General Guide

Fitting the MATLAB Logo Using Neural Networks

Table of Contents

Fitting the MATLAB Logo Using Neural Networks........1
1. Format Requirement........1
2. Data Visualization........2
3. Specify The Structure & Type of Neural Networks........3
4. Solve Least Square Problem By Numerical Optimization........5
   4.1 Stochastic Gradient Descent Based Method........5
   4.2 Quasi-Newton Method........5
5. Quatify & Visualize Fitting Performance........6
6. Train Neural Network Quickly with a Few Lines of Code........9

This live script demonstrates the workflow of using a neural network for regression. The first five sections provide

detailed usage instructions, while **the sixth section offers quick commands that allow users to solve fitting problems with just a few lines of code. (Read the first section to make sure the format of the data are correct.)**

1. Format Requirement

To represent neural network computations using linear algebra notation,

both the input data and corresponding labels should be column vectors.

For example: $f\left(x_1,x_2\right):R^2\rightarrow R^2=\left(x_1^2+x_2^2,2x_1^2+x_2^2\right)$

$$\boldsymbol{d}_1=\begin{bmatrix}1\\1\end{bmatrix},\boldsymbol{d}_2=\begin{bmatrix}1\\2\end{bmatrix},\ \cdots,\boldsymbol{d}_N=\begin{bmatrix}x_{1N}\\x_{2N}\end{bmatrix}$$

$$\boldsymbol{y}_1=\begin{bmatrix}1\\3\end{bmatrix},\boldsymbol{y}_2=\begin{bmatrix}5\\6\end{bmatrix},\ \cdots,\boldsymbol{y}_N=\begin{bmatrix}x_{1N}^2+x_{2N}^2\\2x_{1N}^2+x_{2N}^2\end{bmatrix}$$

data : $\boldsymbol{D}\in R^{n_D\times N}$ ($n_D\times N$ Matrix), where n_D is the dimension of data and N is the number of data points.

label : $\boldsymbol{Y}\in R^{n_L\times N}$ ($n_L\times N$ Matrix), where n_L is the dimension of label.

$$\boldsymbol{D}=\begin{bmatrix}1&1&\cdots&x_{1N}\\1&2&\cdots&x_{2N}\end{bmatrix},\ \boldsymbol{Y}=\begin{bmatrix}1&5&\cdots&x_{1N}^2+x_{2N}^2\\3&6&\cdots&2x_{1N}^2+x_{2N}^2\end{bmatrix}$$

Note that, If your data have the following property, normalize the input or output data before optimization.

1. There is a huge difference in magnitudes of x, y. i.e.

$x_1 \in [-1, 1], x_2 \in [1000, 2000]$

2. The output data are almost the same.

$y = 1 + 0.0001\sin(x_1 + x_2) \approx 1$

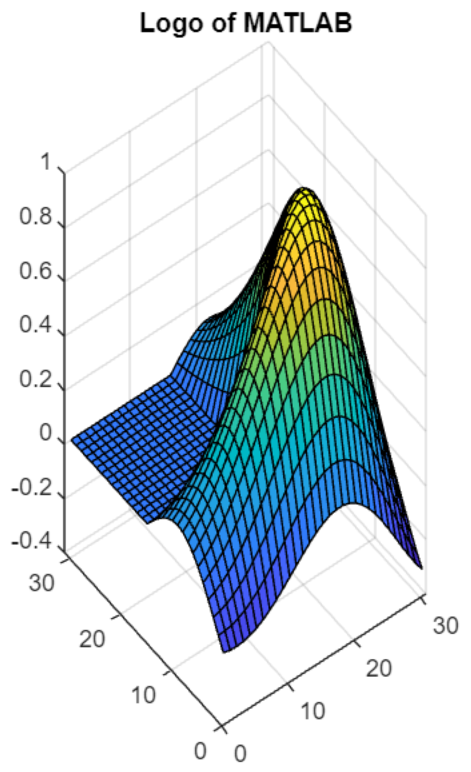
If you are not familiar normalization, you may also use built-in command in section 3

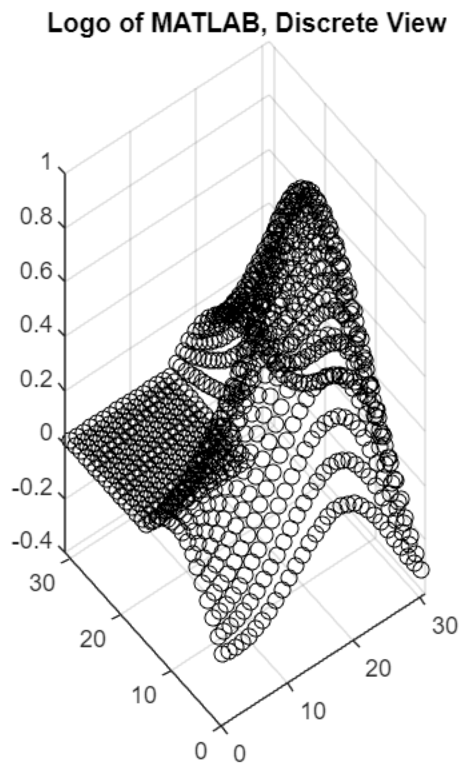
to perform basic normalization method automatically.

```
% Generate Data
clear; clc; close all;
n=15;
x=1:2*n+1; y=1:2*n+1;
[X,Y]=meshgrid(x,y);
V=membrane(1,n);
data=[X(:),Y(:)]';
label=V(:)';
```

2. Data Visualization

```
GridNumber=numel(x);
figure
subplot(1,2,1)
surf(X,Y,reshape(label,GridNumber,GridNumber))
title('Logo of MATLAB')
subplot(1,2,2)
scatter3(data(1,:),data(2,:),label,'black')
title('Logo of MATLAB, Discrete View')
```





Our goal is to find a surface that represents the input-output relation between the data and label.

3. Specify The Structure & Type of Neural Networks

The package support two types of Neural Nets for function approximation & data fitting,

the first is Multilayer Perceptron (MLP) , the second is Residual Neural Networks (ResNet).

ResNet can make training neural networks easier, the effectiveness is particularly

evident when the network is very deep.

To specify the type of network, use the command: `NN.NetworkType='ANN'` or `'ResNet'`,

```
close all
NN.NetworkType='ANN'; % MLP
```

The logo of MATLAB is a function with 2 input 1 output, i.e. $f(\boldsymbol{x}) : R^2 \to R$

Define the network structure using following command:

```
InputDimension=2; OutputDimension=1;
LayerStruct=[InputDimension,8,8,8,OutputDimension];
```

The number between Input & OutputDimension are the size of hidden layers,

larger hidden layer sizes and deeper networks lead to better approximation of complex functions.

To specify the cost function to minimize using `NN.Cost`, you can choose between

three options provided by the package, which are Sum of Square Error (SSE), Mean Square Error (MSE, suitable for

data fitting without noise and function approximation) and Mean Absolute Error (MAE, suitable for noisy data fitting)

$$E(W,b)=Y-\mathbf{NN}(W,b,D),\ J_{\mathrm{MSE}}(W,b)\equiv\frac{\|E(W,b)\|^2}{n_D},J_{\mathrm{MAE}}(W,b)\equiv\frac{|E(W,b)|}{n_D}$$

```
NN.Cost='SSE';
```

You can specify the nonlinear activation function for your neural network using `NN.ActivationFunction`.

Currently, there are four built-in activation functions available, which are:

1. Gaussian

2. Sigmoid

3. tanh

4. ReLU

The recommended activation function for function approximation is `'Gaussian'`.

```
NN.ActivationFunction='Gaussian';
```

You can also use a function handle to define custom activation function.

When using function handles to define custom activation, remember to provide the derivatives as well.

```
% NN.ActivationFunction=@(x) exp(-x.^2);
% NN.activeDerivate=@(x) -2*x.*exp(-x.^2); % derivative of gaussian
```

The activation should be smooth for function approximation problems, so those commonly used activation for

classification problems like 'ReLU' are not recommand.

For ill-behaved datasetsill-behaved datasets metioned in section 1, preprocessing of the data before optimization is necessary.

Alternatively, you can use the built-in function to automatically perform preprocessing (normailzation).

```
% NN.InputAutoScaling='on';
% NN.LabelAutoScaling='on';
```

Detailed explanation can be found in **"`TipsForTrainingNeuralNet.mlx`"**.

Finally, use the command 'Initialization' to Initialize the weights & biases of NN.

```
NN=Initialization(LayerStruct,NN);
```

4. Solve Least Square Problem By Numerical Optimization

Our goal is to find optimal parameters W^*, b^* such that minimize the cost function $J(W, b)$, i.e.

$$W^*, b^* = \text{argmin} \ \|Y - \mathbf{NN}(W, b, D)\|^2$$

The standard workflow can be separated into two stages. First use stochastic gradient descent-based methods (SGD)

to avoid getting stuck in local minima. For SGD-based algorithms, the following information is required:

4.1 Stochastic Gradient Descent Based Method

```
option.Solver='ADAM';
option.BatchSize=100;
option.MaxIteration=20;
option.s0=2e-3; % Step Size
```

Start optimizing using Adaptive Momentum Estimation. (ADAM)

```
NN=OptimizationSolver(data,label,NN,option);
```

```
Iteration : 1 , Cost :      87.40805614
Iteration : 2 , Cost :      84.50008922
Iteration : 3 , Cost :      82.17817358
Iteration : 4 , Cost :      78.75882006
Iteration : 5 , Cost :      74.65420815
Iteration : 6 , Cost :      69.68327716
Iteration : 7 , Cost :      63.72717552
Iteration : 8 , Cost :      57.21862755
Iteration : 9 , Cost :      51.32185497
Iteration : 10 , Cost :      45.40799044
Iteration : 11 , Cost :      40.79798658
Iteration : 12 , Cost :      37.04029401
Iteration : 13 , Cost :      34.53779423
Iteration : 14 , Cost :      32.19926879
Iteration : 15 , Cost :      30.81699083
Iteration : 16 , Cost :      29.29917356
Iteration : 17 , Cost :      28.43830068
Iteration : 18 , Cost :      27.16526463
Iteration : 19 , Cost :      26.89070376
Iteration : 20 , Cost :      25.74577724
------------------------------------------------------
Max Iteration : 20 , Cost :      25.74577724
Optimization Time :   0.2
Mean Absolute Error :   0.1191
------------------------------------------------------
```

4.2 Quasi-Newton Method

Quasi-Newton methods are much superior to SGD-Based methods (superlinear v.s. linear convergence rate)

in small-to-medium-size least-square problems, using Quasi-Newton methods can significantly accelerate the

optimization.

The following information are required when using Quasi-Newton's methods.

If the infinity norm of the gradient is less than terminate condition, the optimization process will be stop.

```
option.Solver='BFGS';
option.MaxIteration=550;
option.TerminateCondition=1e-6; % optional
```

Start optimizing using Broyden-Fletcher-Goldfarb-Shanno method (BFGS).

```
NN=OptimizationSolver(data,label,NN,option);
```

```
Iteration : 27 , Cost :      18.83059742
Iteration : 54 , Cost :       2.23181620
Iteration : 81 , Cost :       1.39115782
Iteration : 108 , Cost :       0.54360235
Iteration : 135 , Cost :       0.27712449
Iteration : 162 , Cost :       0.11862682
Iteration : 189 , Cost :       0.08898759
Iteration : 216 , Cost :       0.07322077
Iteration : 243 , Cost :       0.05508333
Iteration : 270 , Cost :       0.04261210
Iteration : 297 , Cost :       0.03177240
Iteration : 324 , Cost :       0.02447483
Iteration : 351 , Cost :       0.01963213
Iteration : 378 , Cost :       0.01590498
Iteration : 405 , Cost :       0.01294244
Iteration : 432 , Cost :       0.01137768
Iteration : 459 , Cost :       0.01004947
Iteration : 486 , Cost :       0.00853198
Iteration : 513 , Cost :       0.00758698
Iteration : 540 , Cost :       0.00693324
------------------------------------------------------
Max Iteration : 550 , Cost :       0.00673838
Optimization Time :    2.5
Mean Absolute Error :   0.0018
------------------------------------------------------
```

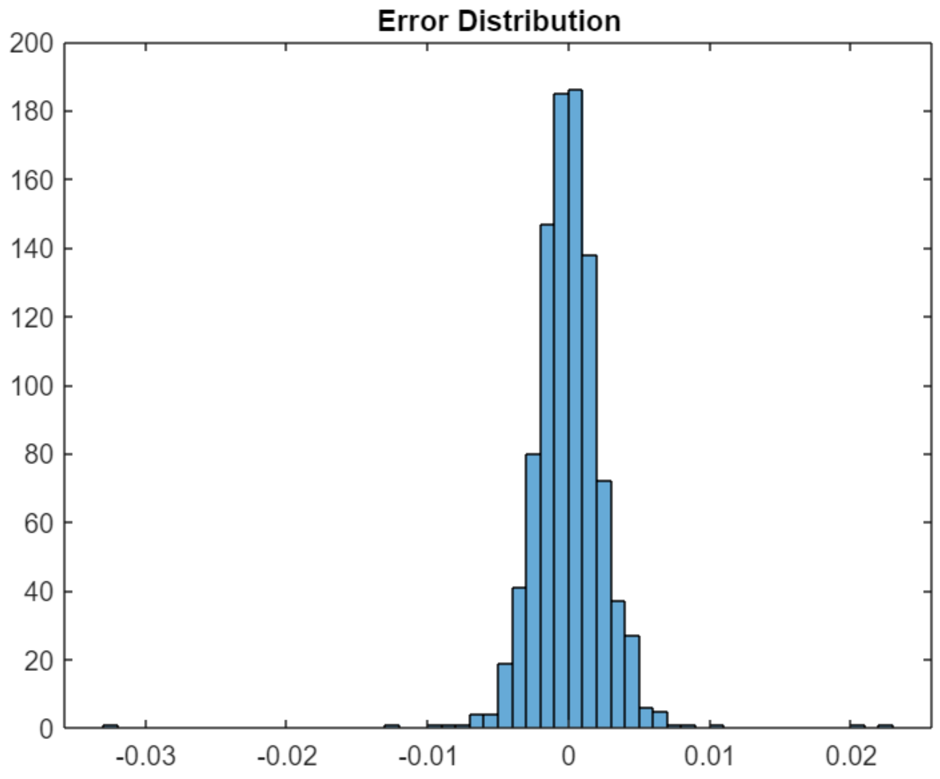
5. Quatify & Visualize Fitting Performance

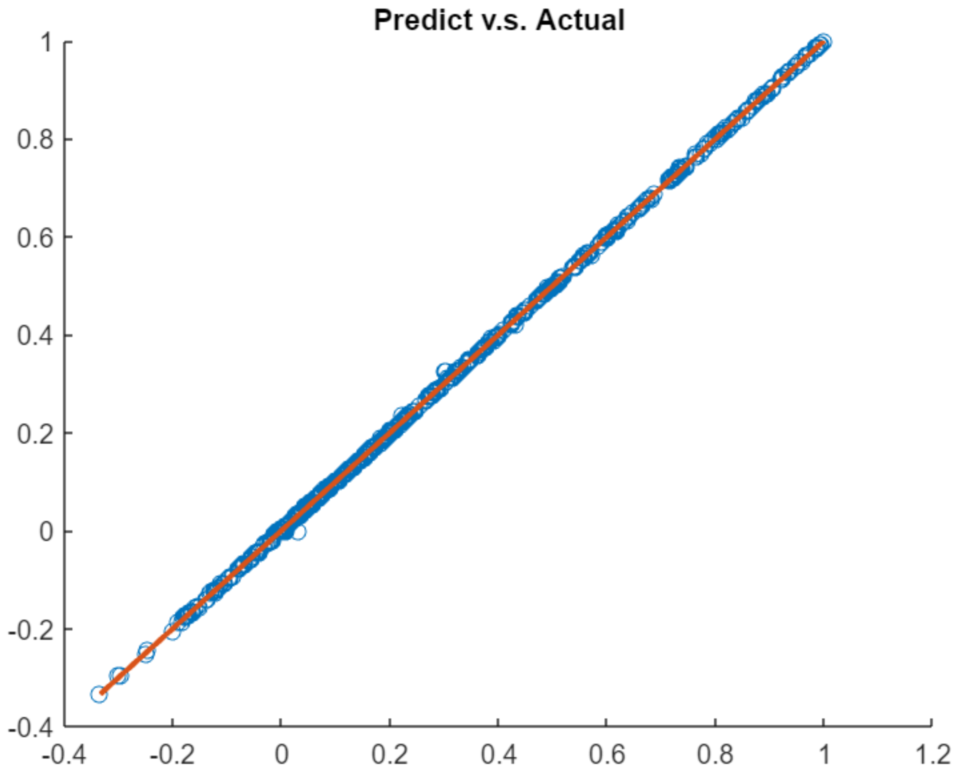
Use NN.Evaluate to predict & analyzing fitting results.

you can quatify the fitting performance by the error matrix : $\boldsymbol{E} = \boldsymbol{Y} - \mathrm{NN}(\boldsymbol{D})$

or use the function "`FittingReport`" to quantify and visualize the performance rapidly.

```
Prediction=NN.Evaluate(data);
Report=FittingReport(data,label,NN);
```

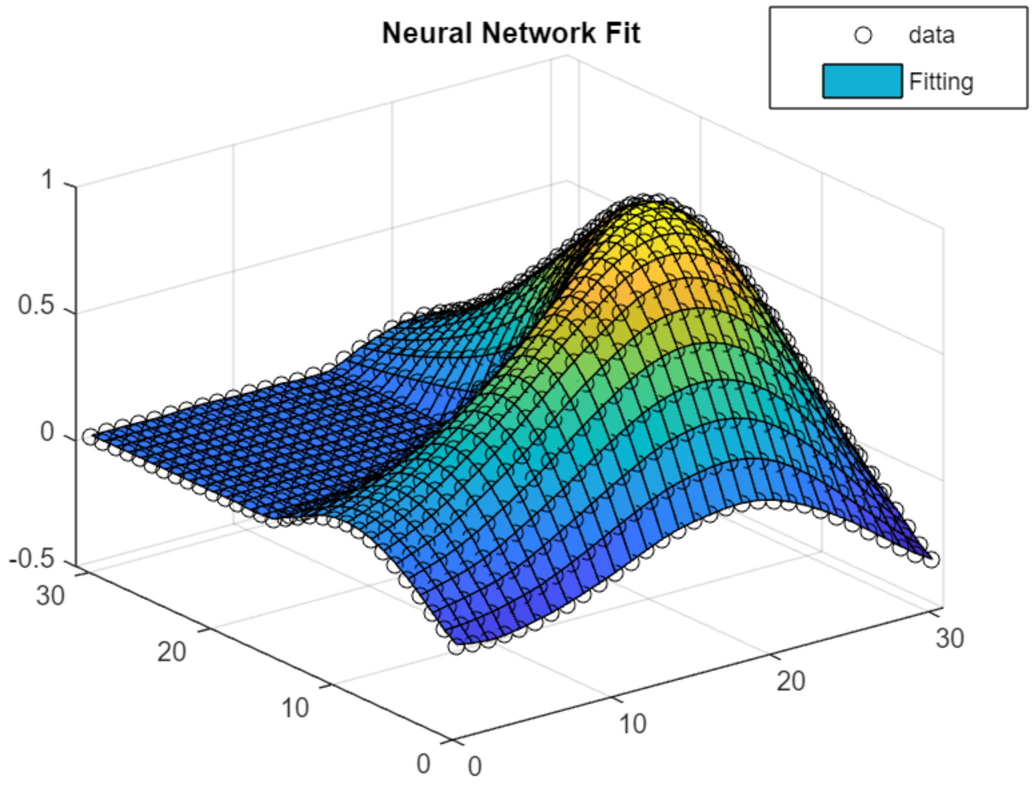




Visualize the optimized neural network

```
figure
```

```
scatter3(data(1,:),data(2,:),label(1,:),'black')
hold on
[X,Y]=meshgrid(x,y);
n1=numel(x); n2=numel(y);
surf(X,Y,reshape(Prediction(1,:),n1,n2))
title('Neural Network Fit')
legend('data','Fitting')
legend("Position", [0.74834,0.86095,0.23184,0.12127])
```



```
FirstOrderOptimality=NN.FirstOrderOptimality
```

```
FirstOrderOptimality = 0.6019
```

The first order optimality may indicate the fitting potential of neural net, if the first order optimality is still large after optimization, you may consider using the BFGS method for further optimization of the neural network.

```
option.Solver='BFGS';
option.MaxIteration=100;
NN=OptimizationSolver(data,label,NN,option);
```

```
Iteration : 5 , Cost :       0.00673572
Iteration : 10 , Cost :       0.00673425
Iteration : 15 , Cost :       0.00673284
Iteration : 20 , Cost :       0.00673109
Iteration : 25 , Cost :       0.00672719
Iteration : 30 , Cost :       0.00672381
Iteration : 35 , Cost :       0.00672035
Iteration : 40 , Cost :       0.00671383
Iteration : 45 , Cost :       0.00670647
Iteration : 50 , Cost :       0.00669513
Iteration : 55 , Cost :       0.00668419
Iteration : 60 , Cost :       0.00667588
Iteration : 65 , Cost :       0.00666669
Iteration : 70 , Cost :       0.00665547
Iteration : 75 , Cost :       0.00664182
Iteration : 80 , Cost :       0.00659266
Iteration : 85 , Cost :       0.00653701
Iteration : 90 , Cost :       0.00648541
Iteration : 95 , Cost :       0.00638918
Iteration : 100 , Cost :       0.00629547
------------------------------------------------------
Max Iteration : 100 , Cost :       0.00629547
Optimization Time :   0.4
Mean Absolute Error :   0.0017
------------------------------------------------------
```

```
clear option; % clear solver setting for next section.
```

6. Train Neural Network Quickly with a Few Lines of Code

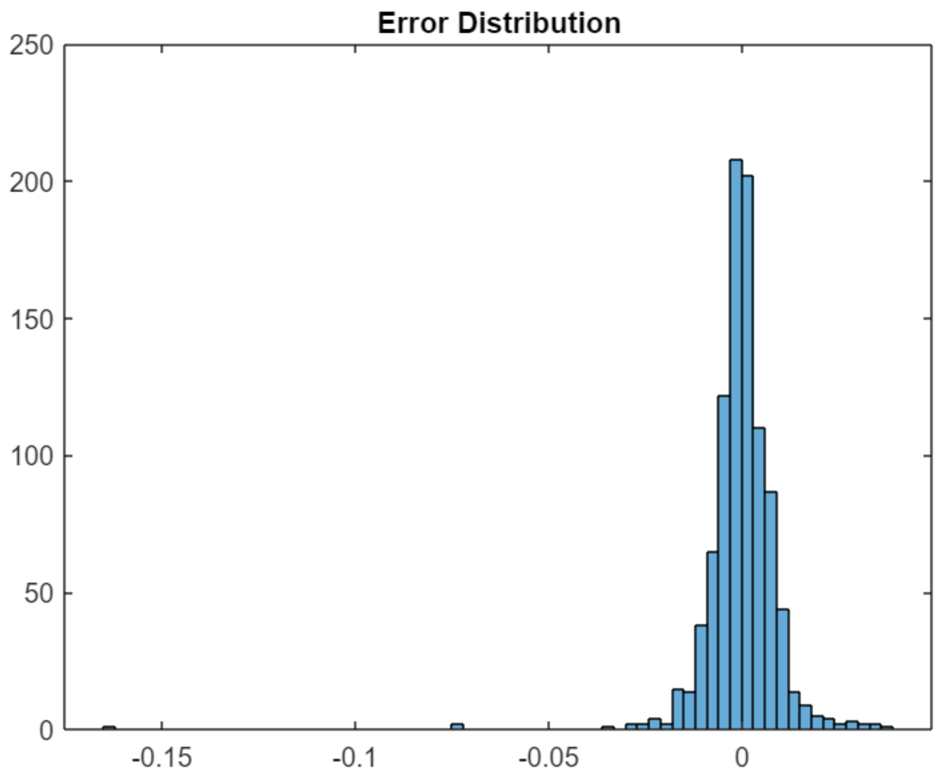
If the user did not provide solver options, the pack will perform the two-stage optimization automatically, that is, using 'ADAM' to find better initial parameters for the Quasi-Newton method, than applying 'BFGS' solver to accelerate convergence.

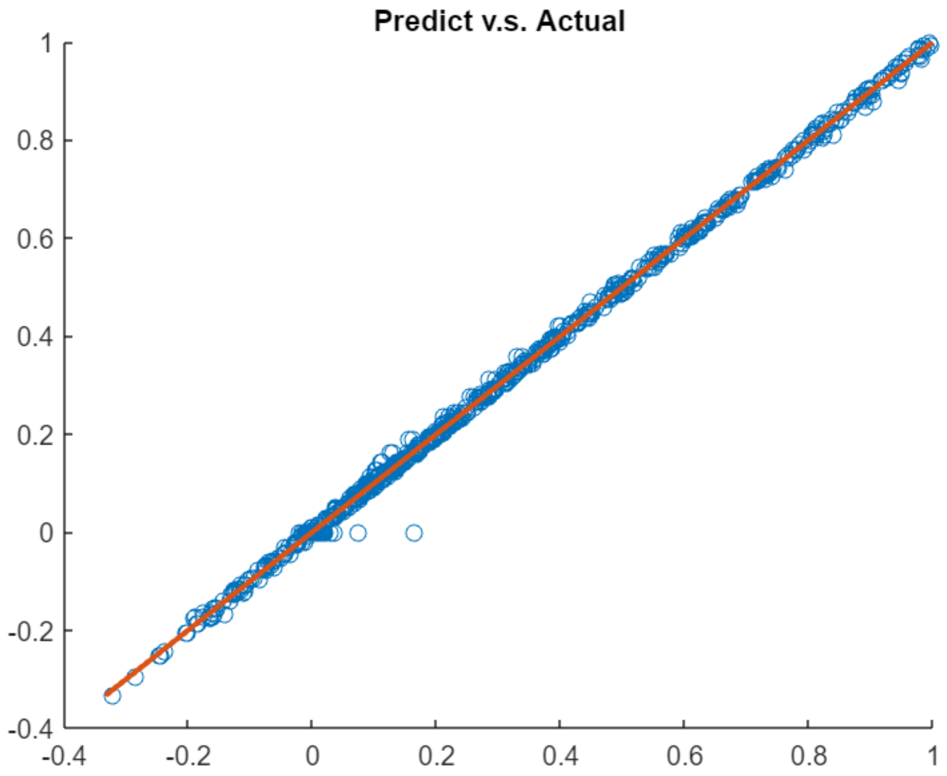
```
% Structure Set Up
InputDimension=2; OutputDimension=1;
LayerStruct=[InputDimension,7,7,8,7,7,OutputDimension];
NN=Initialization(LayerStruct);
% Solver Set Up
option.MaxIteration=600;
NN=OptimizationSolver(data,label,NN,option);
```

```
Iteration : 7 , Cost :       0.17647171
Iteration : 14 , Cost :       0.10864026
Iteration : 21 , Cost :       0.09271308
Iteration : 28 , Cost :       0.08575096
Iteration : 35 , Cost :       0.08049857
Iteration : 42 , Cost :       0.07396149
Iteration : 49 , Cost :       0.07216257
Iteration : 56 , Cost :       0.06921750
Iteration : 63 , Cost :       0.06710528
Iteration : 70 , Cost :       0.06762928
Iteration : 77 , Cost :       0.06524048
Iteration : 84 , Cost :       0.06222377
Iteration : 91 , Cost :       0.06393177
Iteration : 98 , Cost :       0.06045880
Iteration : 105 , Cost :       0.05409182
Iteration : 112 , Cost :       0.05603869
Iteration : 119 , Cost :       0.03094352
Iteration : 126 , Cost :       0.02638018
Iteration : 133 , Cost :       0.02015182
Iteration : 140 , Cost :       0.01562922
Iteration : 147 , Cost :       0.01431934
------------------------------------------------------
First Stage Optimization Finished in  150  Iteration.
------------------------------------------------------
```

```
Iteration : 22 , Cost :       0.00885524
Iteration : 44 , Cost :       0.00560415
Iteration : 66 , Cost :       0.00332043
Iteration : 88 , Cost :       0.00265378
Iteration : 110 , Cost :       0.00182239
Iteration : 132 , Cost :       0.00139887
Iteration : 154 , Cost :       0.00115612
Iteration : 176 , Cost :       0.00090492
Iteration : 198 , Cost :       0.00077816
Iteration : 220 , Cost :       0.00063052
Iteration : 242 , Cost :       0.00049545
Iteration : 264 , Cost :       0.00045161
Iteration : 286 , Cost :       0.00039484
Iteration : 308 , Cost :       0.00036267
Iteration : 330 , Cost :       0.00031795
Iteration : 352 , Cost :       0.00025298
Iteration : 374 , Cost :       0.00023200
Iteration : 396 , Cost :       0.00021969
Iteration : 418 , Cost :       0.00020731
Iteration : 440 , Cost :       0.00019533
------------------------------------------------------
Max Iteration : 450 , Cost :       0.00019067
Optimization Time :   2.4
Mean Absolute Error :   0.0056
------------------------------------------------------
```

```
% Validate Results
Report=FittingReport(data,label,NN);
```




Predict v.s. Actual
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1