Instituto Superior Técnico

Departamento de Engenharia Electrotécnica e de Computadores

Machine Learning

3rd Lab Assignment

Shift: Wed - 14h Group Number: 1

Number 78486 Name Luís Bernardo de Brito mendes Rei

Number 78671 Name João Miguel Limpo de Lacerda Pestana Girão

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.

A linear model can be learned by solving a linear system of equations. However, most models are learned by solving an optimization problem $\hat{\theta} = arg\min_\theta J(\theta)$, where θ denotes the model parameters. In most cases, we rely on numerical optimization algorithms as is the case of the gradient method (first derivatives).

If $\theta \in \mathbb{R}^P$, and J(θ) a differentiable function in a neighborhood of a point $\theta^{(t)}$, then $J(\theta)$ decreases fastest if we move along the opposite direction of the gradient.

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta J(\theta^{(t)})$$

The procedure is repeated until the function stops decreasing, meaning that we reached the vicinity of a local minimum or a plateau. This algorithm is called a gradient descent or steepest descent algorithm, and because it includes the contribution of all the training patterns it's cataloged as batch mode.

The choice of step sizes/learning rate issues a trade-off: if η is too small the update process becomes very slow, but if η is too large the algorithm may skip local minima or produce and update of θ that increases the cost function $J(\theta)$. One technique used to speed up convergence is the momentum technique.

This method performs a low pass filter on the gradient sequence and updates $\theta^{(t+1)}$ using the filtered gradient $v^{(t+1)}$:

$$v^{(t+1)} = \alpha v^{(t)} - \eta \nabla_\theta J(\theta^{(t)}),$$
$$\theta^{(t+1)} = \theta^{(t)} + v^{(t+1)}.$$

The parameter $\alpha \in]0,1[$, typically ranges from 0,5 to 0,95. This technique improves the convergence rate, specifically if the cost function $J(\theta^{(t)})$ exhibits deep valleys in which the gradient method is slow. The objective function J is evaluated in each iteration to check if it decreases. If not, the memory of the moment term is set to zero.

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: 7 Momentum: 0.6

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: 38

Tests	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
Results	45	37	43	29	38	49	40	36	33
Median	29	33	36	37	38	40	43	45	49

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 momentum technique, as explained before, introduces a momentum term α to the gradient method in order to accelerate its convergence:

$$v^{(t+1)} = \alpha v^{(t)} - \eta \nabla_\theta J(\theta^{(t)}),$$
$$\theta^{(t+1)} = \theta^{(t)} + v^{(t+1)}.$$

The adaptive step size method introduced in this question assumes that the step size η changes differently for each component of θ in each iteration:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_i^{(t)} J(\theta^{(t)}).$$

The step size is updated with

$$\eta_i^{(t)} = \begin{cases} u\eta_i^{(t-1)}, & \text{if } \frac{\partial J}{\partial \theta_i}(\theta^{(t)}) \cdot \frac{\partial J}{\theta_i}(\partial \theta^{(t-1)}) > 0 \\ d\eta_i^{(t-1)}, & \text{otherwise} \end{cases}$$

where typically u = 1,2 and d = 0,8. This technique performs very well if the cost function contains valleys aligned with the x axes.

The objective function J is evaluated at each iteration to check if it decreases as expected (u > 1 to ensure that we continue the approximation). If not, the previous values of the parameters are kept and the step sizes reduced (d < 1) because we have already surpassed our goal. Combining the two techniques we arrive at

$$v^{(t+1)} = \alpha v^{(t)} - \eta_i^{(t)} \nabla_\theta J(\theta^{(t)}),$$
$$\theta^{(t+1)} = \theta^{(t)} + v^{(t+1)}.$$

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

Tests	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
Results	102	112	112	101	106	106	111	109	177
Median	101	102	106	106	109	111	112	112	177

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				
		1	3	5	7	10
Momentum term	0.3	233	96	102	49	126
	0.5	239	98	55	45	48
	0.6	220	95	50	38	55
	0.7	276	78	62	40	53
	0.8	173	85	66	62	93

* Fixed step size parameter

		Step size					
		1	3	5	7	10	12
Momentum term	0.3	105	89	97	73	97	76
	0.5	87	106	108	58	58	74
	0.6	105	96	71	100	89	100
	0.7	79	74	74	76	72	65
	0.8	88	60	95	68	55	77
	0.85	85	80	77	86	62	66

* Adaptive step size parameter

For a step size of 10 and momentum term of 0.8 we arrive at a median number of epochs of 58.

Tests	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
Results	58	62	46	57	59	55	58	76	76
Median	46	55	57	58	58	59	62	76	76

With adaptive step sizes the algorithm varies little with the parameters' changes. This is due to the fact that the step sizes are iteratively adapted to the cost function and the initial values tend to lose significance. Even when those initial values are far from the optimal ones they will converge to a goal point.

In the case of the fixed step sizes, the influence of the choice of our parameters is felt as the values fluctuate a lot along both the axis. Overall, the momentum term is more impactful at the question 1.3 settings because in the current setup the adaptive learning rate command the convergence at each iteration.

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?

The obtained confusion matrix is observable in annex (figure 1). By analyzing it, we conclude that 7 of the classes have a classification error of under 20%, while of the other 3, one presents an error of 25%, while the other two of approximately 40%. This observations showcase the variability in the accuracy of the neural network's classifier between different digits.

Note that the errors aren't all of the same kind, as some numbers are easily mistaken with other numbers ('0' is mistaken with half the others numbers) while others tend to be confused as a specific number that shares some silhouette similarities ('6' is only mistaken for '5').

The errors presented are expected as the strong distortion on the outline of some of the numbers will compromise the neural networks accuracy.

Write down the performance values that you obtained:

Training error: 0.049738 Test set Accuracy: 81.3%

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?

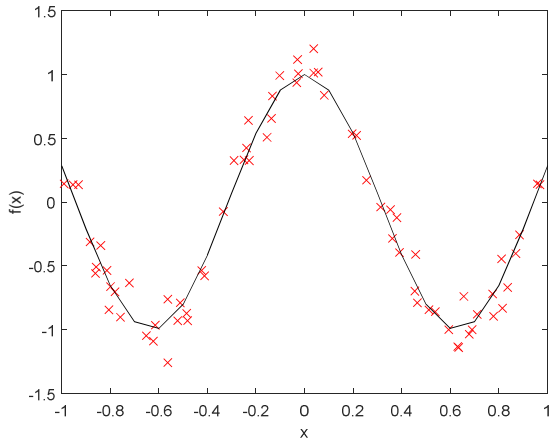
Often, continuous scores are intermediate results that are only converted to class labels (usually by thresholding) at the very last step of the classification. You can calculate the MSE using these continuous scores rather than the class labels with the advantage of avoiding the loss of information due to the dichotomization.

In our problem, however, to evaluate the quality of networks after training, the best option is to use the global percentage of correct classifications as it takes into account the binary result (right or wrong) of our classification, not considering how wrong it is (there is no numerical relation between a number and its misclassified part, i.e. estimating a 0 as an 8, isn't any better/worse than not distinguishing between a 6 as a 5).

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?

Rosenblatt proposed an iterative algorithm to train the weights with a validation set. It starts by collecting a data set $\mathbb{T} = (x^1, y^1), ..., ((x^n, y^n))$, with $x^k \in \mathbb{R}^P, y^k \in 0, 1$, and initializing randomly the weights $w_i(0), i = 0, ..., p$. After presenting training patterns (x(t), y(t)) to model and compute the model output, the weights are updated according to $w_i(t) = w_i(t-1) + \eta x_i(t)\xi(t)$, with $\xi(t) = y(t) - \hat{y}(t)$, where y(t) is the desired outcome for the input x(t). This procedure is repeated until a stop condition is met.

The objective of the algorithm is to minimize the empirical error(*) between the network output and the real output in a way that the neural network can, after weight adjustments, perform correct predictions of the output value for input values outside of the training set.

(*) If the desired output, y, is different from the predicted outcome, $\hat{y} = f(x)$, we defined a loss $L(y, \hat{y})$. Since $L(y, \hat{y})$ is a random variable we can measure an average loss computed from the training data (empirical risk)

$$R = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)).$$

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 performance plot containing the mean squared error measurement obtained for each epoch for the training, validation and test sets is in annex in figure 2. It is observable that in the last epoch the MSE of the training set reaches a value below the defined threshold (0.005).

We verify that in the 5^{th} epoch the minimum MSE is reached on the validation set, increasing all the way from that point until the end of the simulation. We can then conclude that the parameters that best approximate our function, and that the neural network uses, are from outside of the testing set. The inclusion of the validation set guaranteed a more robust solution.

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

In figure 3, the points representing the training and test data are presented, as well as the estimated function. Through observation we see that the shape of the desired function is fairly well preserved (a better comparison is observable in figure 6 where the original function f(x) = cos(5x) without noise is also seen). This happens because the weights used originate from the validation set, restricting them to more acceptable values, and allowing them to preserve the topology of the testing points without over fitting them and disfigure our estimated function plot into something alien and completely different from the cosine we were trying to replicate.

The noise plays an important role not allowing the completely viable approximation we wanted. Because the neural network tends to follow the testing set which is corrupted by Gaussian noise, the approximated function follows said noise.

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?

Looking solely at the evolution of the cost functions (Figure 2 and 4) there is no significant difference between the results obtained with and without validation. This is because even if in both cases the network is trying to minimize the error using MSE, it is extremely unlikely that the weights will be updated similarly. When there is no validation there are less constraints and so the network will try to fit the predicted function to the results as much as possible, resulting in the overfitting seen on figure 5.

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

When comparing both predicted functions with the real output values (figure 6), it is obvious which is the best approximation as one tends to fit better to the training data but ends up less similar to the target values. Even without knowing the expected function one can assume it takes the form of a sinusoid by inspecting the results with the validation set, while looking at the results obtained without the use of said set doesn't give us a good idea of the target function. It should be noted that even the validated set does not, and will not, fit perfectly the cosine function due the noise in the training set.

Concluding, the validation set is a useful tool for making the neural network solution more robust and more likely to fit the function to estimate.

ANNEX

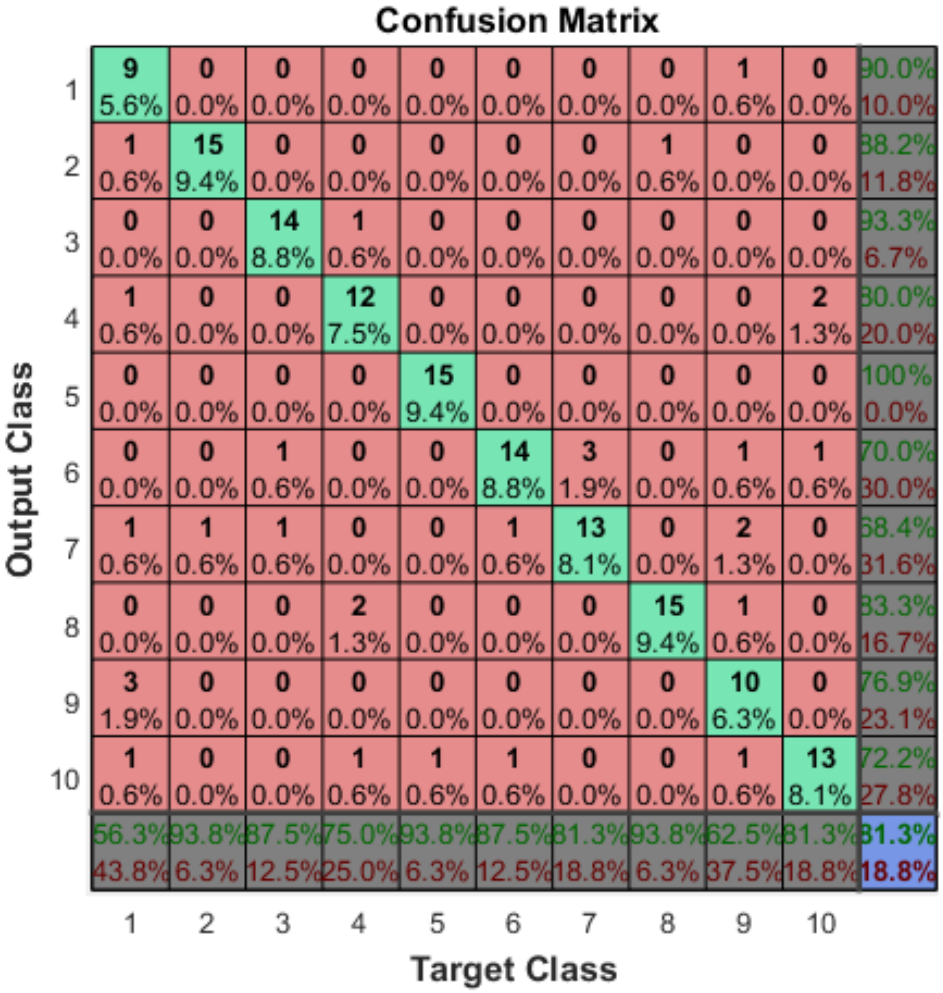


Figure 1 – Confusion matrix for the Gradient method with adaptive step sizes and momentum.

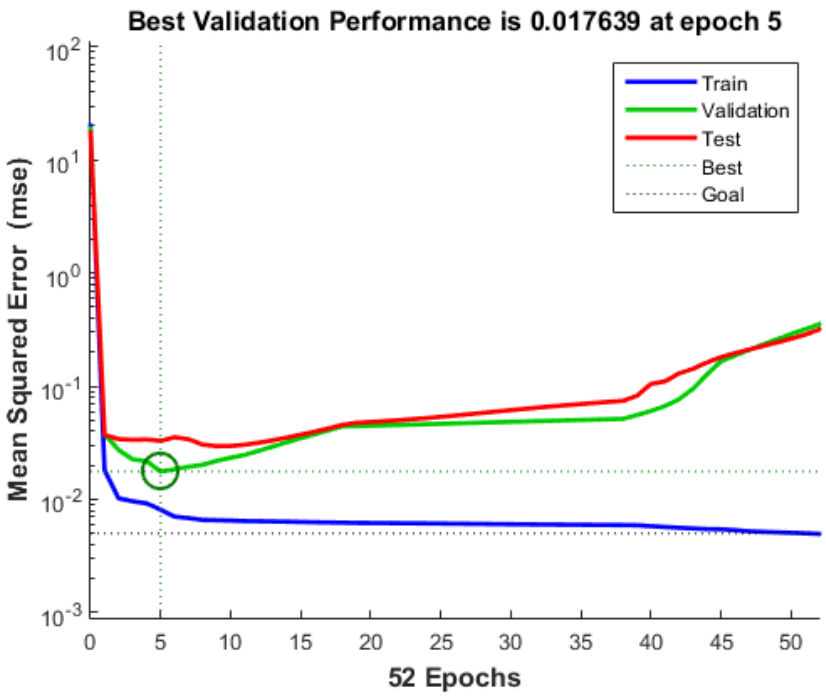


Figure 2 – Performance training with a validation set.

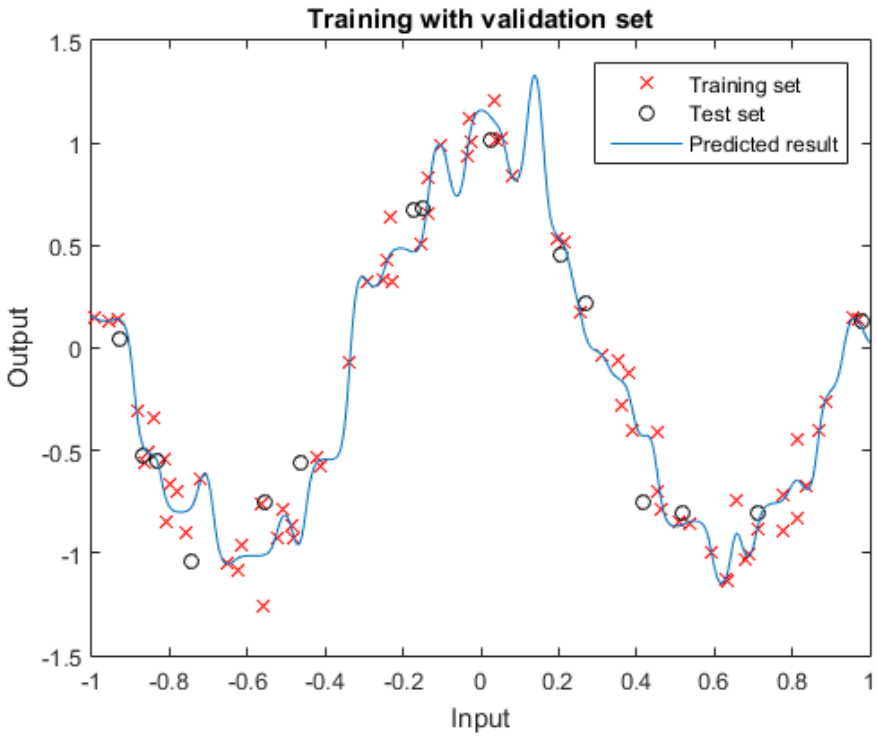


Figure 3 – Training data, test data and estimated function for the training performance with validation set.

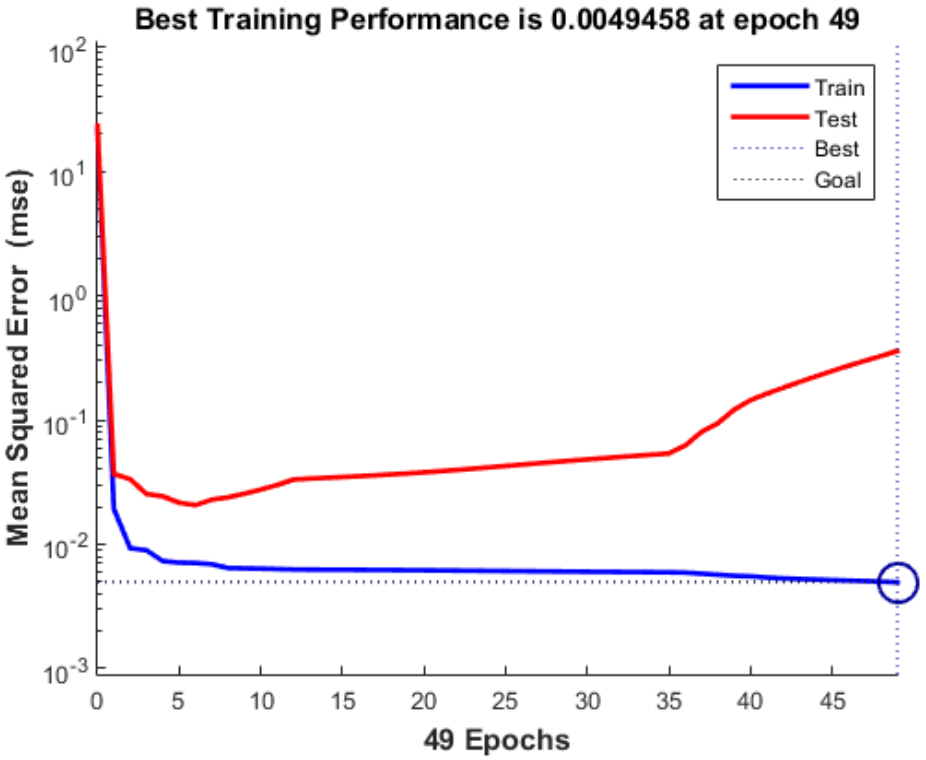


Figure 4 – Performance training without a validation set.

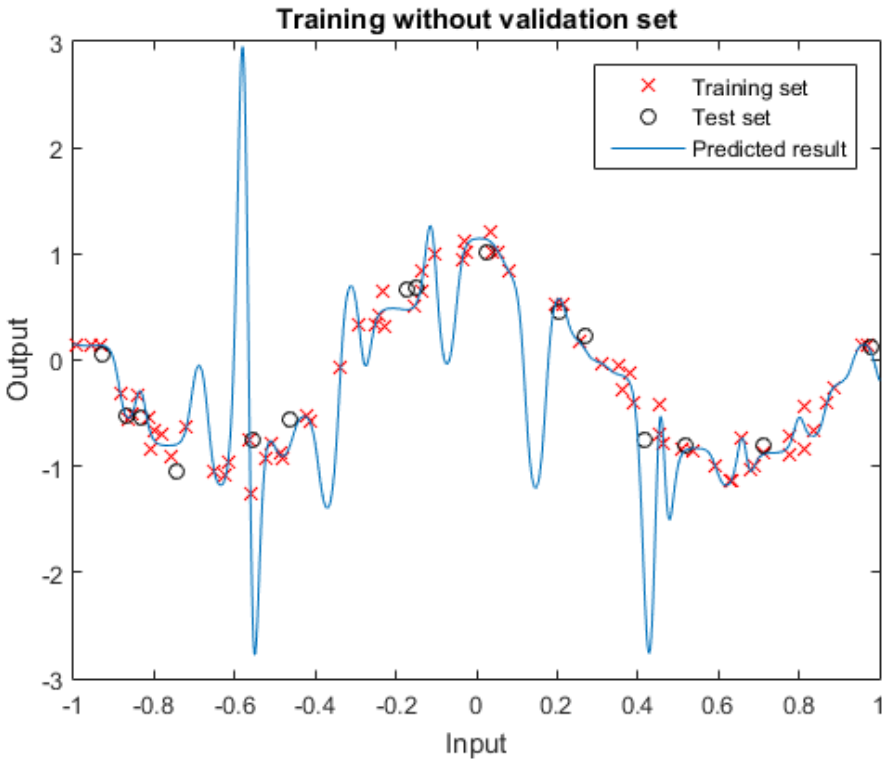


Figure 5 – Training data, test data and estimated function for the training performance with validation set.

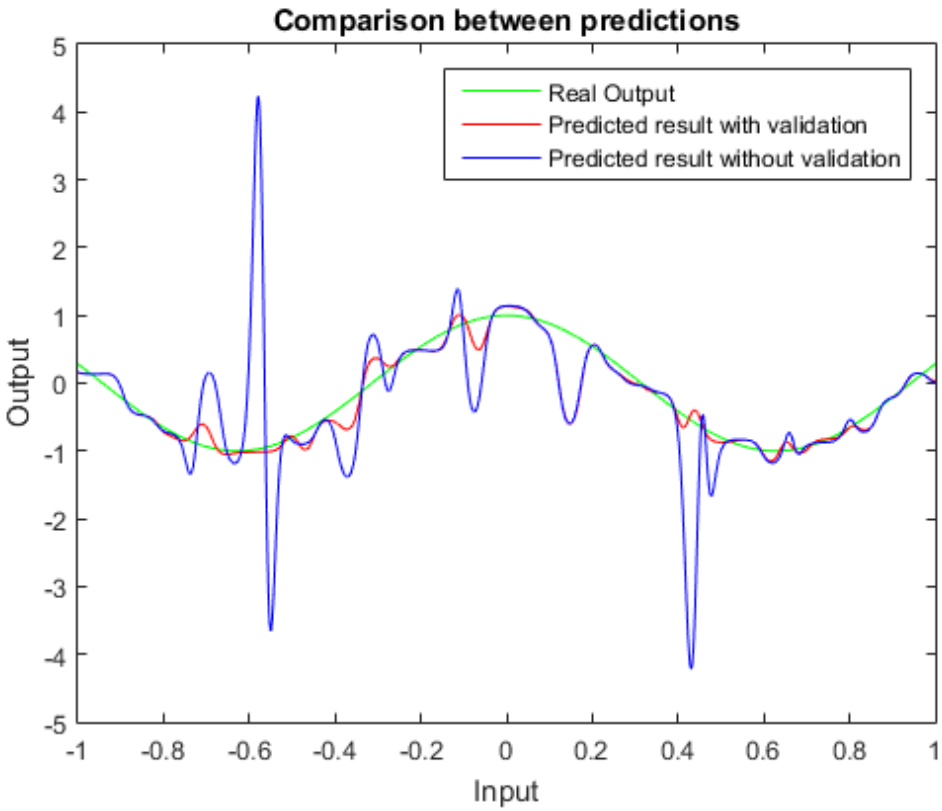


Figure 6 – Comparison of the two estimated functions.