**1.1 THE PERCEPTRON AND BEYOND**

1.We can use the linear relation in the activation function,where the weights of the perceptron model are analogous to the coefficients () on the linear model,and the class labels as target outputs.Then Perceptron model can use backpropagation to fit the model and minimize the sum of squared errors.

Link between the perceptron model and linear regression model: The two models are similar in structure.If the perceptron with no activation function,i.e. then the perceptron model is exactly equal to linear regression,or we can say in addition to acting as a classifier, linear models can also achieve regression effects.OLS minimizes the sum of squares of the residuals of all observed values,but in a perceptron,we can use different training functions to fit and perform it.

2. After using MATLAB to establish a linear neural network, I found that the linear model cannot adequately capture the relationship between the input and output data, this problem is analogous to the inability to solve the xor problem with a single layer perceptron.We can relate this to underfitting,which means a linear model is not complex enough to capture relationships between a dataset’s features and a target variable.

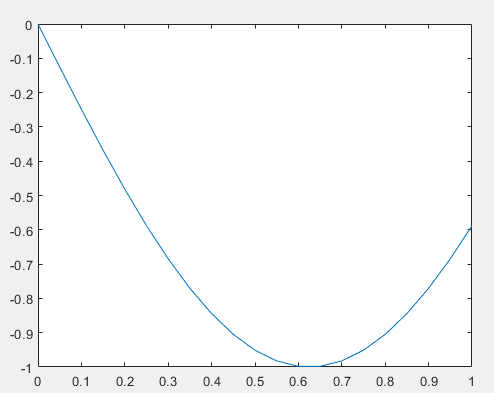


Figure1.1.1 underfitting

3.From the the training results,we can see the neural network uses MSE to measure errors,and the training function is Levenberg-Marquardt(trainlm) of default use.

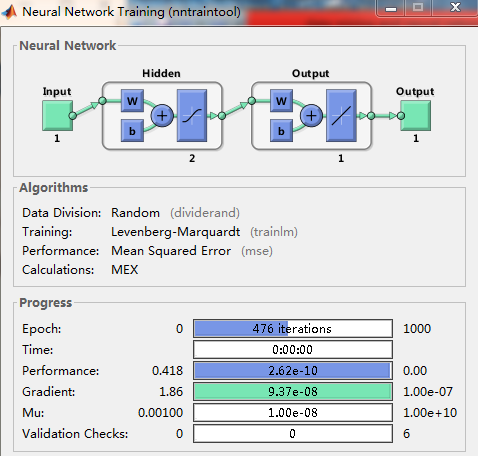


Figure1.1.2 training result

**1.2 BACKPROPAGATION IN FEEDFORWARD MULTI-LAYER NETWORKS**

1.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 1 | 2 |
| 2 | -1 | 4 |
| 3 | 0 | 3 |

Take the weighted sum of the input layer to hidden layer:

Execute sigmoid activation function

Take the weighted sum of the hidden layer to output layer:

,

Desired output :2

Backpropagation to compute gradient:

Output layer to hidden layer gradient:

((1-=0

Hidden layer to input layer gradient:==0

Similarly, we can calculate the other datapoint ,and we will get the same result.

It can be seen from the above analysis,both the output layer to hidden layer gradient and hidden layer to input layer gradient are equal to 0.

Therefore,after onestep of (simple) gradient descent, the updated network would look like the same as befor since the gradient equals 0,which is same thing as not updating the weights.

2.After having tried 5 different algorithms and different iterations, I got a sense of the similarities and differences between these algorithms.

Set the number of epochs equals 100 and find out the average iteration time of the five times:

algorithm1 = 'traingd' ; - batch gradient descent 0.3891s

algorithm2 = 'trainlm' ; - Levenberg - Marquardt 0.3230s

algorithm3 = 'trainbfg';- BFGS quasi Newton 0.6164s

algorithm4 = 'trainscg';- Scaled conjugate gradient 0.2554s

algorithm5 = 'trainbr';- Bayesian regularization 0.5816s

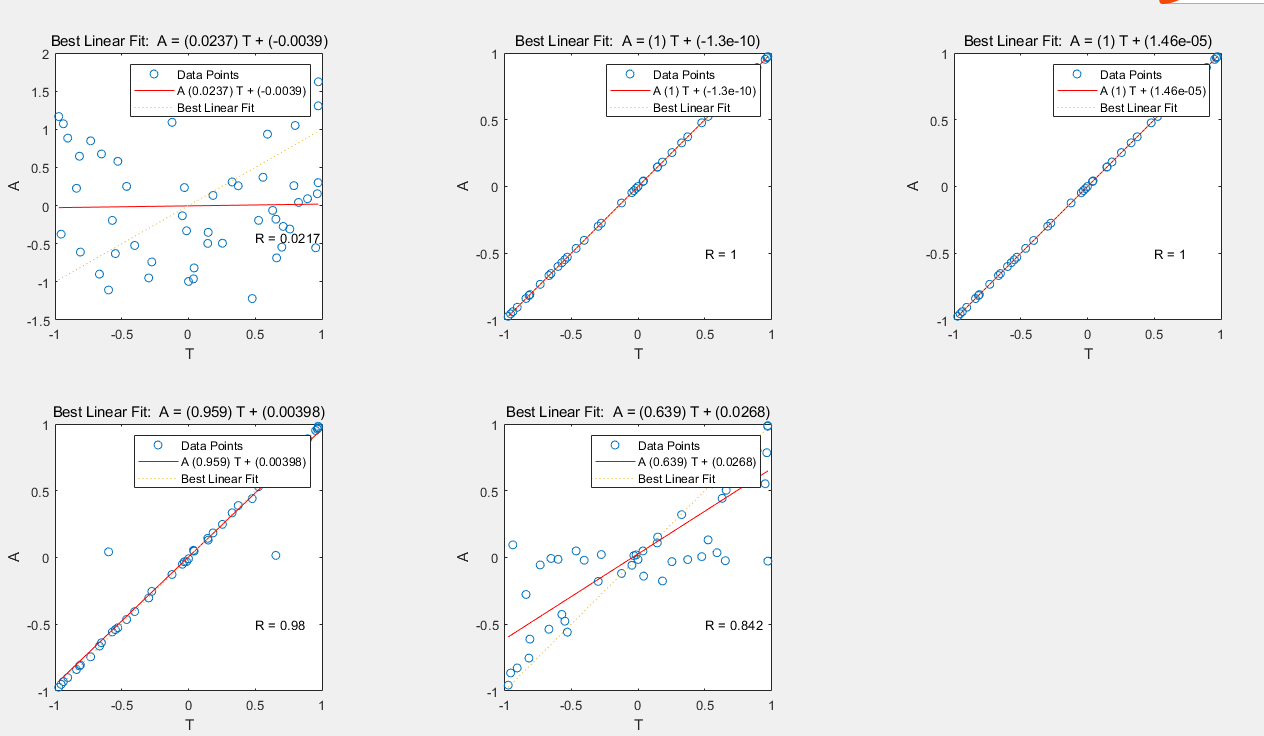


Figure1.2.1 Posterior performance of five different algorithms(noise level=0)

According to the plot and the result of training time,we can see that gradient descent algorithm trains the datapoints relatively slow compared with other algorithms.It is not the slowest but had the worst training effect that could be known from linear regression of the verification result and targets.

Levenberg - Marquardt and BFGS quasi Newton can obtain lower mean square error than other measured method.However, with the increase in the number of weights in the network, the advantages of LM will decrease. BFGS quasi Newton does not need as much storage space as LM, but the required computation does increase geometrically with the increase of network size, because the equivalent inverse matrix must be calculated for each iteration,so it is the slowest. Scaled conjugate gradient performed well, It is almost as fast as LM in function approximation. Bayesian Rule method modified LM to make the network generalization ability better.At the same time, the difficulty of determining the optimal network structure is reduced but increased the timing.

The linear fit became better when I increase the noise level of the dataset,so the reason is probably that adding the noise to the input layer can actually be regarded as a means of data set enhancement, which is essentially a kind of regularization.The reason is that the neural network is not robust to noise, so it has to be mixed with noise for training to improve its robustness. This method has been proved to be able to resist the attack against samples and improve the generalization ability of the model to some extent.

1 Epoch = 1 Forward pass + 1 Backward pass for ALL training samples. We can increase the number of epochs to 1000,and then use the 5 different algorithms to train the network again.From the training result,the timing of gradient descent algorithm and Bayesian Rule method increased substantially due to the highly incresed iterations(especially the gradient descent has 1000 iterations).However,for other algorithms,such as LM,the timing is almost same as before since its iteration haven’t changed correspondingly.

In a word,epoch is measuring the time of one iteration to assess the speed of the algorithms,on the other hand,timing is to assess the sum of the times of iterations.

**1.3 PERSONAL REGRESSION EXAMPLE**

My student number:r0816422

So d1=8,d2=6,d3=4,d4=2,d5=2;

1.First,I used *randperm* in MATLAB to randomly scramble an array(13600\*1).After that,I used the random ID of the array to draw the training set, validation set, and test set.The first 500 data points is given to the training set,then 501-750 points to validation set,and another 250 points to test set. We only use the Test Set to evaluate the performance of the model, not to adjust and optimize the model,so I choose the general proportion of the three is training/validation/test = 50/25/25. Validation sets are used to adjust the model parameters to select the optimal model,and test set is used to judge whether a model is good or bad.

Here is the surface of my training set.

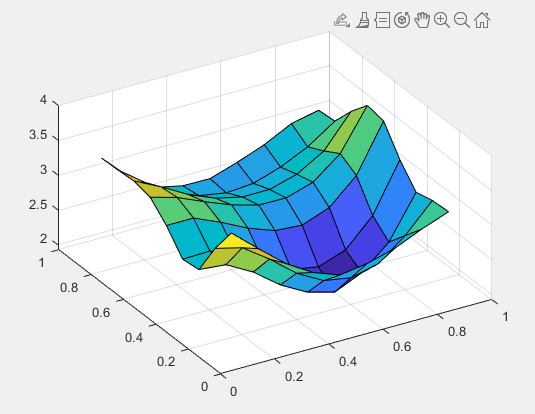
. 

Figure1.3.1 surface of the training set

2.One hidden layer and 30 neurons on the hidden layer would work best for this problem.Since the data is nonlinearly separated,we need only one hidden layer to fit any function that contains a continuous mapping from one finite space to another.The deeper layers may not only bring about the problem of overfitting, but also increase the difficulty of training and make the model difficult to converge.For the number of neurons on on the hidden layer,I refered to some empirical formula (:the number of input neurons, :the number of output nuurons, :the sample size of the training set,constraint of 2-10).For learning algorithm,I choose to use *learngdm* becauseit can improve the learning speed and increase the reliability of the algorithm compared *learngd* which I used at first time.For transfer function,I choose *logsig* for hidden layer and *purelin* for output layer since we are supposed to approximate a nonlinear function.

After training the neural network with the training set, I verified it with the validation set for several times to adjust parameters and determine the structure of the neural network,by Comparing training error and so on.

3. The performance of my selected network on the test set is quite good,and the error among most of the datapoints is almost zero.We can see that the Goodness of Fit equals 1.

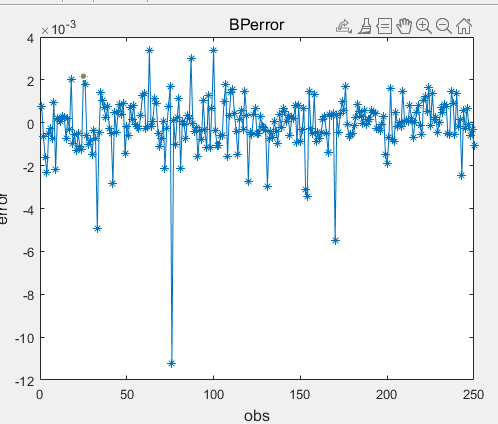
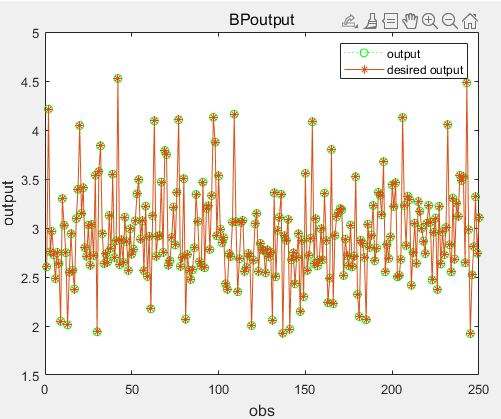


Figure1.3.2 comparison with desired output(approximation)

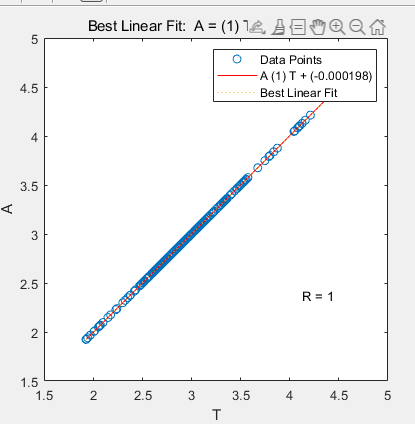
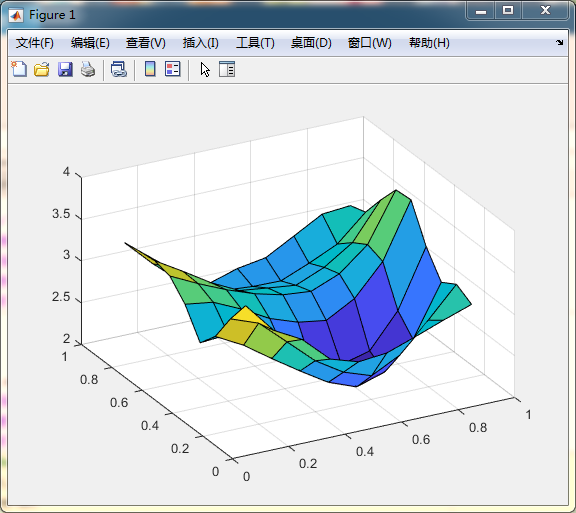


Figure1.3.3 surface of the test set and the linear regression

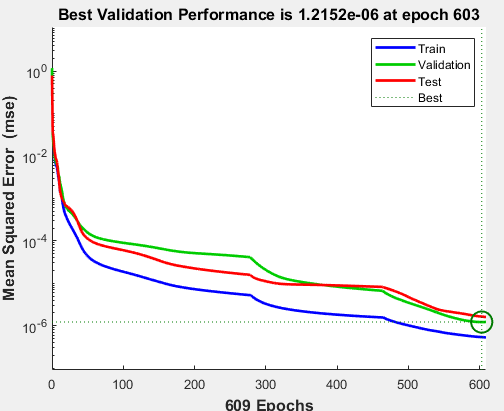


Figure1.3.4 the training,validation and test loss curves

From the plot,we can see both test and valldation values end up to be roughly the same and also are converging . Also,the validation loss is only a little bit larger than training loss,so we can call it fitting very well.(no overfitting or underfitting)

Final RMSE on the test set finally:0.0013.

To summarize,I chose 1 hidden layer and 30 neurons on the hidden layer in this problem to fit the nonlinear function and also avoided overfitting.And I choosed LM algorithm to train the network,which is faster than the Bayesian Rules algorithm that I have tested,and could get a relatively good result. To improve the performance of the network, I think we can also normalize the original data and so on.

**1.4 BAYESIAN INFERENCE**

1. After trying trainbr and other relevant training functions, I found that they differ not only in running time but also in operation precision. When I increased the number of neurons,Gradient descent became slowest and Scaled conjugate gradient and Bayesian rules method also worked slower.LM is still the fastest,but as the number of neurons increasing as well as the noise introduction, this algorithm has a tendency to overfit the model.

On the contrary,I could see an improvement with trainbr both on the fitting effect and stability.

2. The graphs below imply that the neural network trained by the trainlm function achieved "overfitting" for the sample data points, while the neural network trained by the trainbr function was not very sensitive to noise and had good robustness.

