Assignment 1 (ANN & Deeplearning)

Student: Afraz Salim *r-number: r0439731*

In this section, we will build a feedforward neural network architecture to approximate a non-linear function in $3D$. The input layer consist of two neurons and the output layer consists of one neuron. The decision on number on the of hidden layers and number of neurons within those layer, is based on trial and error mechanism.

We first sample our data to get training, test and validation data. Each of these samples contain 1000 data points which are drawn independently.These samples should represent the population. One advantage of the drawing the samples is that they are unbiased. Figure 1 shows the interpolated training data.

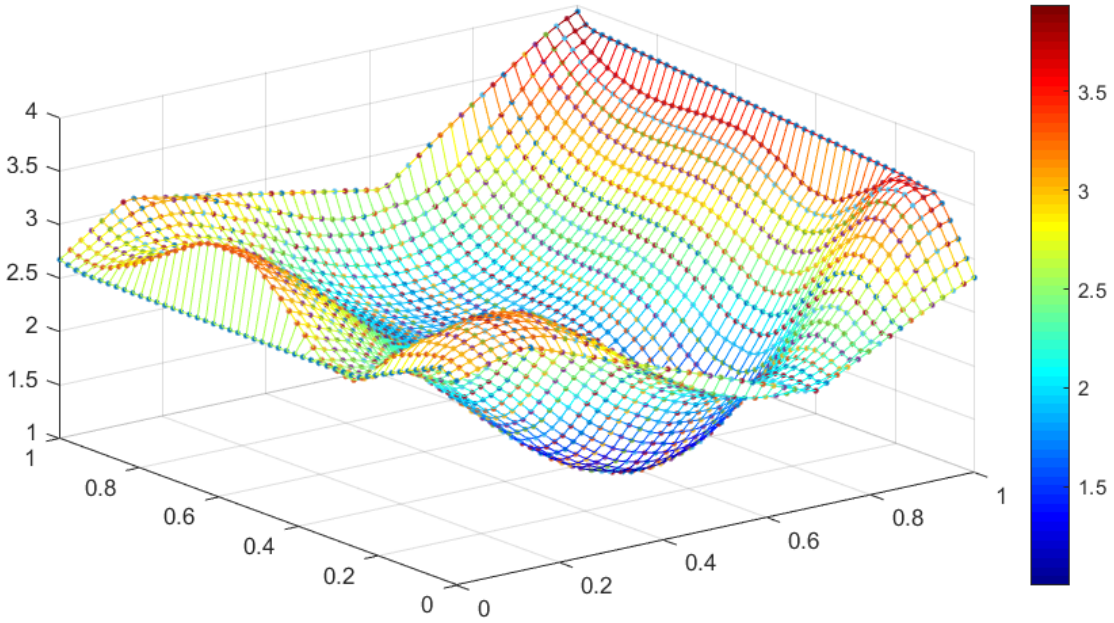


Figura 1: Interpolated plot of the training data

Training a Feedforward neural network:

We will build a small feedforward neural network architecture that will be used to approximate a non-linear function. As we know, there is no single rule to build a neural network architecture, we will start with a worst model(with too many neuron and layers) and tune the model on the basis of validation and test set. We are trying to approximate $z = f(x, y)$. If we build a worst model(with too many neurons), then we need one layer for x direction and a second layer for y direction. We now also assume that the interaction between two variables is not captured by any of these two layers. So we add a third layer.

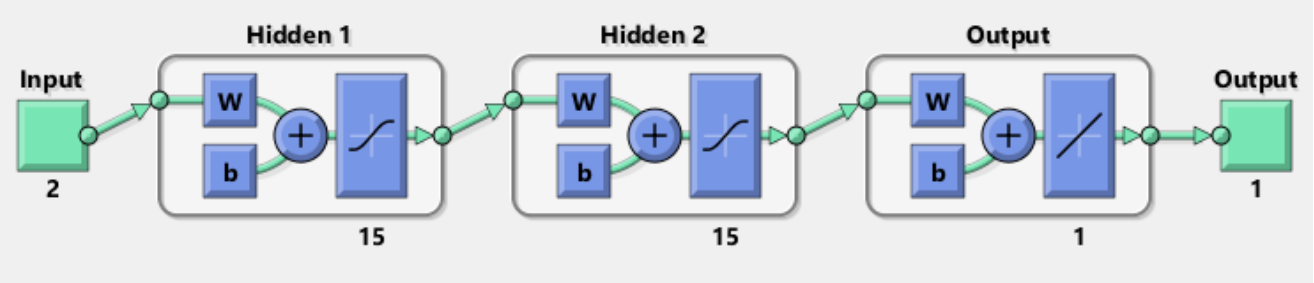


Figura 2: Feedforward model.

Our initial model consist of 3 layer and arround 50 neurons in each layer. However, by tuning our parameters with validation and test set, we could reduce number of layers to 2 and 14 neurons in each layer. Following table show the mean squared errors which were obtained by trying different models.
Our final model looks as shown in following figure 2:

Cuadro 1: Different models and their MSE

Layer	neurons	EPOC	MSE	Learning function
1	150	1000	$8{,}8183 \times 10^{-07}$	trainlm
2	40	1000	$3{,}201 \times 10^{-06}$	trainlm
2	25	1000	$3{,}3036 \times 10^{-07}$	trainlm
2	20	1000	$2{,}6552 \times 10^{-07}$	trainlm
2	15	100	$1{,}0983 \times 10^{-08}$	trainlm
2	10	1000	$1{,}6463 \times 10^{-06}$	trainlm
3	30	1000	$2{,}3935 \times 10^{-06}$	trainlm
3	20	1000	$6{,}1721 \times 10^{-07}$	trainlm
3	15	1000	$4{,}302 \times 10^{-07}$	trainlm

We select this model because we achieve better performance with a simpler model. Mean squared error on test data is $1{,}0983 \times 10^{-08}$.However, the model with three layers also achieves same performance but we prefer a simple model.
Levenberg marquardt outperforms all other learning algorithms except trainbr. Since we have to use validation set, so we have chosen for trainlm. It can be seen in figure **??** that the error on the test set is lower than the error on validation set. This is good indication that there is no overfitting.

Performance Assessment:

Mean squared error on the test data is $1{,}0983 \times 10^{-08}$. Next, We plot the test data and the points estimated by the neural network. Figure 4 shows the test data and simulated data. We now plot the error level curves and also the error as the difference between the estimated points and real points. Figure 5 shows the error level curves while figure 6 shows the error on the vertical axis.

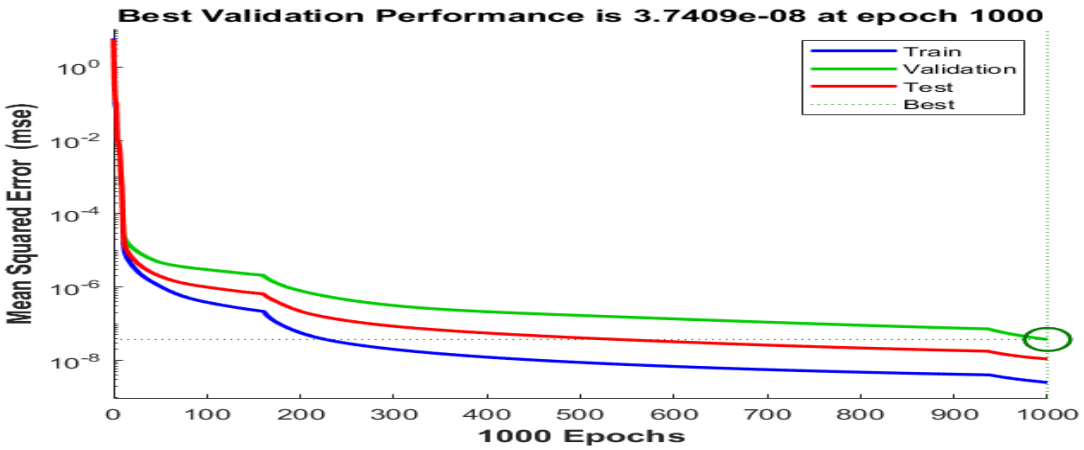


Figura 3: Interpolated plot of the training data

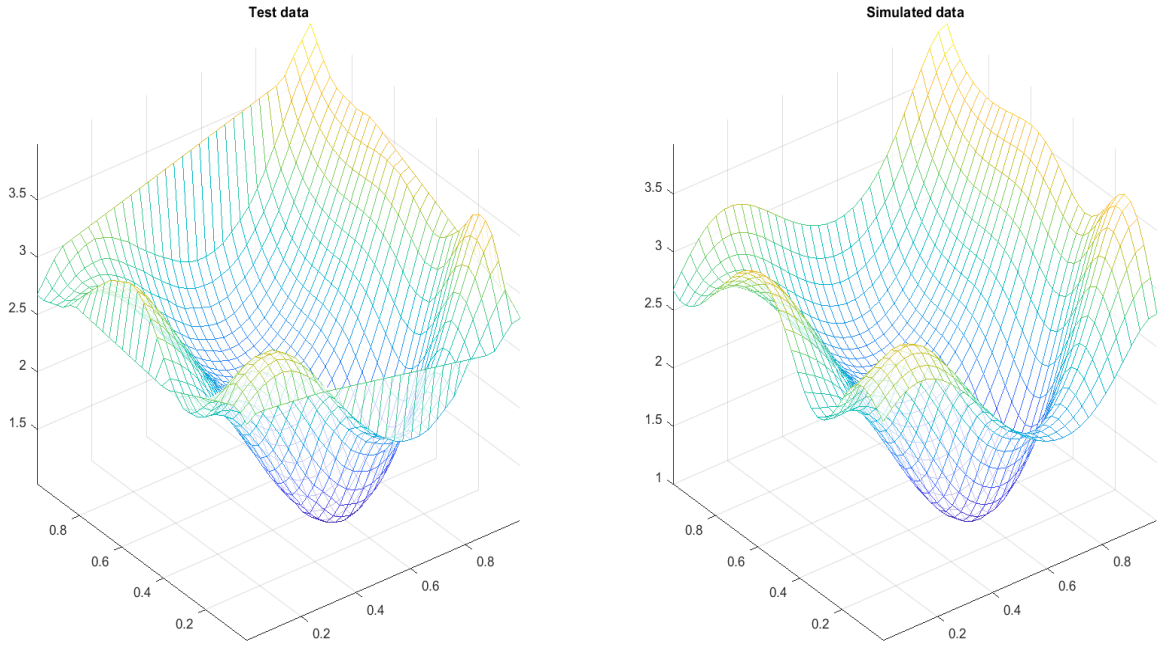


Figura 4: Test data vs estimated points

Generalization:

We can avoid a model to overfit by adding noise to input. In this section, we will add noise to the input and analyze the result. We will first analyze the results obtained with different algorithms on noiseless data.

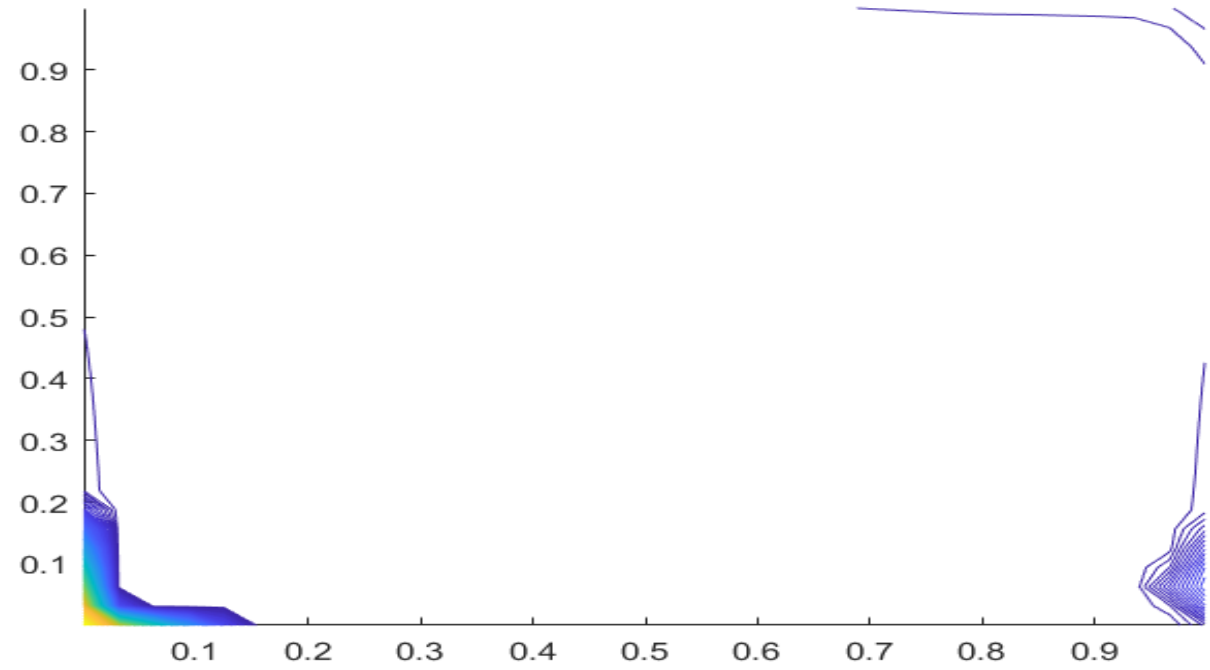


Figura 5: Error level curves.

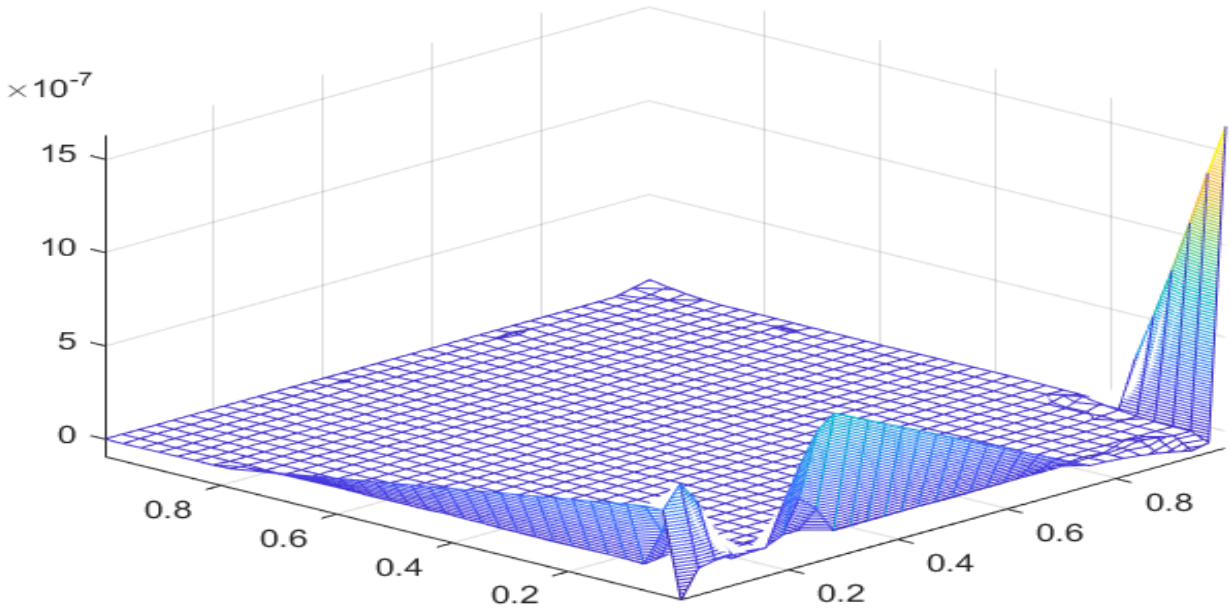
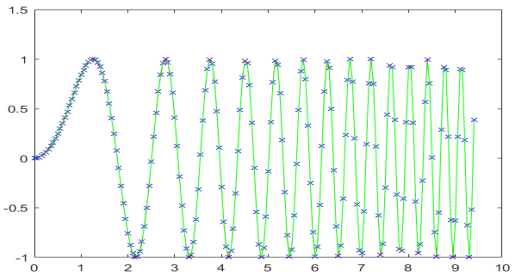


Figura 6: Error as a difference between estimated points and real points.

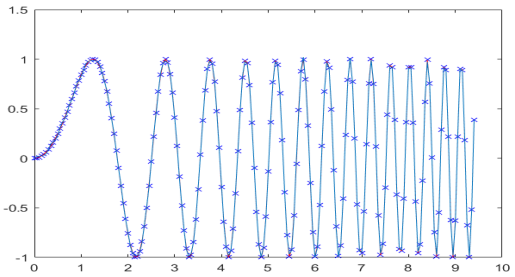
Algorithm	Epochs	neurons	MSE
trainlm	1	50	0.22219
trainbfg	1	50	4.4084
trainlm	14	50	29.9265
trainbfg	14	50	0.11529
trainlm	985	50	$2{,}2921 \times 10^{-09}$
trainbfg	985	50	$9{,}532 \times 10^{-05}$

MSE with different algorithm on noiseless data.

Table shown above shows the MSE with different algorithms on noiseless data. The following figures show the estimated sin curves on the noiseless data. We now test same algorithms with noisy data but



Noiseless estimation(trainlm)

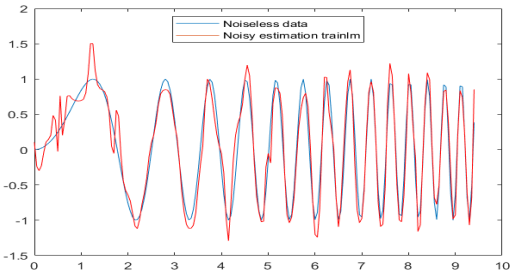


Noiseless estimation(trainbfg)

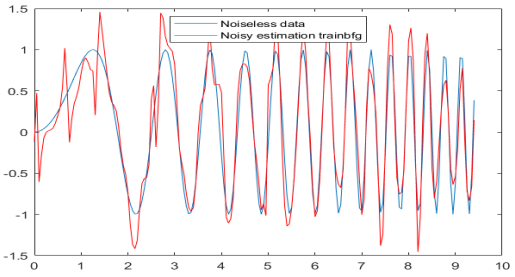
with 985 epochs.

Algorithm	Noisy Data	neurons	Epoch	MSE
Trainlm	Yes	50	985	0.0415
Trainlm	Yes	50	985	0.072392

The corresponding curves are shown in following figure. We now focus on *trainbr* algorithm. The MSE



Noisy estimation(trainlm)



Noisy estimation(trainbfg)

on the noiseless data with *trainbr* is $3{,}3014 \times 10^{-09}$. We now test *trainbr* with 100 neurons on the noisy data.
MSE error between estimated data and true data with *trainbr* is 0,039311. Figure 7 shows the curve that we obtained with trainbr with 150 neurons.

We did not include the gradient descent as we saw from the initial experiments that it requires many iterations to converge.

Conclusion

Bayesian regularization backpropagation tries to minimize the weights also. If we introduce the noise and increase the number of neurons then it will create many small peaks arround a large peak(because all vectors must have small weight and this corresponds to many smaller peaks) and at the same time, we avoid overfitting due to the noise. Trainbr also outperformed other learning algorithms in second section where we tried to approximate a non-linear functions but since we had to use validation set so we experimented on other algorithms.

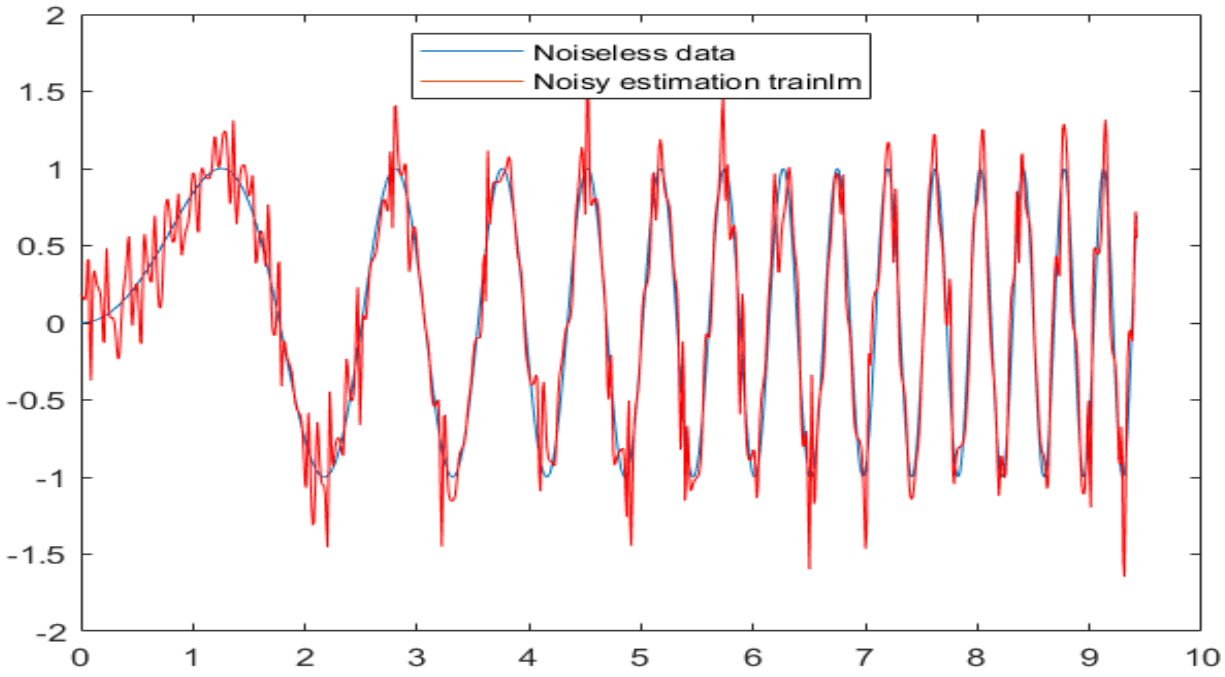


Figura 7: Error level curves.