partial E}{\partial w_{ji}}\right)^{n} \text{ and } \left(\frac{\partial E}{\partial w_{ji}}\right)^{n-1} \text{ have different signs} \end{cases} \qquad \Delta w_{ji}^{n} = -\eta_{ji}^{n} \frac{\partial E}{\partial w_{ji}}$$

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: 74
Values: 68 71 71 71 72 74 75 76 101 104 116

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.

Step size: 8 Momentum: 0.6
Median of the numbers of epochs: 64
Values: 53 60 60 63 63 64 72 75 81 86 89

Fixed Step Size:
Com passo fixo verifica-se que o número de épocas varia cerca de 25% para uma variação de 20% do step size. No que toca ao parâmetro do momentum, ao variar 20% o valor deste, obtém-se uma variação de 60% no número de épocas.

Adaptative Step Size:
Com passo adaptativo verifica-se que o número de épocas varia cerca de 15% para uma variação de 20% do step size. No que toca ao parâmetro do momentum, ao variar 20% o valor deste, obtém-se uma variação de 15% no número de épocas.

As variações de 20% introduzidas nos parâmetros foram feitas em relação ao valor óptimo respetivo e foi feita a media para ±20%.
Com estes resultados conclui-se que o método com passos adaptativos é mais resistente às variações dos parâmetros iniciais, logo tem uma sensibilidade menor.

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);
```

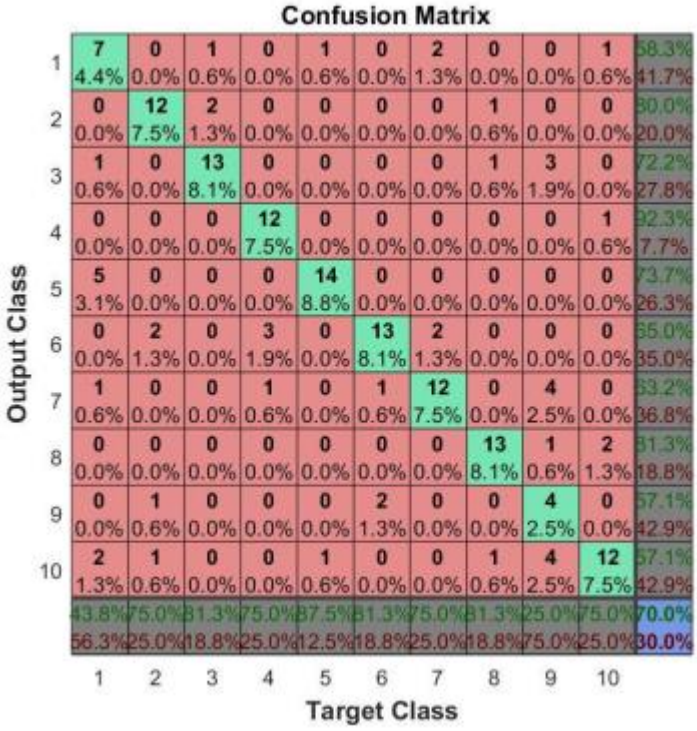
Comment on the values in the confusion matrix you obtained. Is the accuracy the same for every digit? Are the errors what you'd expect?

Verifica-se que os resultados obtidos são interessantes visto evidenciarem uma taxa de eficácia maior que 50%. No entanto, estes resultados seriam melhores se fosse utilizado o conjunto de treino para a construção da matriz. Isto deve-se ao facto de estes dados de teste moldarem menos o modelo que os dados de treino.

Para digitos diferentes, obtiveram-se valores de eficácia diferentes, pois alguns digitos podem ser mais dificeis de aprender que outros. Podem também ocorrer um digito ser confundido com outro por serem parecidos. É o caso do 6 e do 8.

O erro é o esperado porque apesar de se alcançar um erro de treino pequeno, isto pode não refletir uma eficácia perto de 100%, pois estas são duas medidas completamente distintas

Para se obter uma eficácia melhor poderia utilizar-se um critério de paragem que não envolve-se o MSE, mas sim uma avaliação qualitativa.



Write down the performance values that you obtained:

Training error: 0.0500 Test set Accuracy: 70,0%

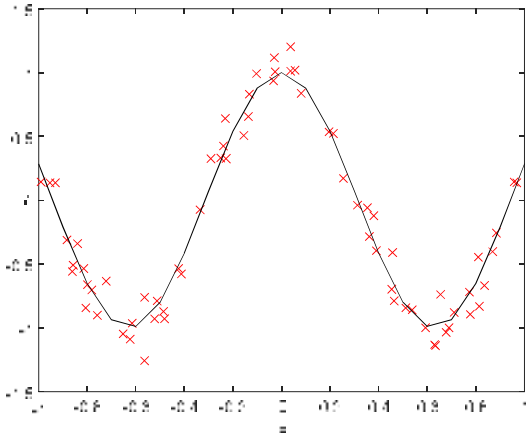
Which quantity (mean squared error, or global percentage of correct classifications) is best to evaluate the quality of networks, after training, in this problem? Why?

O MSE, para este caso não apresenta uma avaliação tão correta para este sistema como a percentagem de classificações corretas. Neste tipo de processos estamos interessados em ter uma avaliação qualitativa e não quantitativa, como o MSE nos dá, através do erro absoluto. O MSE funciona melhor para processos continuos. Para este processo, a melhor avaliação é feita através das percentagens pois estas evidenciam como é que o nosso modelo classifica as patterns, formulando uma eficácia para cada classe.

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 $d = f(x) + s$, in which s 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.

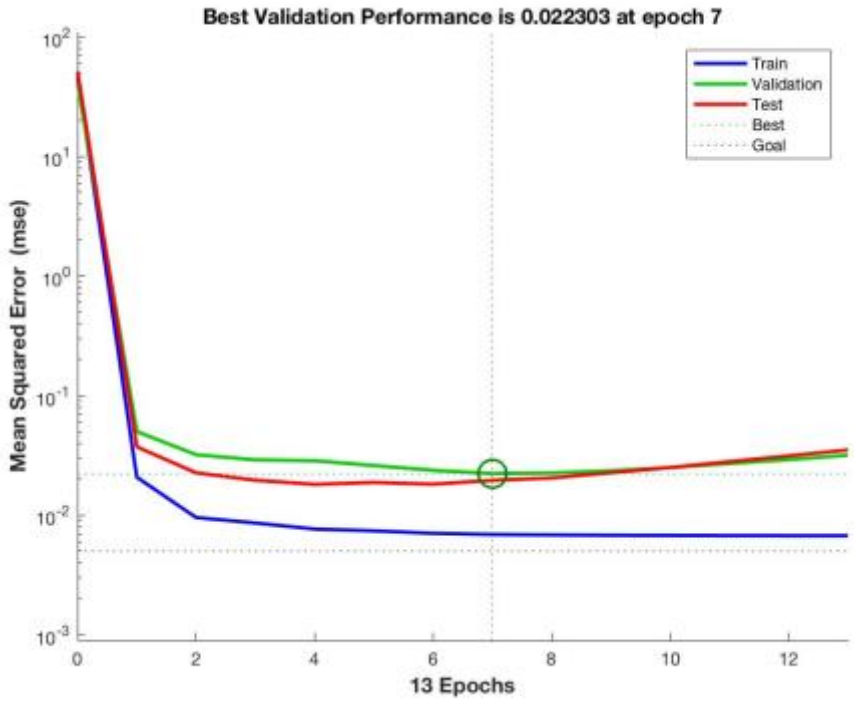
2.3 Training with a validation set

(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?

Quando treinamos a rede com um *validation set*, temos como objetivo avaliar imparcialmente o treino efetuado (feito com o *training set*). Interessa-nos aqui, por exemplo, minimizar o *overfitting*. Na validação, não nos preocupamos a ajustar os pesos da rede neuronal, tentamos perceber se o treino que está a ser feito (utilizando o *training set*) está a traduzir-se num aumento de eficácia num conjunto de dados fora do *training set*. Por outras palavras, o treino da rede neuronal está iterativamente a diminuir o *mse* na predição de dados no *training set*. Na validação interessa-nos perceber se isso também está a acontecer em dados que ainda não foram mostrados à rede neuronal. Desta maneira, se o erro ao longo dos dados de treino, em cada iteração, diminui, mas o erro ao longo dos dados de validação, em cada iteração aumenta ou mantém-se igual então quer dizer que estamos perante *overfitting*. Dessa forma devemos escolher os pesos correspondentes à época onde se obteve o menor erro na validação, que poderá ou não ser os pesos obtidos no final do treino.

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.

Neste gráfico conseguimos perceber a eficácia da rede ao longo das diferentes fases da aprendizagem, bem como ao longo das épocas. Como é de esperar, a curva que diz respeito ao treino é aquela que tem um *mse* menor. Isto deve-se ao facto dos pesos estarem a ser ajustados utilizando este tipo de dados, ou seja, a rede acaba por estar *fitted* para este conjunto de dados e, para dados fora deste conjunto, o erro tende a ser maior. Além disso, também como esperado, a curva dos dados de teste, segue bastante aproximada à dos dados de validação. Para o nosso caso, a melhor *performance* foi obtida na época 7, onde o *mse* obtido para a validação foi de 0.02303. A partir desta época o erro no *validation set* começa a aumentar um pouco, apesar de o erro no treino continuar a diminuir.

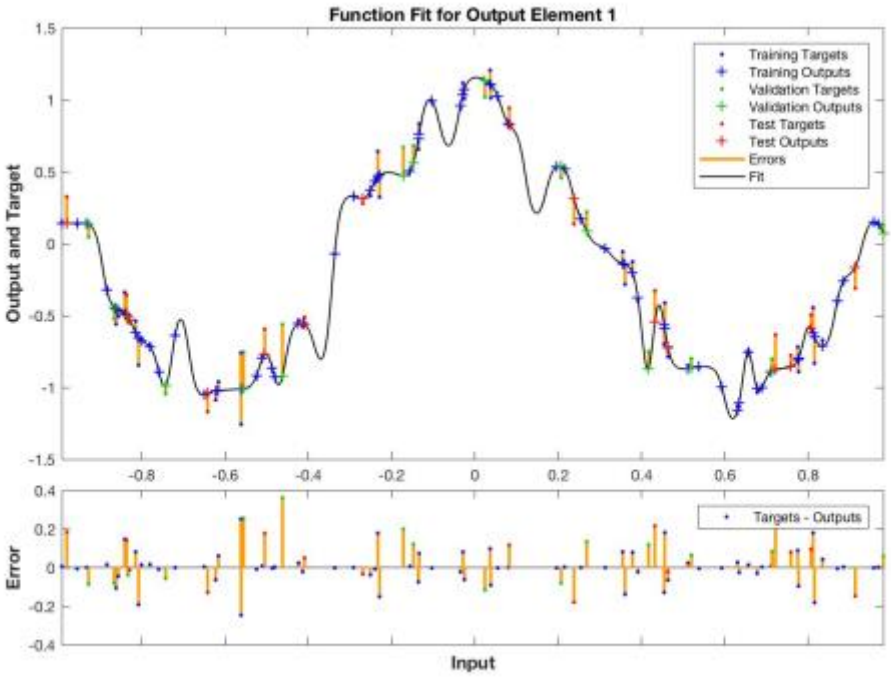


Plot the training data, the test data and the estimated function, all in the same figure and comment.

A função estimada acaba por seguir um pouco a curva do cosseno, no entanto apresenta alguns "desvios" para se adaptar ao ruido e aos seus pontos.

Como é de esperar, os pontos a azul, de treino, são aqueles que se nota que o *fit* mais "segue". Neste conjunto de dados, em geral, os *training targets* e os *training outputs* são iguais, salvo algumas exceções.

No que diz respeito ao conjunto de teste, na pergunta anterior reparou-se que o erro era maior em relação ao conjunto de treino, o que é evidenciado agora no *plot*, pois notam-se alguns desvios entre *target*s e *outputs*.



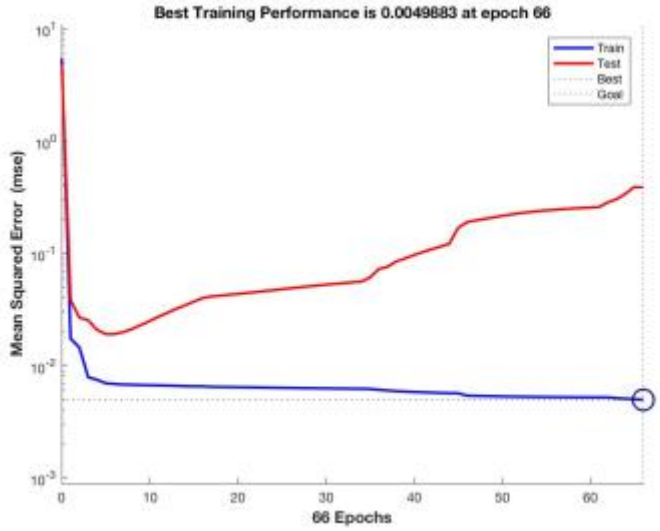
2.4 Training without a validation set

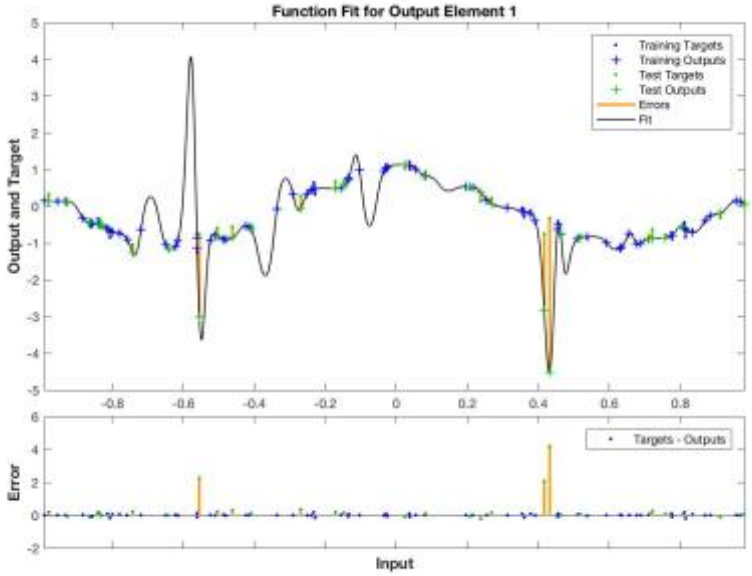
Repeat the previous item but do not use a validation set this time. Observe the evolution of the cost function for the different sets and comment. Is there any sign of overfitting?

Analisando a evolução da função de custo, percebemos que por volta da época 6, num total de 66 épocas, o erro deixou de diminuir no conjunto de teste, para aumentar a partir deste ponto. O erro no conjunto de treino continuou a diminuir, como esperado.

No final do treino, quando é atingido o objetivo para o *mse* de treino, este encontra-se então próximo de 0,005, e o *mse* para o conjunto de teste encontra-se próximo de 0,5. Desta forma, nota-se claramente que a rede se encontra *overfitted* para o conjunto de treino e que para dados novos, fora deste conjunto de treino, a rede tenderá a comportar-se mal.

A existência de um conjunto de validação poderia resolver este problema de *overffiting*, pois poderíamos ter escolhido os pesos da época para a qual o erro de validação parava de diminuir.


Best Training Performance is 0.0049883 at epoch 66
Train
Test
Best
Goal
Mean Squared Error (mse)
66 Epochs


Function Fit for Output Element 1
Training Targets
Training Outputs
Test Targets
Test Outputs
Errors
Fit
Output and Target
Targets - Outputs
Error
Input

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

Na função estimada sem validação nota-se claramente a existência de algumas curvas que se afastam do conjunto de dados, sinais de *overfitting*, colocando o *output* da função em valores como 4. Como é esperado, no conjunto de dados de treino, a curva porta-se bastante bem, estando estes todos cobertos pela curva, com erro menor que no caso da função anterior. No conjunto de dados de teste, existem pontos que introduzem bastante erro, cuja curva falhou em estimar. Comparando as duas estimativas, é bastante manifesto que a primeira estimativa comporta-se muito melhor para dados fora do conjunto de treino e portanto, seria muito mais eficaz em predições em dados novos. Sabemos o *mse* é uma aproximação da variância e esta é o quadrado do desvio padrão. Uma vez que os dados de treino já apresentam um desvio em relação à função que queremos obter, este efeito vai-se refletir diretamente no nosso modelo, que não se irá conseguir aproximar com rigor da função cos(5x).