Reading data

Importing the data

```
my = readtable('weather_train.csv');
```

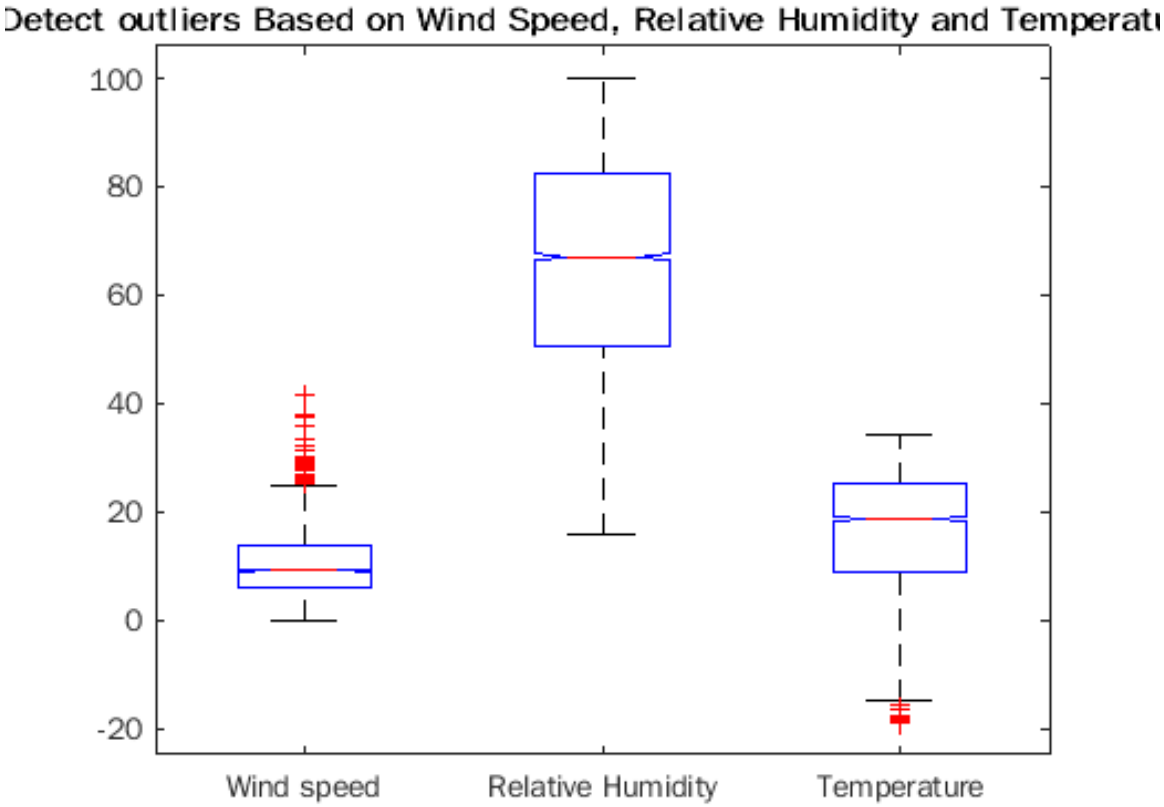
Data Preprocessing

Removing missing values and droping non relevent columns

```
final_table= removevars(my,{'Var1'});
final_table=final_table(~any(ismissing(final_table),2),:);
```

Detect and remove outliers

```
figure;
boxplot([final_table.Var10,final_table.Var9,final_table.Var7],'Notch','on','Labels',{'W
title('Detect outliers Based on Wind Speed, Relative Humidity and Temperature');
```




```
data = final_table.Var12;
threshold = 3 * std( data );
validRange = mean( data ) + [-1 1] * threshold;
[m,n] = size(final_table);
for i = i:m
    if data >= validRange(1) & data <= validRange(2)
        final_table(i,:) = [];
```

```
    end
end
```

```
Warning: Colon operands must be real scalars.
```

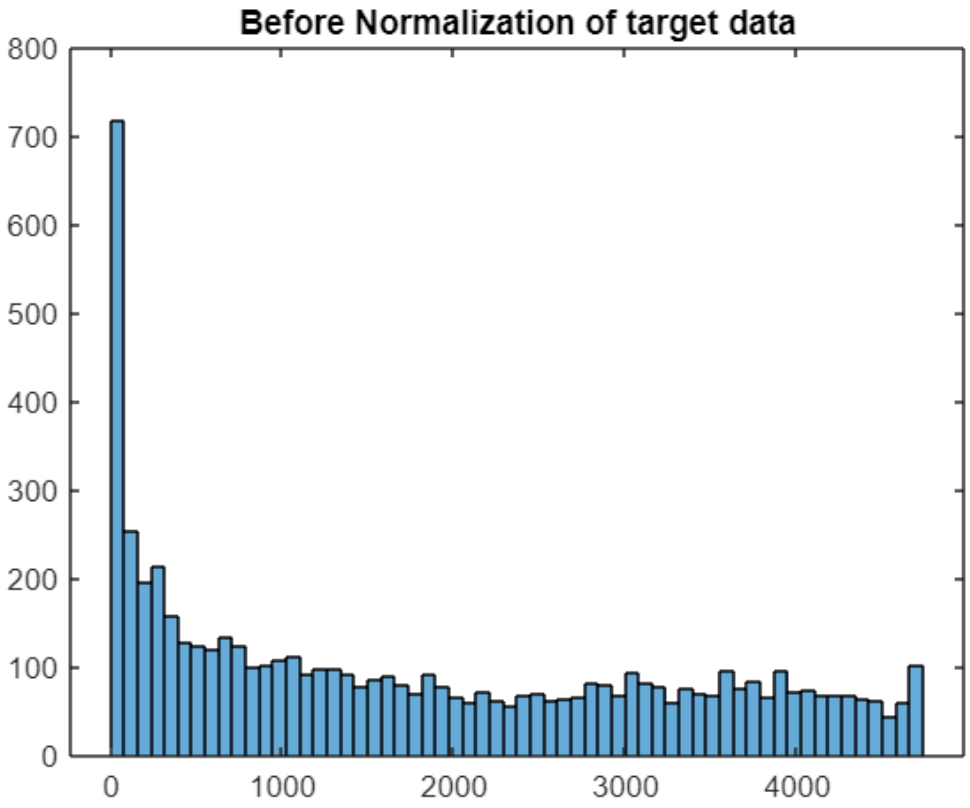
Spliting data as feature and target

```
X = table2array(final_table(:,1:n-1));
y = table2array(final_table(:,end));
```

Normization or Scaling

HIstogram visualization

```
histogram(y,60)
title('Before Normalization of target data')
```



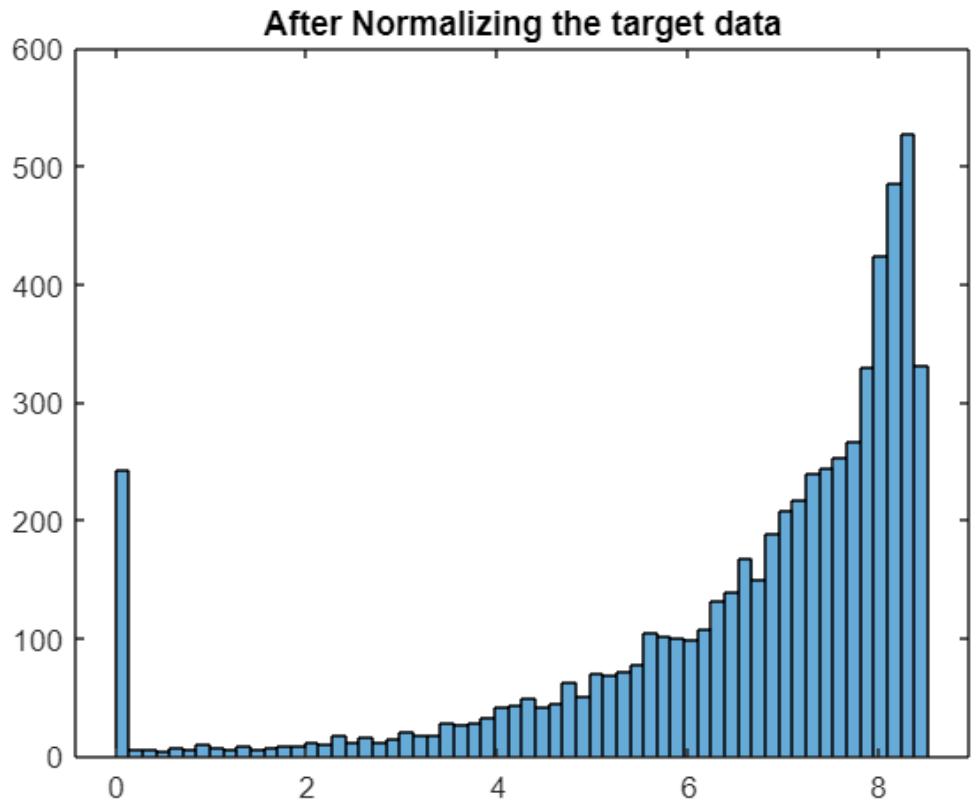
Normalize the Target and transform the output

```
y2 = log(1+y)
```

```
y2 = 6028×1
    6.1098
    8.0604
    7.8778
    6.5670
    6.5243
    5.6166
    8.4538
```

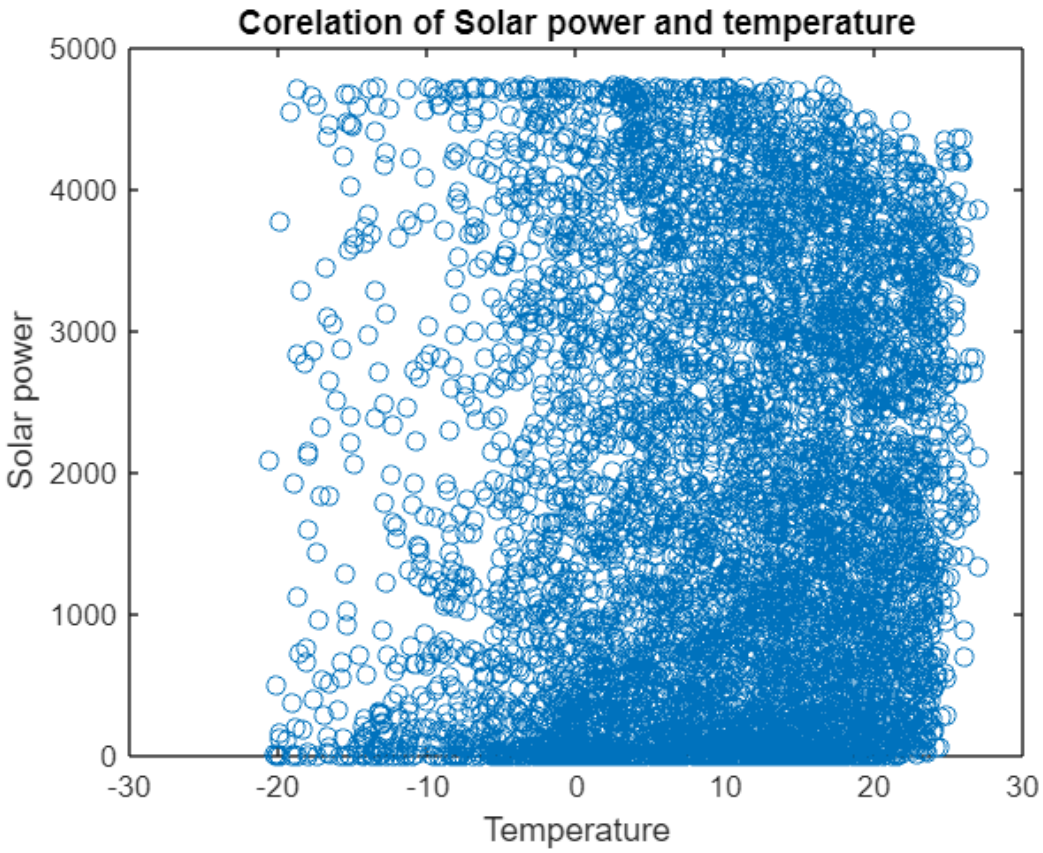
```
    8.0067
    8.0201
    5.6996
     ⋮
     ⋮
```

```
histogram(y2,60)
title('After Normalizing the target data')
```



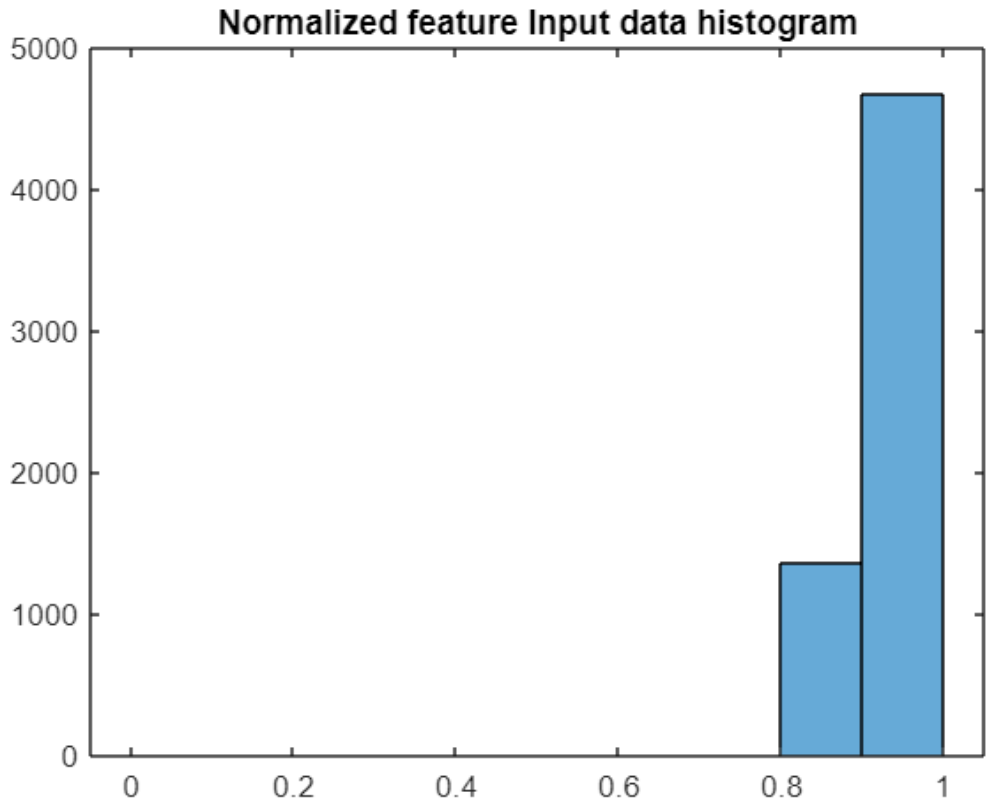
Correlation between solar power and temperature

```
plot(X(:,7),y,'o')
title('Corelation of Solar power and temperature');
xlabel('Temperature')
ylabel('Solar power')
```



Normalizing feature inputs

```
for i = 1:n-1
    X2(:,i) = (X(:,i) - min(X(:,i)))/max(X(:,i)-min(X(:,i)));
end
histogram(X2(:,11),10)
title('Normalized feature Input data histogram')
```



Artificial Neural Network

Model creation

```
xt = X2';
yt = y2';
hiddenLayerSize = 30;
net = fitnet(hiddenLayerSize);
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 30/100;
net.divideParam.testRatio = 0/100;
[net,tr] = train(net,xt,yt)
```

```
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 391
```

```
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
```

```
      divideFcn: 'dividerand'
    divideMode: 'sample'
   divideParam: [1×1 struct]
      trainInd: [1 3 4 5 6 8 10 11 12 13 14 15 16 17 18 19 20 21 22 23 27 28 29 33 34 35 36 37 38 39 40
        valInd: [2 7 9 24 25 26 30 31 32 53 54 55 60 69 71 73 83 85 90 95 97 101 107 109 111 123 125 132
       testInd: [1×0 double]
          stop: 'Training finished: Met validation criterion'
    num_epochs: 21
     trainMask: {[1 NaN 1 1 1 1 NaN 1 NaN 1 1 1 1 1 1 1 1 1 1 1 1 1 NaN NaN NaN 1 1 1 NaN NaN NaN 1 1
       valMask: {[NaN 1 NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
      testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
    best_epoch: 15
          goal: 0
        states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
         epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21]
          time: [6.7972 6.9333 7.0181 7.0799 7.1442 7.2142 7.2838 7.3489 7.4012 7.4509 7.5090 7.5642 7.6
          perf: [29.4889 21.3783 19.2583 15.5940 7.8107 4.6222 3.1845 2.8019 2.4587 2.3553 2.2406 2.1940
         vperf: [31.7186 21.5671 19.7271 15.5375 8.2144 4.9411 3.6314 3.2471 2.9859 2.8843 2.7612 2.7602
         tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
            mu: [1.0000e-03 1.0000e-04 1.0000e-05 1.0000e-06 1.0000e-07 1.0000e-08 1.0000e-09 1.0000e-06
      gradient: [87.5212 120.2398 85.0880 74.2559 30.6283 22.7885 10.3101 8.5650 5.4082 3.9769 2.7907 2.
      val_fail: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 2 3 4 5 6]
     best_perf: 2.0621
    best_vperf: 2.7067
    best_tperf: NaN
```

Performance

```
yTrain = exp(net(xt(:,tr.trainInd)))-1;
yTrainTrue = exp(yt(tr.trainInd))-1;

yVal = exp(net(xt(:,tr.valInd)))-1;
yValTrue = exp(yt(tr.valInd))-1;

Rmse_train = sqrt(mean(yTrain-yTrainTrue).^2)
```

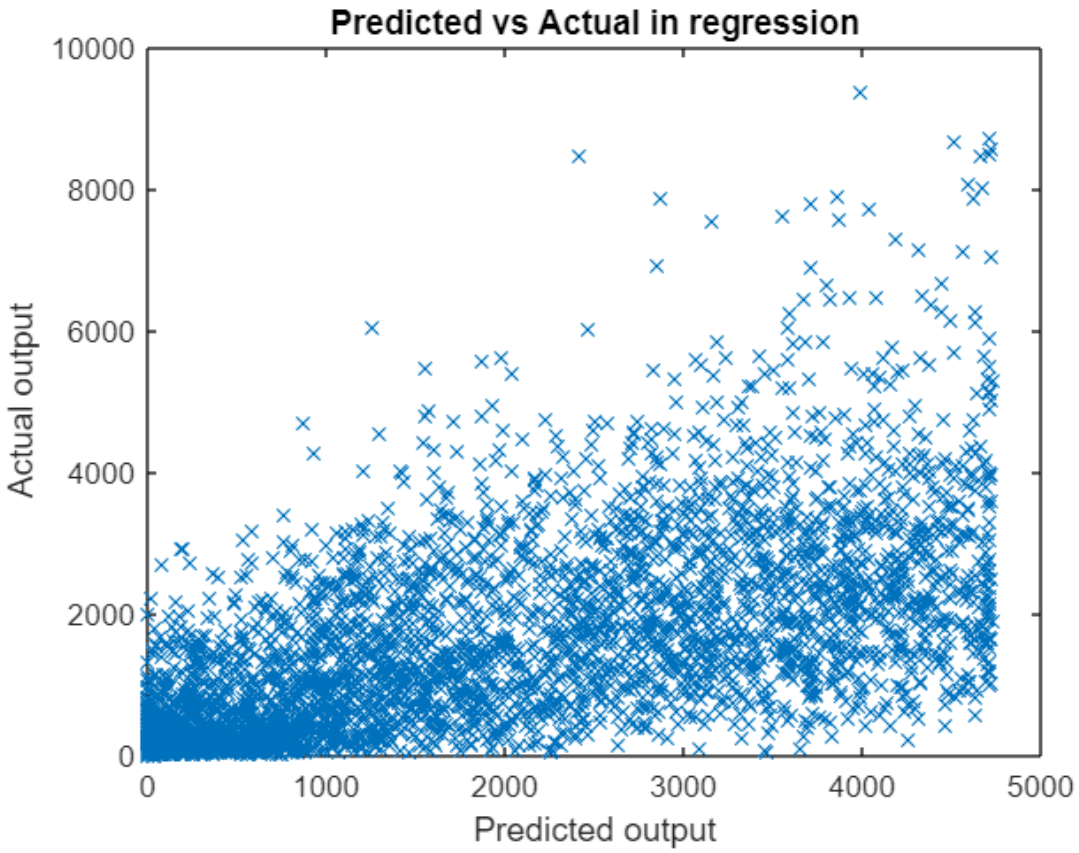
```
Rmse_train = 336.3740
```

```
Rmse_val = sqrt(mean(yVal-yValTrue).^2)
```

```
Rmse_val = 371.7834
```

Visualize the prediction from ANN model

```
plot(yTrainTrue,yTrain,'x')
xlabel('Predicted output')
ylabel('Actual output')
title('Predicted vs Actual in regression')
```



Optimize number of neuron of hidden size

```
for i=1:60
    hiddenLayerSize = i;
    net = fitnet(hiddenLayerSize);
    net.divideParam.trainRatio = 70/100;
    net.divideParam.valRatio = 30/100;
    net.divideParam.testRatio = 0/100;
    [net,tr] = train(net,xt,yt)

    yTrain = exp(net(xt(:,tr.trainInd)))-1;
    yTrainTrue = exp(yt(tr.trainInd))-1;

    yVal = exp(net(xt(:,tr.valInd)))-1;
    yValTrue = exp(yt(tr.valInd))-1;

    Rmse_train(i) = sqrt(mean(yTrain-yTrainTrue).^2)
    Rmse_val(i) = sqrt(mean(yVal-yValTrue).^2)

end
```

```
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)
```

```
   dimensions:

        numInputs: 1
        numLayers: 2
       numOutputs: 1
   numInputDelays: 0
   numLayerDelays: 0
numFeedbackDelays: 0
numWeightElements: 14
       sampleTime: 1

   connections:

      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

   subobjects:

            input: Equivalent to inputs{1}
           output: Equivalent to outputs{2}

           inputs: {1x1 cell array of 1 input}
           layers: {2x1 cell array of 2 layers}
          outputs: {1x2 cell array of 1 output}
           biases: {2x1 cell array of 2 biases}
     inputWeights: {2x1 cell array of 1 weight}
     layerWeights: {2x2 cell array of 1 weight}

   functions:

         adaptFcn: 'adaptwb'
       adaptParam: (none)
         derivFcn: 'defaultderiv'
        divideFcn: 'dividerand'
      divideParam: .trainRatio, .valRatio, .testRatio
       divideMode: 'sample'
          initFcn: 'initlay'
       performFcn: 'mse'
     performParam: .regularization, .normalization
         plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                   'plotregression', 'plotfit'}
       plotParams: {1x5 cell array of 5 params}
         trainFcn: 'trainlm'
       trainParam: .showWindow, .showCommandLine, .show, .epochs,
                   .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                   .mu_inc, .mu_max

   weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

   methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
```

```
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
    performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 4 5 8 10 11 12 13 14 15 16 17 19 21 25 27 28 29 31 32 33 34 35 36 37 38 39 40 41 43
            valInd: [3 6 7 9 18 20 22 23 24 26 30 42 44 45 46 48 50 51 54 60 62 65 66 88 90 96 104 105 109 1
           testInd: [1×0 double]
              stop: 'Training finished: Reached minimum gradient'
        num_epochs: 40
         trainMask: {[1 1 NaN 1 1 NaN NaN 1 NaN 1 1 1 1 1 1 1 NaN 1 NaN 1 NaN NaN NaN 1 NaN 1 1 1 NaN 1 1
           valMask: {[NaN NaN 1 NaN NaN 1 1 NaN 1 NaN NaN NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN 1 1 1 NaN 1 Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 40
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 3
              time: [0.2627 0.2817 0.2885 0.2978 0.3133 0.3285 0.3530 0.3586 0.3642 0.3721 0.3762 0.3800 0.3
              perf: [6.3459 4.0146 3.3580 2.7435 2.6338 2.5706 2.5567 2.5525 2.5513 2.5510 2.5509 2.5508 2.5
             vperf: [6.3335 4.0744 3.3794 2.6762 2.5762 2.4933 2.4758 2.4695 2.4665 2.4649 2.4635 2.4621 2.4
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
                mu: [1.0000e-03 1.0000e-03 1.0000e-04 0.0100 1.0000e-03 1.0000e-04 1.0000e-05 1.0000e-06 1.0
          gradient: [16.6973 1.4693 4.1451 2.6103 1.1016 0.0846 0.0445 0.0278 0.0105 0.0053 0.0043 0.0044 0.
          val_fail: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
         best_perf: 2.5462
        best_vperf: 2.4192
        best_tperf: NaN
Rmse_train = 559.6753
Rmse_val = 559.7966
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

           numInputs: 1
           numLayers: 2
          numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 27
          sampleTime: 1

    connections:

         biasConnect: [1; 1]
        inputConnect: [1; 0]
        layerConnect: [0 0; 1 0]
       outputConnect: [0 1]

    subobjects:
```

```
          input: Equivalent to inputs{1}
         output: Equivalent to outputs{2}

         inputs: {1x1 cell array of 1 input}
         layers: {2x1 cell array of 2 layers}
        outputs: {1x2 cell array of 1 output}
         biases: {2x1 cell array of 2 biases}
   inputWeights: {2x1 cell array of 1 weight}
   layerWeights: {2x2 cell array of 1 weight}

  functions:

       adaptFcn: 'adaptwb'
     adaptParam: (none)
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
    divideParam: .trainRatio, .valRatio, .testRatio
     divideMode: 'sample'
        initFcn: 'initlay'
     performFcn: 'mse'
   performParam: .regularization, .normalization
       plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                  'plotregression', 'plotfit'}
     plotParams: {1x5 cell array of 5 params}
       trainFcn: 'trainlm'
     trainParam: .showWindow, .showCommandLine, .show, .epochs,
                 .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                 .mu_inc, .mu_max

  weight and bias values:

              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

  methods:

           adapt: Learn while in continuous use
       configure: Configure inputs & outputs
          gensim: Generate Simulink model
            init: Initialize weights & biases
         perform: Calculate performance
             sim: Evaluate network outputs given inputs
           train: Train network with examples
            view: View diagram
     unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
       trainFcn: 'trainlm'
     trainParam: [1×1 struct]
     performFcn: 'mse'
   performParam: [1×1 struct]
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
     divideMode: 'sample'
    divideParam: [1×1 struct]
       trainInd: [1 2 4 5 6 7 8 9 11 12 14 15 16 18 19 20 22 23 24 25 26 27 28 29 30 31 33 35 36 37 38 40
         valInd: [3 10 13 17 21 32 34 39 41 44 45 47 49 51 56 60 63 67 70 72 73 76 82 84 86 87 97 98 100
        testInd: [1×0 double]
           stop: 'Training finished: Met validation criterion'
     num_epochs: 16
      trainMask: {[1 1 NaN 1 1 1 1 1 1 NaN 1 1 NaN 1 1 1 NaN 1 1 1 NaN 1 1 1 1 1 1 1 1 1 NaN 1 NaN 1 1
        valMask: {[NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN
       testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
```

```
      best_epoch: 10
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
            time: [0.0469 0.1064 0.1146 0.1213 0.1302 0.1378 0.1462 0.1511 0.1630 0.1683 0.1767 0.1823 0.1
            perf: [14.6485 8.1532 2.9598 2.5971 2.5917 2.5532 2.5321 2.5179 2.4993 2.4960 2.4929 2.4914 2.
           vperf: [14.1665 8.7803 2.7140 2.4217 2.4305 2.3603 2.3612 2.3529 2.3634 2.3545 2.3510 2.3528 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-05 1.0000e-06 1.0000e-05 1.0000e-05 1.0000e-04 1.0000e-05
        gradient: [45.3496 27.8037 2.5028 0.7306 1.3252 0.3452 0.2168 0.4039 0.1836 0.0551 0.0259 0.0150 0
        val_fail: [0 0 0 0 1 0 1 0 1 2 0 1 2 3 4 5 6]
       best_perf: 2.4929
      best_vperf: 2.3510
      best_tperf: NaN
Rmse_train = 1×2
  559.6753  541.1303
Rmse_val = 1×2
  559.7966  540.3455
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 40
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
```

```
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [3 4 5 7 9 10 11 15 17 18 20 21 22 23 24 25 27 28 29 31 34 35 36 37 39 41 42 43 44 45 46
            valInd: [1 2 6 8 12 13 14 16 19 26 30 32 33 38 40 47 50 51 53 56 58 62 63 66 68 69 72 76 78 87 8
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 19
         trainMask: {[NaN NaN 1 1 1 NaN 1 NaN 1 1 1 NaN NaN NaN 1 NaN 1 1 NaN 1 1 1 1 1 1 NaN 1 1 1 NaN 1 Na
           valMask: {[1 1 NaN NaN NaN 1 NaN 1 NaN NaN NaN 1 1 1 NaN 1 NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 13
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]
              time: [0.0339 0.0875 0.0960 0.1044 0.1121 0.1190 0.1259 0.1316 0.1382 0.1480 0.1557 0.1640 0.1
              perf: [24.8739 4.5962 4.5161 3.9548 3.4713 2.6706 2.6016 2.5948 2.5755 2.5635 2.5596 2.5595 2.
             vperf: [25.6197 4.4431 4.4637 3.9822 3.4451 2.4938 2.3595 2.3662 2.3443 2.3286 2.3353 2.3247 2.
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
                mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-05 1.0000e-06 1.0000e-07 1.0000e-08
          gradient: [64.4430 7.7971 10.2107 11.9487 8.8597 2.1037 0.8529 1.0142 0.2713 0.3176 0.0585 0.2746
          val_fail: [0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 2 3 4 5 6]
         best_perf: 2.5479
        best_vperf: 2.3186
        best_tperf: NaN
Rmse_train = 1×3
  559.6753  541.1303  539.4591
Rmse_val = 1×3
  559.7966  540.3455  581.9837
```

```
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 53
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:
```

```
             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
         trainInd: [1 2 4 5 6 7 8 9 11 12 13 14 15 16 18 20 21 22 26 27 28 30 32 33 34 35 36 37 38 40 41 42
           valInd: [3 10 17 19 23 24 25 29 31 39 45 48 50 54 57 61 62 64 66 70 73 80 87 91 92 96 109 110 11
          testInd: [1×0 double]
             stop: 'Training finished: Met validation criterion'
       num_epochs: 15
        trainMask: {[1 1 NaN 1 1 1 1 1 1 NaN 1 1 1 1 1 1 NaN 1 NaN 1 1 1 NaN NaN NaN 1 1 1 NaN 1 NaN 1 1 1
          valMask: {[NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN 1 1 1
         testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
       best_epoch: 9
             goal: 0
           states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
            epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
             time: [0.2306 0.2653 0.2775 0.2869 0.2956 0.3072 0.3168 0.3244 0.3328 0.3446 0.3527 0.3620 0.3
             perf: [101.5232 4.0456 3.7911 2.8835 2.7491 2.5565 2.5090 2.4650 2.4504 2.4382 2.4322 2.4245 2
            vperf: [102.7605 4.1437 3.6396 2.7547 2.5970 2.4422 2.3730 2.3699 2.3347 2.3259 2.3409 2.3335 2
            tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
               mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-04
         gradient: [166.1378 6.0014 7.7409 4.3631 1.4713 2.0778 1.5755 1.2097 0.2690 0.2862 1.6304 0.9770 0
         val_fail: [0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
        best_perf: 2.4382
       best_vperf: 2.3259
       best_tperf: NaN
Rmse_train = 1×4
  559.6753  541.1303  539.4591  460.8964
Rmse_val = 1×4
  559.7966  540.3455  581.9837  488.1037
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 66
        sampleTime: 1

    connections:
```

```
       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
```

```
        trainInd: [2 4 5 7 8 9 10 11 12 13 14 16 17 18 19 22 23 24 25 27 28 29 30 31 32 35 36 37 38 39 40
          valInd: [1 3 6 15 20 21 26 33 34 41 42 43 44 45 51 55 56 60 63 64 65 69 71 80 82 86 90 91 98 100
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 31
       trainMask: {[NaN 1 NaN 1 1 NaN 1 1 1 1 1 1 1 NaN 1 1 1 1 NaN NaN 1 1 1 1 NaN 1 1 1 1 1 1 NaN NaN
         valMask: {[1 NaN 1 NaN NaN 1 NaN NaN NaN NaN NaN NaN NaN NaN 1 NaN NaN NaN NaN 1 1 NaN NaN NaN Na
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 25
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31]
            time: [10.5966 10.6155 10.6280 10.6391 10.6520 10.6677 10.6787 10.6911 10.7006 10.7174 10.7271
            perf: [12.4753 8.6995 6.5082 6.4594 3.8323 3.3908 2.6194 2.4579 2.3814 2.3455 2.3430 2.3365 2.
           vperf: [12.6320 8.7331 6.4198 6.7290 3.8447 3.4488 2.6801 2.5774 2.4562 2.4402 2.4410 2.4266 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
              mu: [1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-04 1.0000e-04
        gradient: [31.1657 32.5082 23.8976 11.9628 10.8146 8.1643 2.0748 1.8719 0.6720 0.7151 0.3872 0.825
        val_fail: [0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 2 3 0 1 2 3 4 5 6]
       best_perf: 2.2718
      best_vperf: 2.3676
      best_tperf: NaN
Rmse_train = 1×5
  559.6753  541.1303  539.4591  460.8964  417.2784
Rmse_val = 1×5
  559.7966  540.3455  581.9837  488.1037  449.9189
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 79
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:
```

```
          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 4 6 7 8 10 11 16 19 21 22 23 25 26 27 28 29 30 31 35 36 37 38 40 41 42 43 45 46 47 48
            valInd: [2 3 5 9 12 13 14 15 17 18 20 24 32 33 34 39 44 49 50 51 52 53 54 55 62 66 70 75 76 79 8
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 21
         trainMask: {[1 NaN NaN 1 NaN 1 1 1 NaN 1 1 NaN NaN NaN NaN 1 NaN NaN 1 NaN 1 1 1 NaN 1 1 1 1 1 1 1
           valMask: {[NaN 1 1 NaN 1 NaN NaN NaN 1 NaN NaN 1 1 1 1 NaN 1 1 NaN 1 NaN NaN NaN 1 NaN NaN NaN Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 15
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21]
              time: [0.0830 0.1279 0.1432 0.1553 0.1713 0.1855 0.2017 0.2140 0.2262 0.2390 0.2528 0.2650 0.2
              perf: [20.1165 4.6264 4.5873 4.0545 3.0686 2.9341 2.5194 2.5174 2.4755 2.4544 2.4116 2.3887 2.
             vperf: [20.2421 4.3315 4.5574 3.9921 3.0181 2.7984 2.4239 2.4602 2.4310 2.3786 2.3688 2.4014 2.
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
                mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-04 1.0000e-04
          gradient: [65.8915 2.7917 12.6176 8.3355 8.0645 8.0544 1.4140 2.2959 2.5186 1.2443 1.8254 2.0888 0
          val_fail: [0 0 1 0 0 0 0 1 2 0 0 1 2 3 4 0 1 2 3 4 5 6]
```

```
       best_perf: 2.3221
      best_vperf: 2.3635
      best_tperf: NaN
Rmse_train = 1×6
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563
Rmse_val = 1×6
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 92
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:
```

```
                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 3 4 5 6 8 10 11 14 16 18 19 22 23 24 25 27 28 30 31 32 33 34 36 37 39 40 42 43 44 4
            valInd: [7 9 12 13 15 17 20 21 26 29 35 38 41 47 49 52 55 56 63 68 74 75 80 83 84 90 91 97 98 10
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 62
         trainMask: {[1 1 1 1 1 1 NaN 1 NaN 1 1 NaN NaN 1 NaN 1 NaN 1 1 NaN NaN 1 1 1 1 NaN 1 1 NaN 1 1 1 1
           valMask: {[NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN NaN 1 1 NaN 1 NaN 1 NaN NaN 1 1 NaN NaN NaN NaN 1
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 56
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 3
              time: [0.0922 0.1362 0.1526 0.1675 0.1845 0.1973 0.2163 0.2277 0.2413 0.2532 0.2658 0.2767 0.2
              perf: [8.8573 3.8855 2.7899 2.6696 2.5542 2.4500 2.4157 2.4136 2.3073 2.2855 2.2651 2.2370 2.2
             vperf: [9.2026 4.0555 3.0343 2.8547 2.7907 2.6983 2.6685 2.6340 2.5432 2.5482 2.5296 2.5203 2.4
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
                mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 0.0100 1.0000e-03 1.0000e-03 1.0000e-04 1.0
          gradient: [16.5898 5.9017 2.1368 2.6448 1.9837 0.3329 0.8323 2.0461 0.2441 0.8434 1.0066 0.6759 1.
          val_fail: [0 0 0 0 0 0 0 0 1 0 0 0 1 2 3 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1
         best_perf: 2.1010
        best_vperf: 2.3732
        best_tperf: NaN
Rmse_train = 1×7
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847
Rmse_val = 1×7
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

           numInputs: 1
           numLayers: 2
          numOutputs: 1
```

```
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 105
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
```

```
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 3 4 6 7 8 9 10 13 14 15 16 17 18 19 20 21 25 26 28 29 30 34 36 38 41 43 44 45 46 47
          valInd: [5 11 12 22 23 24 27 31 32 33 35 37 39 40 42 48 51 59 61 62 70 71 74 86 92 93 94 96 97 9
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 35
       trainMask: {[1 1 1 1 NaN 1 1 1 1 1 NaN NaN 1 1 1 1 1 1 1 1 NaN NaN NaN 1 1 NaN 1 1 1 NaN NaN NaN
         valMask: {[NaN NaN NaN NaN 1 NaN NaN NaN NaN NaN 1 1 NaN NaN NaN NaN NaN NaN NaN NaN NaN 1 1 1 Na
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 29
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 3
            time: [0.1032 0.1387 0.1554 0.1675 0.1884 0.2044 0.2368 0.2482 0.2631 0.2764 0.2870 0.3023 0.3
            perf: [33.9750 5.8658 3.7951 2.8812 2.4722 2.4116 2.3813 2.3779 2.3315 2.3109 2.3098 2.2900 2.
           vperf: [33.0407 5.7580 3.9424 2.9760 2.6108 2.5785 2.5481 2.5658 2.5257 2.5221 2.5293 2.5229 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
              mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-04 0.0100 1.0000e-03 1.0000e-03 1.0000e-04 1.0
        gradient: [70.4549 8.3739 12.7903 6.2497 1.7555 1.4300 1.1278 1.0164 1.1220 0.7166 1.1473 0.6477 0
        val_fail: [0 0 0 0 0 0 0 1 0 0 1 2 0 1 0 0 0 0 0 1 0 0 1 2 3 4 5 0 1 0 1 2 3 4 5 6]
       best_perf: 2.1919
      best_vperf: 2.4546
      best_tperf: NaN
Rmse_train = 1×8
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845
Rmse_val = 1×8
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 118
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}
```

```
          inputs: {1x1 cell array of 1 input}
          layers: {2x1 cell array of 2 layers}
         outputs: {1x2 cell array of 1 output}
          biases: {2x1 cell array of 2 biases}
    inputWeights: {2x1 cell array of 1 weight}
    layerWeights: {2x2 cell array of 1 weight}

  functions:

        adaptFcn: 'adaptwb'
      adaptParam: (none)
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
     divideParam: .trainRatio, .valRatio, .testRatio
      divideMode: 'sample'
         initFcn: 'initlay'
      performFcn: 'mse'
    performParam: .regularization, .normalization
        plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                   'plotregression', 'plotfit'}
      plotParams: {1x5 cell array of 5 params}
        trainFcn: 'trainlm'
      trainParam: .showWindow, .showCommandLine, .show, .epochs,
                  .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                  .mu_inc, .mu_max

  weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

  methods:

           adapt: Learn while in continuous use
       configure: Configure inputs & outputs
          gensim: Generate Simulink model
            init: Initialize weights & biases
         perform: Calculate performance
             sim: Evaluate network outputs given inputs
           train: Train network with examples
            view: View diagram
     unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 5 6 8 9 10 12 13 14 15 16 17 19 21 22 23 25 26 28 32 34 35 36 37 39 41 42 45 48 49
          valInd: [3 4 7 11 18 20 24 27 29 30 31 33 38 40 43 44 46 47 52 57 59 60 62 64 68 71 74 75 76 83
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 18
       trainMask: {[1 1 NaN NaN 1 1 NaN 1 1 1 NaN 1 1 1 1 1 NaN 1 NaN 1 1 1 NaN 1 1 NaN 1 NaN NaN NaN 1
         valMask: {[NaN NaN 1 1 NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN 1 NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 12
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
```

```
          epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18]
           time: [0.1443 0.1791 0.1927 0.2065 0.2319 0.2472 0.2685 0.2842 0.3022 0.3199 0.3334 0.3448 0.3
           perf: [51.4768 8.8291 6.3248 3.3542 2.9488 2.8411 2.5761 2.4752 2.4302 2.3914 2.3696 2.3695 2.
          vperf: [56.1666 8.0684 6.2697 3.1558 2.6815 2.8296 2.4643 2.3737 2.3707 2.3380 2.3516 2.3582 2.
          tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
             mu: [1.0000e-03 1.0000e-04 1.0000e-05 1.0000e-06 1.0000e-03 1.0000e-04 1.0000e-05 1.0000e-05
       gradient: [129.4267 21.5345 19.2290 6.1538 6.3615 3.2826 3.9224 1.0579 1.5141 1.4077 0.8113 1.4678
       val_fail: [0 0 0 0 0 1 0 0 0 0 1 2 0 1 2 3 4 5 6]
      best_perf: 2.3217
     best_vperf: 2.3204
     best_tperf: NaN
Rmse_train = 1×9
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×9
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

             name: 'Function Fitting Neural Network'
         userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 131
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

           input: Equivalent to inputs{1}
          output: Equivalent to outputs{2}

          inputs: {1x1 cell array of 1 input}
          layers: {2x1 cell array of 2 layers}
         outputs: {1x2 cell array of 1 output}
          biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

        adaptFcn: 'adaptwb'
      adaptParam: (none)
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
     divideParam: .trainRatio, .valRatio, .testRatio
      divideMode: 'sample'
         initFcn: 'initlay'
      performFcn: 'mse'
    performParam: .regularization, .normalization
        plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
```

```
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
           configure: Configure inputs & outputs
              gensim: Generate Simulink model
                init: Initialize weights & biases
             perform: Calculate performance
                 sim: Evaluate network outputs given inputs
               train: Train network with examples
                view: View diagram
         unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 4 6 8 10 13 15 16 17 18 20 21 24 25 26 27 29 30 32 33 34 37 38 41 42 43 44 45 46 47
            valInd: [3 5 7 9 11 12 14 19 22 23 28 31 35 36 39 40 52 56 57 63 68 73 74 76 85 86 89 92 95 98 1
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 22
         trainMask: {[1 1 NaN 1 NaN 1 NaN 1 NaN 1 NaN NaN 1 NaN 1 1 1 1 NaN 1 1 NaN NaN 1 1 1 1 NaN 1 1 NaN
           valMask: {[NaN NaN 1 NaN 1 NaN 1 NaN 1 NaN 1 1 NaN 1 NaN NaN NaN NaN 1 NaN NaN 1 1 NaN NaN NaN Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 16
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22]
              time: [0.0930 0.1371 0.1650 0.1845 0.2042 0.2313 0.2485 0.2663 0.2823 0.3297 0.3593 0.3820 0.4
              perf: [45.4396 5.2545 4.7859 3.0804 2.6530 2.5216 2.4239 2.3674 2.3658 2.2731 2.2344 2.2269 2.
             vperf: [46.6219 5.4233 4.9422 3.1396 2.6982 2.5596 2.4214 2.3844 2.4464 2.3234 2.3196 2.3073 2.
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
                mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
          gradient: [130.6375 12.1296 20.4234 4.9387 3.9135 2.8538 1.5558 0.7570 2.1076 0.6009 0.5702 0.9485
          val_fail: [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 2 3 4 5 6]
         best_perf: 2.1420
        best_vperf: 2.2450
        best_tperf: NaN
Rmse_train = 1×10
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×10
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network
```

```
          name: 'Function Fitting Neural Network'
      userdata: (your custom info)

   dimensions:

       numInputs: 1
       numLayers: 2
      numOutputs: 1
   numInputDelays: 0
   numLayerDelays: 0
numFeedbackDelays: 0
numWeightElements: 144
      sampleTime: 1

   connections:

     biasConnect: [1; 1]
    inputConnect: [1; 0]
    layerConnect: [0 0; 1 0]
   outputConnect: [0 1]

   subobjects:

           input: Equivalent to inputs{1}
          output: Equivalent to outputs{2}

          inputs: {1x1 cell array of 1 input}
          layers: {2x1 cell array of 2 layers}
         outputs: {1x2 cell array of 1 output}
          biases: {2x1 cell array of 2 biases}
    inputWeights: {2x1 cell array of 1 weight}
    layerWeights: {2x2 cell array of 1 weight}

   functions:

      adaptFcn: 'adaptwb'
    adaptParam: (none)
      derivFcn: 'defaultderiv'
     divideFcn: 'dividerand'
   divideParam: .trainRatio, .valRatio, .testRatio
    divideMode: 'sample'
       initFcn: 'initlay'
    performFcn: 'mse'
  performParam: .regularization, .normalization
      plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                'plotregression', 'plotfit'}
    plotParams: {1x5 cell array of 5 params}
      trainFcn: 'trainlm'
    trainParam: .showWindow, .showCommandLine, .show, .epochs,
                .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                .mu_inc, .mu_max

   weight and bias values:

            IW: {2x1 cell} containing 1 input weight matrix
            LW: {2x2 cell} containing 1 layer weight matrix
             b: {2x1 cell} containing 2 bias vectors

   methods:

         adapt: Learn while in continuous use
     configure: Configure inputs & outputs
        gensim: Generate Simulink model
          init: Initialize weights & biases
```

```
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 4 5 6 7 9 11 14 15 16 18 19 20 21 22 24 25 26 27 28 29 30 32 34 35 36 37 38 41 43 4
          valInd: [3 8 10 12 13 17 23 31 33 39 40 42 44 46 47 48 51 52 54 55 57 58 60 62 65 66 67 69 75 76
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 22
       trainMask: {[1 1 NaN 1 1 1 1 NaN 1 NaN 1 NaN NaN 1 1 1 NaN 1 1 1 1 1 NaN 1 1 1 1 1 1 NaN 1 NaN 1
         valMask: {[NaN NaN 1 NaN NaN NaN NaN 1 NaN 1 NaN 1 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN 1 NaN NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 16
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22]
            time: [0.1083 0.1390 0.1591 0.1768 0.1987 0.2219 0.2448 0.2612 0.2798 0.2980 0.3134 0.3368 0.3
            perf: [169.7096 7.2125 3.6295 3.1867 2.6888 2.3713 2.2966 2.2622 2.2334 2.2074 2.2019 2.1747 2
           vperf: [168.7358 7.8419 3.8069 3.5870 2.9431 2.6750 2.5761 2.5267 2.5007 2.4817 2.5004 2.4922 2
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
              mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [306.9831 26.2237 17.3390 12.2650 3.6072 2.4172 0.8158 0.8721 0.8896 0.5631 2.8056 3.003
        val_fail: [0 0 0 0 0 0 0 0 0 0 1 2 3 4 0 1 0 1 2 3 4 5 6]
       best_perf: 2.0829
      best_vperf: 2.4700
      best_tperf: NaN
Rmse_train = 1×11
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×11
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 157
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
```

```
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 3 4 5 7 8 9 11 12 16 17 18 19 21 22 23 24 25 26 28 32 33 36 37 38 39 41 43 44 45 46 4
            valInd: [2 6 10 13 14 15 20 27 29 30 31 34 35 40 42 59 62 64 76 83 84 85 87 92 93 98 104 106 109
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
```

```
      num_epochs: 16
       trainMask: {[1 NaN 1 1 1 NaN 1 1 1 NaN 1 1 NaN NaN NaN 1 1 1 1 NaN 1 1 1 1 1 1 NaN 1 NaN NaN NaN 1
         valMask: {[NaN 1 NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN 1 1 1 NaN NaN NaN NaN 1 NaN NaN NaN NaN NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 10
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
            time: [0.0438 0.0842 0.1102 0.1413 0.1673 0.2029 0.2206 0.2452 0.2621 0.2844 0.3180 0.3368 0.3
            perf: [18.1219 4.7981 4.2573 2.6308 2.4804 2.4380 2.4115 2.3670 2.3343 2.2889 2.2063 2.1724 2.
           vperf: [18.1149 4.6748 4.4610 2.7594 2.5709 2.5529 2.5462 2.5392 2.5709 2.5408 2.5015 2.5271 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-04 1.0000e-03
        gradient: [62.7201 10.3317 8.5144 4.7201 1.2214 1.3797 0.6870 2.0195 1.2151 1.0843 1.5425 1.1268 0
        val_fail: [0 0 0 0 0 0 0 0 1 2 0 1 2 3 4 5 6]
       best_perf: 2.2063
      best_vperf: 2.5015
      best_tperf: NaN
Rmse_train = 1×12
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×12
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 170
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
```

```
          divideFcn: 'dividerand'
        divideParam: .trainRatio, .valRatio, .testRatio
         divideMode: 'sample'
            initFcn: 'initlay'
         performFcn: 'mse'
       performParam: .regularization, .normalization
           plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                      'plotregression', 'plotfit'}
         plotParams: {1x5 cell array of 5 params}
           trainFcn: 'trainlm'
         trainParam: .showWindow, .showCommandLine, .show, .epochs,
                     .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                     .mu_inc, .mu_max

    weight and bias values:

                  IW: {2x1 cell} containing 1 input weight matrix
                  LW: {2x2 cell} containing 1 layer weight matrix
                   b: {2x1 cell} containing 2 bias vectors

    methods:

               adapt: Learn while in continuous use
           configure: Configure inputs & outputs
              gensim: Generate Simulink model
                init: Initialize weights & biases
             perform: Calculate performance
                 sim: Evaluate network outputs given inputs
               train: Train network with examples
                view: View diagram
         unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
           trainFcn: 'trainlm'
         trainParam: [1×1 struct]
         performFcn: 'mse'
       performParam: [1×1 struct]
           derivFcn: 'defaultderiv'
          divideFcn: 'dividerand'
         divideMode: 'sample'
        divideParam: [1×1 struct]
           trainInd: [1 2 4 7 8 11 13 14 15 16 17 18 19 20 21 23 24 25 26 27 28 30 31 34 35 36 37 39 40 41 42
             valInd: [3 5 6 9 10 12 22 29 32 33 38 43 48 51 53 55 58 68 69 71 73 76 80 81 85 88 91 98 100 101
            testInd: [1×0 double]
               stop: 'Training finished: Met validation criterion'
         num_epochs: 19
          trainMask: {[1 1 NaN 1 NaN NaN 1 1 NaN NaN 1 NaN 1 1 1 1 1 1 1 1 1 NaN 1 1 1 1 1 1 NaN 1 1 NaN NaN
            valMask: {[NaN NaN 1 NaN 1 1 NaN NaN 1 1 NaN 1 NaN NaN NaN NaN NaN NaN NaN NaN NaN 1 NaN NaN NaN
           testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
         best_epoch: 13
               goal: 0
             states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
              epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]
               time: [0.1096 0.1421 0.1662 0.1869 0.2080 0.2316 0.2648 0.2871 0.3091 0.3304 0.3559 0.3874 0.4
               perf: [51.9821 8.0905 3.1506 2.7279 2.3624 2.3058 2.2144 2.1700 2.1448 2.1303 2.0976 2.0835 2.
              vperf: [52.3990 8.3710 3.1962 2.9208 2.5269 2.5331 2.4971 2.4647 2.4893 2.4633 2.4815 2.4631 2.
              tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
                 mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
           gradient: [148.5882 28.8385 6.2757 8.1758 2.3122 1.9058 1.3184 0.5923 0.6890 2.4783 0.5141 0.7371
           val_fail: [0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 2 3 4 5 6]
          best_perf: 2.0573
         best_vperf: 2.4433
         best_tperf: NaN
Rmse_train = 1×13
```

```
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×13
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 183
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors
```

```
    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
         trainFcn: 'trainlm'
       trainParam: [1×1 struct]
       performFcn: 'mse'
     performParam: [1×1 struct]
         derivFcn: 'defaultderiv'
        divideFcn: 'dividerand'
       divideMode: 'sample'
      divideParam: [1×1 struct]
         trainInd: [1 3 4 5 6 7 8 9 10 15 16 17 18 20 23 24 25 27 30 31 32 33 35 36 37 39 41 42 43 44 45 47
           valInd: [2 11 12 13 14 19 21 22 26 28 29 34 38 40 46 49 53 55 57 69 73 75 76 79 83 87 90 104 107
          testInd: [1×0 double]
             stop: 'Training finished: Met validation criterion'
       num_epochs: 17
        trainMask: {[1 NaN 1 1 1 1 1 1 1 1 NaN NaN NaN NaN 1 1 1 1 NaN 1 NaN NaN 1 1 1 NaN 1 NaN NaN 1 1 1
          valMask: {[NaN 1 NaN NaN NaN NaN NaN NaN NaN NaN NaN 1 1 1 1 NaN NaN NaN NaN 1 NaN 1 1 NaN NaN NaN 1
         testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
       best_epoch: 11
             goal: 0
           states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
            epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17]
             time: [0.0420 0.0919 0.1163 0.1472 0.1814 0.2091 0.2318 0.2582 0.2931 0.3207 0.3451 0.3674 0.4
             perf: [99.7184 26.3480 10.7652 5.8995 3.9573 3.3423 2.5086 2.3783 2.3055 2.2803 2.2608 2.2421
            vperf: [96.2600 25.3142 11.3258 5.9917 3.9020 3.4748 2.5550 2.4801 2.4004 2.3846 2.3770 2.3626
            tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
               mu: [1.0000e-03 1.0000e-04 1.0000e-05 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03
         gradient: [214.1942 65.0213 41.0312 27.9657 15.2620 20.8782 4.4486 6.1436 0.6607 1.1895 1.5429 0.7
         val_fail: [0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
        best_perf: 2.2421
       best_vperf: 2.3626
       best_tperf: NaN
Rmse_train = 1×14
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×14
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

          numInputs: 1
          numLayers: 2
         numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 196
```

```
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
```

```
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [3 4 5 6 7 8 9 13 15 16 17 18 21 22 23 24 25 26 28 29 30 31 32 36 37 38 40 41 42 43 44 4
          valInd: [1 2 10 11 12 14 19 20 27 33 34 35 39 46 47 50 51 52 53 61 63 68 70 72 74 85 86 88 91 92
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 15
       trainMask: {[NaN NaN 1 1 1 1 1 1 1 NaN NaN NaN 1 NaN 1 1 1 1 NaN NaN 1 1 1 1 1 NaN 1 1 1 1 1 NaN
         valMask: {[1 1 NaN NaN NaN NaN NaN NaN NaN 1 1 1 NaN 1 NaN NaN NaN NaN 1 1 NaN NaN NaN NaN NaN Na
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 9
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
            time: [0.0995 0.1495 0.1747 0.2030 0.2391 0.2598 0.2791 0.3060 0.3353 0.3591 0.3844 0.4060 0.4
            perf: [83.3879 5.8134 5.2548 2.9772 2.4562 2.4441 2.4326 2.3082 2.2219 2.1878 2.1622 2.1516 2.
           vperf: [81.6431 5.4682 5.2927 2.8691 2.4015 2.4352 2.4387 2.3171 2.2645 2.2609 2.2686 2.2812 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 0.0100 1.0000e-03 1.0000e-04 1.0000e-03 1.0
        gradient: [243.6047 23.2845 8.7021 11.8721 1.1763 2.9702 1.9838 2.5231 1.2224 1.3945 0.3298 0.7852
        val_fail: [0 0 0 0 0 1 2 0 0 0 1 2 3 4 5 6]
       best_perf: 2.1878
      best_vperf: 2.2609
      best_tperf: NaN
Rmse_train = 1×15
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×15
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 209
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
```

```
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 4 5 6 8 9 10 11 13 15 16 18 19 20 21 22 23 24 25 26 27 28 30 32 35 38 39 41 43 44 4
            valInd: [3 7 12 14 17 29 31 33 34 36 37 40 42 51 54 55 66 75 78 79 84 95 98 112 114 115 120 127
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 17
         trainMask: {[1 1 NaN 1 1 1 NaN 1 1 1 1 NaN 1 NaN 1 1 NaN 1 1 1 1 1 1 1 1 1 1 NaN 1 NaN 1 NaN NaN
           valMask: {[NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN 1 NaN 1 NaN NaN 1 NaN NaN NaN NaN NaN NaN NaN
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 11
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17]
              time: [0.1095 0.1649 0.2079 0.2272 0.2589 0.2814 0.3033 0.3243 0.3453 0.3805 0.4050 0.4280 0.4
              perf: [74.5141 5.3627 3.9781 3.2369 3.2108 2.5418 2.4551 2.3960 2.3343 2.2954 2.2491 2.2276 2.
             vperf: [77.1036 5.3670 3.9314 3.1147 3.1053 2.4455 2.3713 2.3561 2.3024 2.3084 2.2687 2.2512 2.
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
```

```
             mu: [1.0000e-03 1.0000e-04 0.0100 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0
       gradient: [207.2962 6.8249 20.0990 13.7100 16.3390 2.1903 2.4651 1.8905 1.7003 2.2813 0.5610 0.240
       val_fail: [0 0 0 0 0 0 0 0 0 1 0 0 1 2 3 4 5 6]
      best_perf: 2.2276
     best_vperf: 2.2512
     best_tperf: NaN
Rmse_train = 1×16
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×16
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

             name: 'Function Fitting Neural Network'
         userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 222
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

    subobjects:

          input: Equivalent to inputs{1}
         output: Equivalent to outputs{2}

         inputs: {1x1 cell array of 1 input}
         layers: {2x1 cell array of 2 layers}
        outputs: {1x2 cell array of 1 output}
         biases: {2x1 cell array of 2 biases}
   inputWeights: {2x1 cell array of 1 weight}
   layerWeights: {2x2 cell array of 1 weight}

    functions:

       adaptFcn: 'adaptwb'
     adaptParam: (none)
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
    divideParam: .trainRatio, .valRatio, .testRatio
     divideMode: 'sample'
        initFcn: 'initlay'
     performFcn: 'mse'
   performParam: .regularization, .normalization
       plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                 'plotregression', 'plotfit'}
     plotParams: {1x5 cell array of 5 params}
       trainFcn: 'trainlm'
     trainParam: .showWindow, .showCommandLine, .show, .epochs,
                 .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
```

```
                .mu_inc, .mu_max

    weight and bias values:

               IW: {2x1 cell} containing 1 input weight matrix
               LW: {2x2 cell} containing 1 layer weight matrix
                b: {2x1 cell} containing 2 bias vectors

    methods:

            adapt: Learn while in continuous use
        configure: Configure inputs & outputs
           gensim: Generate Simulink model
             init: Initialize weights & biases
          perform: Calculate performance
              sim: Evaluate network outputs given inputs
            train: Train network with examples
             view: View diagram
      unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [2 4 5 6 7 8 9 10 13 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 31 32 33 34 37 38 39 4
          valInd: [1 3 11 12 14 30 35 36 43 60 61 64 65 66 73 76 78 85 86 87 92 93 101 107 108 113 117 119
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 31
       trainMask: {[NaN 1 NaN 1 1 1 1 1 1 NaN NaN 1 NaN 1 1 1 1 1 1 1 1 1 1 1 1 1 NaN 1 1 1 1 NaN Na
         valMask: {[1 NaN 1 NaN NaN NaN NaN NaN NaN NaN 1 1 NaN 1 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 25
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31]
            time: [0.0997 0.1454 0.1706 0.1947 0.2253 0.2647 0.2898 0.3181 0.3458 0.3731 0.4026 0.4248 0.4
            perf: [218.9114 7.8310 5.4146 4.3422 3.2648 2.4202 2.4116 2.3302 2.2012 2.1351 2.0918 2.0598 2
           vperf: [215.5020 7.9746 5.5885 4.4523 3.3953 2.5877 2.5972 2.5847 2.4734 2.4104 2.4252 2.4004 2
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-04 1.0000e-04
        gradient: [422.5232 45.5959 17.7047 13.5058 14.2873 5.1351 6.5116 5.9241 2.4408 1.9065 1.8050 0.53
        val_fail: [0 0 0 0 0 0 1 0 0 0 1 0 1 2 3 4 5 0 1 0 1 0 1 0 1 0 1 2 3 4 5 6]
       best_perf: 1.9674
      best_vperf: 2.3508
      best_tperf: NaN
Rmse_train = 1×17
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×17
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:
```

```
        numInputs: 1
        numLayers: 2
       numOutputs: 1
   numInputDelays: 0
   numLayerDelays: 0
numFeedbackDelays: 0
numWeightElements: 235
       sampleTime: 1

   connections:

      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

   subobjects:

            input: Equivalent to inputs{1}
           output: Equivalent to outputs{2}

           inputs: {1x1 cell array of 1 input}
           layers: {2x1 cell array of 2 layers}
          outputs: {1x2 cell array of 1 output}
           biases: {2x1 cell array of 2 biases}
     inputWeights: {2x1 cell array of 1 weight}
     layerWeights: {2x2 cell array of 1 weight}

   functions:

         adaptFcn: 'adaptwb'
       adaptParam: (none)
         derivFcn: 'defaultderiv'
        divideFcn: 'dividerand'
      divideParam: .trainRatio, .valRatio, .testRatio
       divideMode: 'sample'
          initFcn: 'initlay'
       performFcn: 'mse'
     performParam: .regularization, .normalization
         plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
       plotParams: {1x5 cell array of 5 params}
         trainFcn: 'trainlm'
       trainParam: .showWindow, .showCommandLine, .show, .epochs,
                   .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                   .mu_inc, .mu_max

   weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

   methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
```

```
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 3 4 5 6 7 8 11 12 13 14 15 18 21 22 24 25 26 27 29 31 33 34 35 36 37 38 39 40 41 42 4
          valInd: [2 9 10 16 17 19 20 23 28 30 32 46 49 50 52 56 59 67 71 75 83 87 89 94 95 100 103 105 10
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 14
       trainMask: {[1 NaN 1 1 1 1 1 1 NaN NaN 1 1 1 1 1 NaN NaN 1 NaN NaN 1 1 NaN 1 1 1 1 NaN 1 NaN 1 NaN
         valMask: {[NaN 1 NaN NaN NaN NaN NaN NaN 1 1 NaN NaN NaN NaN NaN 1 1 NaN 1 1 NaN NaN 1 NaN NaN Na
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 8
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
            time: [0.1038 0.1519 0.1782 0.2037 0.2293 0.2687 0.2968 0.3246 0.3521 0.3886 0.4172 0.4454 0.4
            perf: [25.6640 9.6680 8.3441 4.4685 2.8523 2.4852 2.4397 2.2631 2.2171 2.1931 2.1660 2.1405 2.
           vperf: [24.8059 9.4174 8.1080 4.4102 2.7952 2.4414 2.5030 2.3279 2.3261 2.3297 2.3342 2.3289 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [69.4406 32.5646 38.5520 12.7817 7.2977 4.0686 4.2248 2.8590 0.4593 0.5939 0.5440 0.6258
        val_fail: [0 0 0 0 0 0 1 0 0 1 2 3 4 5 6]
       best_perf: 2.2171
      best_vperf: 2.3261
      best_tperf: NaN
Rmse_train = 1×18
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×18
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 248
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:
```

```
          input: Equivalent to inputs{1}
         output: Equivalent to outputs{2}

         inputs: {1x1 cell array of 1 input}
         layers: {2x1 cell array of 2 layers}
        outputs: {1x2 cell array of 1 output}
         biases: {2x1 cell array of 2 biases}
   inputWeights: {2x1 cell array of 1 weight}
   layerWeights: {2x2 cell array of 1 weight}

  functions:

       adaptFcn: 'adaptwb'
     adaptParam: (none)
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
    divideParam: .trainRatio, .valRatio, .testRatio
     divideMode: 'sample'
        initFcn: 'initlay'
     performFcn: 'mse'
   performParam: .regularization, .normalization
       plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                  'plotregression', 'plotfit'}
     plotParams: {1x5 cell array of 5 params}
       trainFcn: 'trainlm'
     trainParam: .showWindow, .showCommandLine, .show, .epochs,
                 .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                 .mu_inc, .mu_max

  weight and bias values:

              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

  methods:

           adapt: Learn while in continuous use
       configure: Configure inputs & outputs
          gensim: Generate Simulink model
            init: Initialize weights & biases
         perform: Calculate performance
             sim: Evaluate network outputs given inputs
           train: Train network with examples
            view: View diagram
     unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
       trainFcn: 'trainlm'
     trainParam: [1×1 struct]
     performFcn: 'mse'
   performParam: [1×1 struct]
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
     divideMode: 'sample'
    divideParam: [1×1 struct]
       trainInd: [4 5 6 7 9 10 11 12 15 16 17 19 21 22 24 25 28 31 32 33 34 35 36 37 39 40 41 42 43 45 46
         valInd: [1 2 3 8 13 14 18 20 23 26 27 29 30 38 44 50 51 53 57 60 64 66 69 70 71 86 88 91 93 94 9
        testInd: [1×0 double]
           stop: 'Training finished: Met validation criterion'
     num_epochs: 24
      trainMask: {[NaN NaN NaN 1 1 1 1 NaN 1 1 1 1 NaN NaN 1 1 1 NaN 1 NaN 1 1 NaN 1 1 NaN NaN 1 NaN NaN
        valMask: {[1 1 1 NaN NaN NaN NaN 1 NaN NaN NaN NaN 1 1 NaN NaN NaN 1 NaN 1 NaN NaN 1 NaN NaN 1 1
       testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
```

```
      best_epoch: 18
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
            time: [0.1065 0.1616 0.1923 0.2276 0.2648 0.2875 0.3187 0.3436 0.3804 0.4039 0.4311 0.4576 0.4
            perf: [47.3181 11.3978 4.6493 2.7541 2.5488 2.4629 2.4486 2.2055 2.1303 2.0895 2.0699 2.0574 2
           vperf: [46.4740 12.0896 5.0137 2.9918 2.7835 2.7628 2.7508 2.5134 2.4838 2.4604 2.4448 2.4408 2
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03
        gradient: [89.2640 37.2900 20.0953 8.2941 10.1102 5.3392 3.6998 3.1715 1.0448 1.0167 0.8317 0.2235
        val_fail: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 0 1 2 3 4 5 6]
       best_perf: 1.9809
      best_vperf: 2.4324
      best_tperf: NaN
Rmse_train = 1×19
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×19
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 261
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
```

```
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 3 4 6 7 8 10 13 15 16 19 21 22 24 25 26 27 28 29 30 31 33 34 37 38 40 41 43 44 45 4
            valInd: [5 9 11 12 14 17 18 20 23 32 35 36 39 42 46 50 60 63 67 74 76 78 80 83 88 92 105 106 107
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 17
         trainMask: {[1 1 1 1 NaN 1 1 1 NaN 1 NaN NaN 1 NaN 1 1 NaN NaN 1 NaN 1 1 NaN 1 1 1 1 1 1 1 NaN 1
           valMask: {[NaN NaN NaN NaN 1 NaN NaN NaN 1 NaN 1 1 NaN 1 NaN NaN 1 1 NaN 1 NaN NaN 1 NaN NaN NaN
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 11
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17]
              time: [0.1089 0.1596 0.1907 0.2244 0.2563 0.2881 0.3144 0.3404 0.3748 0.4078 0.4379 0.4661 0.4
              perf: [45.3604 6.6464 5.0390 3.6926 3.1279 2.7001 2.4763 2.3895 2.3079 2.2494 2.2167 2.1804 2.
             vperf: [43.7743 6.3105 4.5509 3.4650 2.9235 2.6531 2.3689 2.3532 2.2670 2.2457 2.2430 2.2258 2.
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
                mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03
          gradient: [145.5501 24.8041 16.9499 18.6260 5.1359 8.2092 3.6635 4.3198 1.6565 1.4430 1.9809 0.533
          val_fail: [0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
         best_perf: 2.1804
        best_vperf: 2.2258
        best_tperf: NaN
Rmse_train = 1×20
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×20
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
```

```
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 274
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:
```

```
             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [2 3 4 6 7 8 9 12 13 14 15 16 20 23 24 25 28 29 30 31 32 33 34 35 36 37 38 41 42 47 48 4
          valInd: [1 5 10 11 17 18 19 21 22 26 27 39 40 43 44 45 46 50 51 52 54 63 66 67 80 81 82 83 86 91
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 16
       trainMask: {[NaN 1 1 1 NaN 1 1 1 1 NaN NaN 1 1 1 1 1 NaN NaN NaN 1 NaN NaN 1 1 1 NaN NaN 1 1 1 1 1
         valMask: {[1 NaN NaN NaN 1 NaN NaN NaN NaN 1 1 NaN NaN NaN NaN NaN 1 1 1 NaN 1 1 NaN NaN NaN 1 1
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 10
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
            time: [0.0496 0.1068 0.1404 0.1641 0.2026 0.2314 0.2661 0.3049 0.3386 0.3711 0.4135 0.4460 0.4
            perf: [30.4852 8.3645 7.4263 5.0579 4.1135 2.6941 2.5382 2.1462 2.1166 2.0824 2.0640 2.0456 2.
           vperf: [30.8850 8.1642 7.7397 5.2415 4.2875 2.9303 2.8284 2.5004 2.4727 2.4570 2.4520 2.4670 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03
        gradient: [135.9206 33.9035 55.5958 12.3445 13.4749 6.8833 16.7080 2.9149 3.8411 1.6467 1.1845 1.0
        val_fail: [0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
       best_perf: 2.0640
      best_vperf: 2.4520
      best_tperf: NaN
Rmse_train = 1×21
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×21
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 287
        sampleTime: 1

    connections:
```

```
       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
```

```
        trainInd: [1 2 5 6 7 9 10 11 12 13 14 16 17 19 20 21 22 24 27 28 30 31 32 33 35 37 38 39 41 42 43
          valInd: [3 4 8 15 18 23 25 26 29 34 36 40 44 45 47 49 64 66 67 71 72 78 80 89 96 100 101 109 112
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 16
       trainMask: {[1 1 NaN NaN 1 1 1 NaN 1 1 1 1 1 1 NaN 1 1 NaN 1 1 1 1 NaN 1 NaN NaN 1 1 NaN 1 1 1 1 Na
         valMask: {[NaN NaN 1 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 NaN NaN 1 NaN NaN NaN NaN 1 NaN 1
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 10
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16]
            time: [0.0530 0.1052 0.1342 0.1711 0.2133 0.2528 0.2807 0.3172 0.3453 0.3762 0.4061 0.4404 0.4
            perf: [95.1396 10.7865 3.6530 2.5936 2.4621 2.2585 2.2356 2.1668 2.1220 2.0918 2.0680 2.0634 2
           vperf: [93.0647 10.3530 3.9090 2.8165 2.6970 2.5181 2.5240 2.4838 2.4597 2.4557 2.4476 2.4884 2
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 0.0100 1.0000e-03 1.0000e-03 1.0
        gradient: [238.4556 38.5721 11.3706 4.7753 9.1278 1.7996 3.5804 2.2111 0.3683 0.7491 0.6551 2.6451
        val_fail: [0 0 0 0 0 0 1 0 0 0 0 1 2 3 4 5 6]
       best_perf: 2.0680
      best_vperf: 2.4476
      best_tperf: NaN
Rmse_train = 1×22
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×22
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 300
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:
```

```
          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 5 6 7 8 10 11 12 15 16 17 18 19 21 22 23 24 26 27 29 31 32 33 35 38 39 42 43 44 45
            valInd: [3 4 9 13 14 20 25 28 30 34 36 37 40 41 49 51 58 59 62 68 70 71 72 74 75 78 81 85 93 96
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 22
         trainMask: {[1 1 NaN NaN 1 1 1 1 NaN 1 1 1 NaN NaN 1 1 1 1 1 NaN 1 1 1 1 NaN 1 1 NaN 1 NaN 1 1 1 Na
           valMask: {[NaN NaN 1 1 NaN NaN NaN NaN 1 NaN NaN NaN 1 1 NaN NaN NaN NaN NaN 1 NaN NaN NaN NaN 1
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 16
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22]
              time: [0.1137 0.1685 0.1979 0.2301 0.2653 0.2915 0.3157 0.3502 0.3820 0.4134 0.4598 0.4994 0.5
              perf: [350.1600 17.9245 13.9529 13.2555 6.5633 3.9643 2.9870 2.7583 2.6167 2.6132 2.3886 2.345
             vperf: [338.2161 16.5395 14.8633 13.5128 6.3632 4.3392 3.3289 2.9864 2.9708 3.0313 2.7368 2.724
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
                mu: [1.0000e-03 1.0000e-04 1.0000e-05 1.0000e-05 1.0000e-06 1.0000e-07 1.0000e-08 1.0000e-07
          gradient: [595.6949 64.2541 25.0539 58.3507 20.7567 13.6624 9.1638 8.1866 5.1507 5.8772 3.4730 2.5
          val_fail: [0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 2 3 4 5 6]
```

```
       best_perf: 2.1966
      best_vperf: 2.5981
      best_tperf: NaN
Rmse_train = 1×23
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×23
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 313
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:
```

```
                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 3 4 5 6 8 9 10 11 12 16 17 22 23 25 26 27 28 29 30 31 32 35 36 38 39 41 42 43 44 45 4
            valInd: [2 7 13 14 15 18 19 20 21 24 33 34 37 40 48 49 50 51 56 59 60 67 72 73 74 75 76 80 82 84
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 27
         trainMask: {[1 NaN 1 1 1 1 NaN 1 1 1 1 1 NaN NaN NaN 1 1 NaN NaN NaN NaN 1 1 NaN 1 1 1 1 1 1 1 Na
           valMask: {[NaN 1 NaN NaN NaN NaN 1 NaN NaN NaN NaN NaN 1 1 1 NaN NaN 1 1 1 1 NaN NaN 1 NaN NaN Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 21
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27]
              time: [0.1118 0.1662 0.2028 0.2412 0.2795 0.3191 0.3508 0.3958 0.4351 0.4713 0.5138 0.5503 0.5
              perf: [46.2677 8.0135 5.6347 3.4021 2.4795 2.3015 2.2914 2.1698 2.1315 2.1005 2.0727 2.0511 2.
             vperf: [45.6814 8.4063 5.7367 3.5743 2.8276 2.6191 2.6357 2.5059 2.5110 2.5058 2.4911 2.4833 2.
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
                mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 0.0100 1.0000e-03 1.0000e-03 1.0
          gradient: [156.8157 20.8628 14.8414 17.9479 4.6308 3.2280 4.0307 0.6834 0.9643 0.5410 0.1843 0.478
          val_fail: [0 0 0 0 0 0 1 0 1 0 0 0 1 2 0 0 1 2 3 4 0 0 1 2 3 4 5 6]
         best_perf: 1.8685
        best_vperf: 2.4601
        best_tperf: NaN
Rmse_train = 1×24
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×24
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

           numInputs: 1
           numLayers: 2
          numOutputs: 1
```

```
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 326
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
```

```
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [2 3 6 7 8 9 10 11 14 15 16 18 19 20 21 22 23 24 25 27 28 32 33 35 36 40 42 43 44 45 48
          valInd: [1 4 5 12 13 17 26 29 30 31 34 37 38 39 41 46 47 52 56 61 63 67 69 75 80 84 86 87 89 99
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 15
       trainMask: {[NaN 1 1 NaN NaN 1 1 1 1 1 1 NaN NaN 1 1 1 NaN 1 1 1 1 1 1 1 NaN 1 1 NaN NaN NaN 1 1
         valMask: {[1 NaN NaN 1 1 NaN NaN NaN NaN NaN NaN 1 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN NaN Na
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 9
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
            time: [0.0872 0.1362 0.1696 0.2027 0.2431 0.2810 0.3124 0.3485 0.3801 0.4132 0.4521 0.4851 0.5
            perf: [53.2038 10.9155 5.9011 4.3590 3.5593 2.5988 2.3751 2.2561 2.2162 2.1837 2.1625 2.1419 2
           vperf: [52.3214 10.3226 5.6180 4.1802 3.5635 2.5505 2.3593 2.2786 2.2627 2.2612 2.2742 2.2789 2
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [127.1714 32.6361 35.4185 11.6741 14.3451 7.4009 8.1689 2.7177 2.1054 0.8538 0.7568 0.67
        val_fail: [0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
       best_perf: 2.1837
      best_vperf: 2.2612
      best_tperf: NaN
Rmse_train = 1×25
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×25
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 339
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}
```

```
          inputs: {1x1 cell array of 1 input}
          layers: {2x1 cell array of 2 layers}
         outputs: {1x2 cell array of 1 output}
          biases: {2x1 cell array of 2 biases}
    inputWeights: {2x1 cell array of 1 weight}
    layerWeights: {2x2 cell array of 1 weight}

  functions:

        adaptFcn: 'adaptwb'
      adaptParam: (none)
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
     divideParam: .trainRatio, .valRatio, .testRatio
      divideMode: 'sample'
         initFcn: 'initlay'
      performFcn: 'mse'
    performParam: .regularization, .normalization
        plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                   'plotregression', 'plotfit'}
      plotParams: {1x5 cell array of 5 params}
        trainFcn: 'trainlm'
      trainParam: .showWindow, .showCommandLine, .show, .epochs,
                  .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                  .mu_inc, .mu_max

  weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

  methods:

           adapt: Learn while in continuous use
       configure: Configure inputs & outputs
          gensim: Generate Simulink model
            init: Initialize weights & biases
         perform: Calculate performance
             sim: Evaluate network outputs given inputs
           train: Train network with examples
            view: View diagram
     unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 4 5 8 10 14 16 18 19 20 21 22 23 26 27 28 29 32 33 35 36 37 38 41 42 43 44 45 46 47
          valInd: [3 6 7 9 11 12 13 15 17 24 25 30 31 34 39 40 54 58 59 61 69 74 77 78 84 86 90 91 93 96 9
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 15
       trainMask: {[1 1 NaN 1 1 NaN NaN 1 NaN 1 NaN NaN NaN 1 NaN 1 NaN 1 1 1 1 1 1 NaN NaN 1 1 1 1 NaN Na
         valMask: {[NaN NaN 1 NaN NaN 1 1 NaN 1 NaN 1 1 1 NaN 1 NaN 1 NaN NaN NaN NaN NaN NaN 1 1 NaN NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 9
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
```

```
          epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
           time: [0.0921 0.1499 0.1837 0.2250 0.2658 0.3032 0.3364 0.3751 0.4105 0.4458 0.4930 0.5323 0.5
           perf: [597.3150 10.4529 5.2408 2.7015 2.3120 2.2717 2.2036 2.1654 2.1212 2.0842 2.0565 2.0321
          vperf: [611.4884 10.0925 5.3017 2.7726 2.4123 2.4030 2.3426 2.3377 2.3320 2.3230 2.3373 2.3291
          tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
             mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
       gradient: [869.5000 22.4730 14.7989 7.9648 2.0354 4.2660 1.0646 1.7728 1.0661 1.7972 1.9432 1.2179
       val_fail: [0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
      best_perf: 2.0842
     best_vperf: 2.3230
     best_tperf: NaN
Rmse_train = 1×26
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×26
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

             name: 'Function Fitting Neural Network'
         userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 352
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

           input: Equivalent to inputs{1}
          output: Equivalent to outputs{2}

          inputs: {1x1 cell array of 1 input}
          layers: {2x1 cell array of 2 layers}
         outputs: {1x2 cell array of 1 output}
          biases: {2x1 cell array of 2 biases}
    inputWeights: {2x1 cell array of 1 weight}
    layerWeights: {2x2 cell array of 1 weight}

    functions:

        adaptFcn: 'adaptwb'
      adaptParam: (none)
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
     divideParam: .trainRatio, .valRatio, .testRatio
      divideMode: 'sample'
         initFcn: 'initlay'
      performFcn: 'mse'
    performParam: .regularization, .normalization
        plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
```

```
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
           configure: Configure inputs & outputs
              gensim: Generate Simulink model
                init: Initialize weights & biases
             perform: Calculate performance
                 sim: Evaluate network outputs given inputs
               train: Train network with examples
                view: View diagram
         unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 3 4 5 6 7 8 9 10 11 12 14 16 17 18 19 20 21 23 27 28 30 31 32 33 34 35 36 37 38 39
            valInd: [13 15 22 24 25 26 29 44 45 54 55 56 61 63 66 69 70 71 73 75 77 84 88 91 97 100 102 103
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 25
         trainMask: {[1 1 1 1 1 1 1 1 1 1 1 1 NaN 1 NaN 1 1 1 1 1 1 NaN 1 NaN NaN NaN 1 1 NaN 1 1 1 1 1 1 1
           valMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN NaN NaN NaN 1 NaN
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 19
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]
              time: [0.1058 0.1718 0.2128 0.2607 0.3041 0.3406 0.3833 0.4148 0.4481 0.4900 0.5259 0.5610 0.6
              perf: [64.8451 10.2960 4.6428 2.6531 2.2240 2.1922 2.1232 2.0791 2.0542 2.0326 2.0150 1.9999 1
             vperf: [67.4440 10.8230 4.7012 2.7845 2.3812 2.4063 2.3605 2.3651 2.3510 2.3514 2.3483 2.3474 2
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
                mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
          gradient: [141.9756 31.6585 14.7169 8.1142 0.9178 3.6007 3.3911 0.5430 0.9747 0.5036 0.4209 0.1953
          val_fail: [0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
         best_perf: 1.9162
        best_vperf: 2.3264
        best_tperf: NaN
Rmse_train = 1×27
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×27
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network
```

```
          name: 'Function Fitting Neural Network'
      userdata: (your custom info)

   dimensions:

       numInputs: 1
       numLayers: 2
      numOutputs: 1
   numInputDelays: 0
   numLayerDelays: 0
numFeedbackDelays: 0
numWeightElements: 365
      sampleTime: 1

   connections:

     biasConnect: [1; 1]
    inputConnect: [1; 0]
    layerConnect: [0 0; 1 0]
   outputConnect: [0 1]

   subobjects:

           input: Equivalent to inputs{1}
          output: Equivalent to outputs{2}

          inputs: {1x1 cell array of 1 input}
          layers: {2x1 cell array of 2 layers}
         outputs: {1x2 cell array of 1 output}
          biases: {2x1 cell array of 2 biases}
    inputWeights: {2x1 cell array of 1 weight}
    layerWeights: {2x2 cell array of 1 weight}

   functions:

        adaptFcn: 'adaptwb'
      adaptParam: (none)
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
     divideParam: .trainRatio, .valRatio, .testRatio
      divideMode: 'sample'
         initFcn: 'initlay'
      performFcn: 'mse'
    performParam: .regularization, .normalization
        plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                  'plotregression', 'plotfit'}
      plotParams: {1x5 cell array of 5 params}
        trainFcn: 'trainlm'
      trainParam: .showWindow, .showCommandLine, .show, .epochs,
                  .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                  .mu_inc, .mu_max

   weight and bias values:

              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

   methods:

           adapt: Learn while in continuous use
       configure: Configure inputs & outputs
          gensim: Generate Simulink model
            init: Initialize weights & biases
```

```
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 3 5 7 8 9 13 15 16 17 18 22 24 25 26 28 30 32 33 34 36 37 38 39 41 42 43 44 45 46 4
          valInd: [4 6 10 11 12 14 19 20 21 23 27 29 31 35 40 55 56 60 62 63 71 72 74 75 79 81 89 91 92 99
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 13
       trainMask: {[1 1 1 NaN 1 NaN 1 1 1 NaN NaN NaN 1 NaN 1 1 1 1 NaN NaN NaN 1 NaN 1 1 1 NaN 1 NaN 1 Na
         valMask: {[NaN NaN NaN 1 NaN 1 NaN NaN NaN 1 1 1 NaN 1 NaN NaN NaN NaN 1 1 1 NaN 1 NaN NaN NaN 1
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 7
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13]
            time: [0.0332 0.0872 0.1301 0.1681 0.2114 0.2485 0.2851 0.3282 0.3724 0.4111 0.4552 0.4943 0.5
            perf: [62.9701 7.8396 3.0398 2.9817 2.3196 2.1730 2.1258 2.0923 2.0693 2.0422 2.0210 2.0002 1.
           vperf: [63.1881 7.6644 3.0652 3.0028 2.3305 2.3231 2.2879 2.2866 2.2906 2.2946 2.3053 2.3099 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [184.3106 20.7952 8.9669 19.9963 7.1881 4.8534 3.3129 2.0069 2.0657 0.2591 0.6374 0.5930
        val_fail: [0 0 0 0 0 0 0 0 1 2 3 4 5 6]
       best_perf: 2.0923
      best_vperf: 2.2866
      best_tperf: NaN
Rmse_train = 1×28
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×28
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 378
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
```

```
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [2 3 4 6 7 8 9 11 12 13 15 16 17 20 21 22 23 24 25 26 27 28 31 32 33 34 38 39 42 43 45 4
            valInd: [1 5 10 14 18 19 29 30 35 36 37 40 41 44 50 51 52 58 64 67 68 69 72 76 88 91 92 93 99 10
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
```

```
      num_epochs: 15
       trainMask: {[NaN 1 1 1 NaN 1 1 1 1 NaN 1 1 1 NaN 1 1 1 NaN NaN 1 1 1 1 1 1 1 1 NaN NaN 1 1 1 1 Na
         valMask: {[1 NaN NaN NaN 1 NaN NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN 1 1 NaN NaN NaN NaN NaN Na
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 9
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
            time: [0.1089 0.1742 0.2119 0.2574 0.2956 0.3422 0.3853 0.4357 0.4897 0.5383 0.5873 0.6363 0.6
            perf: [325.6918 16.8776 7.5566 7.3533 5.3344 4.4040 3.3655 2.3847 2.0403 1.9714 1.9450 1.9298
           vperf: [320.9190 16.2668 7.8317 7.6308 5.3712 4.6156 3.5112 2.5763 2.3950 2.3736 2.3807 2.3950
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03
        gradient: [606.6418 60.8784 30.5914 48.1369 27.9011 34.3824 30.1731 8.3576 4.9532 1.1670 0.3757 0.
        val_fail: [0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
       best_perf: 1.9714
      best_vperf: 2.3736
      best_tperf: NaN
Rmse_train = 1×29
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×29
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 391
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
```

```
          divideFcn: 'dividerand'
        divideParam: .trainRatio, .valRatio, .testRatio
         divideMode: 'sample'
            initFcn: 'initlay'
         performFcn: 'mse'
       performParam: .regularization, .normalization
           plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                      'plotregression', 'plotfit'}
         plotParams: {1x5 cell array of 5 params}
           trainFcn: 'trainlm'
         trainParam: .showWindow, .showCommandLine, .show, .epochs,
                     .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                     .mu_inc, .mu_max

    weight and bias values:

                  IW: {2x1 cell} containing 1 input weight matrix
                  LW: {2x2 cell} containing 1 layer weight matrix
                   b: {2x1 cell} containing 2 bias vectors

    methods:

               adapt: Learn while in continuous use
           configure: Configure inputs & outputs
              gensim: Generate Simulink model
                init: Initialize weights & biases
             perform: Calculate performance
                 sim: Evaluate network outputs given inputs
               train: Train network with examples
                view: View diagram
         unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [3 4 5 6 8 10 11 12 13 15 16 17 19 20 21 22 23 26 27 28 29 31 32 33 35 36 38 39 40 41 42
            valInd: [1 2 7 9 14 18 24 25 30 34 37 46 47 49 55 57 58 69 70 74 75 88 93 97 103 110 112 125 130
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 13
         trainMask: {[NaN NaN 1 1 1 1 NaN 1 NaN 1 1 1 1 NaN 1 1 1 NaN 1 1 1 1 1 NaN NaN 1 1 1 1 NaN 1 1 1 Na
           valMask: {[1 1 NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN 1 1 Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 7
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13]
              time: [0.0922 0.1607 0.2105 0.2601 0.3047 0.3579 0.4108 0.4616 0.5196 0.5978 0.6739 0.7163 0.7
              perf: [42.8596 9.8314 9.4481 4.8536 4.3428 2.4700 2.1967 2.1045 2.0717 2.0483 2.0153 1.9974 1.
             vperf: [40.4708 9.7928 9.4400 4.9353 4.3088 2.6263 2.4301 2.4205 2.4314 2.4272 2.4306 2.4228 2.
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
                mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03
          gradient: [86.1364 31.8914 60.1886 19.5142 33.4200 12.2466 3.0929 0.9335 1.1982 2.9517 0.7357 0.39
          val_fail: [0 0 0 0 0 0 0 0 1 2 3 4 5 6]
         best_perf: 2.1045
        best_vperf: 2.4205
        best_tperf: NaN
Rmse_train = 1×30
```

```
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×30
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 404
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors
```

```
    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
         trainFcn: 'trainlm'
       trainParam: [1×1 struct]
       performFcn: 'mse'
     performParam: [1×1 struct]
         derivFcn: 'defaultderiv'
        divideFcn: 'dividerand'
       divideMode: 'sample'
      divideParam: [1×1 struct]
         trainInd: [2 3 4 5 7 8 9 11 12 13 15 16 17 18 19 21 22 23 24 26 29 30 31 32 33 35 36 38 39 40 41 4
           valInd: [1 6 10 14 20 25 27 28 34 37 44 46 50 54 67 68 74 76 77 82 83 85 87 88 93 95 96 97 102 1
          testInd: [1×0 double]
             stop: 'Training finished: Met validation criterion'
       num_epochs: 14
        trainMask: {[NaN 1 1 1 1 NaN 1 1 1 NaN 1 1 1 NaN 1 1 1 1 1 NaN 1 1 1 1 NaN 1 NaN NaN 1 1 1 1 1 NaN
          valMask: {[1 NaN NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN 1 NaN NaN NaN NaN
         testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
       best_epoch: 8
             goal: 0
           states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
            epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
             time: [0.0966 0.1641 0.2171 0.2848 0.3303 0.3930 0.4392 0.4895 0.5393 0.5910 0.6343 0.6780 0.7
             perf: [54.2743 13.3314 4.2452 3.7823 3.3000 2.5516 2.2600 2.1260 2.0802 2.0573 2.0163 1.9967 1
            vperf: [56.6417 13.6544 4.5326 3.9323 3.3953 2.7861 2.4396 2.3650 2.3438 2.3537 2.3562 2.3636 2
            tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
               mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03
         gradient: [179.6041 72.4705 8.2330 28.0386 10.1172 13.5663 4.3808 1.6994 0.8922 2.8154 1.0297 3.14
         val_fail: [0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
        best_perf: 2.0802
       best_vperf: 2.3438
      best_tperf: NaN
Rmse_train = 1×31
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×31
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 417
```

```
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
```

```
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 3 4 5 6 7 8 9 10 11 12 15 16 17 18 19 20 21 23 24 25 26 27 30 32 33 34 35 38 41 42
          valInd: [13 14 22 28 29 31 36 37 39 40 43 46 47 49 51 56 62 65 66 68 71 75 78 79 82 84 85 91 95
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 14
       trainMask: {[1 1 1 1 1 1 1 1 1 1 1 1 NaN NaN 1 1 1 1 1 1 1 NaN 1 1 1 1 1 NaN NaN 1 NaN 1 1 1 1 NaN
         valMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 1 1 NaN NaN NaN NaN NaN NaN NaN 1 NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 8
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
            time: [0.1063 0.1646 0.2214 0.2734 0.3188 0.3631 0.4122 0.4546 0.4992 0.5511 0.5971 0.6436 0.6
            perf: [141.8165 7.8118 3.3733 2.5790 2.3600 2.2449 2.1866 2.1426 2.0719 2.0318 2.0043 1.9799 1
           vperf: [139.6884 7.6359 3.3235 2.5337 2.3136 2.1968 2.1125 2.1249 2.0790 2.0825 2.0855 2.0944 2
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [435.0634 50.2494 19.2073 10.8408 5.5503 2.3062 3.4273 4.3750 1.4814 0.3470 0.6287 0.782
        val_fail: [0 0 0 0 0 0 0 1 0 1 2 3 4 5 6]
       best_perf: 2.0719
      best_vperf: 2.0790
      best_tperf: NaN
Rmse_train = 1×32
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×32
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 430
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
```

```
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 3 6 8 9 11 12 13 16 17 18 19 21 22 23 24 25 26 28 29 31 32 34 35 36 37 40 41 42 44 45
            valInd: [2 4 5 7 10 14 15 20 27 30 33 38 39 43 50 51 52 53 56 58 59 61 64 67 68 74 77 79 84 90 9
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 20
         trainMask: {[1 NaN 1 NaN NaN 1 NaN 1 1 NaN 1 1 1 NaN NaN 1 1 1 1 NaN 1 1 1 1 1 1 NaN 1 1 NaN 1 1 Na
           valMask: {[NaN 1 NaN 1 1 NaN 1 NaN NaN 1 NaN NaN NaN 1 1 NaN NaN NaN NaN 1 NaN NaN NaN NaN NaN Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 14
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]
              time: [0.1336 0.2262 0.2853 0.3590 0.4181 0.4947 0.5490 0.6424 0.7172 0.7942 0.8640 0.9343 0.9
              perf: [92.1386 14.2543 12.9870 11.7243 11.6765 11.4652 3.6086 2.9676 2.9074 2.4145 2.3016 2.26
             vperf: [96.3521 14.0229 12.6621 11.7153 11.3339 11.9114 3.3977 2.9122 2.9513 2.4192 2.3535 2.32
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
```

```
             mu: [1.0000e-03 1.0000e-04 1.0000e-05 1.0000e-06 1.0000e-06 1.0000e-06 1.0000e-07 1.0000e-05
       gradient: [200.3026 75.2466 36.5620 26.9418 55.5430 50.0135 14.0153 7.2738 10.4697 8.0852 2.6020 2
       val_fail: [0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 2 3 4 5 6]
      best_perf: 2.1122
     best_vperf: 2.2910
     best_tperf: NaN
Rmse_train = 1×33
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×33
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

             name: 'Function Fitting Neural Network'
         userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 443
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

           input: Equivalent to inputs{1}
          output: Equivalent to outputs{2}

          inputs: {1x1 cell array of 1 input}
          layers: {2x1 cell array of 2 layers}
         outputs: {1x2 cell array of 1 output}
          biases: {2x1 cell array of 2 biases}
    inputWeights: {2x1 cell array of 1 weight}
    layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
```

```
                .mu_inc, .mu_max

    weight and bias values:

               IW: {2x1 cell} containing 1 input weight matrix
               LW: {2x2 cell} containing 1 layer weight matrix
                b: {2x1 cell} containing 2 bias vectors

    methods:

            adapt: Learn while in continuous use
        configure: Configure inputs & outputs
           gensim: Generate Simulink model
             init: Initialize weights & biases
          perform: Calculate performance
              sim: Evaluate network outputs given inputs
            train: Train network with examples
             view: View diagram
      unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 3 5 6 8 11 12 14 15 16 17 18 19 23 25 26 27 29 30 31 33 34 35 36 38 39 40 41 42 43
          valInd: [4 7 9 10 13 20 21 22 24 28 32 37 50 52 55 59 66 72 76 78 82 87 89 92 93 96 100 109 111
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 15
       trainMask: {[1 1 1 NaN 1 1 NaN 1 NaN NaN 1 1 NaN 1 1 1 1 1 1 NaN NaN NaN 1 NaN 1 1 1 NaN 1 1 1 NaN
         valMask: {[NaN NaN NaN 1 NaN NaN 1 NaN 1 1 NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 1 1 NaN 1 NaN NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 9
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
            time: [0.1072 0.1796 0.2406 0.3118 0.3680 0.4341 0.4941 0.5591 0.6295 0.7181 0.7805 0.8543 0.9
            perf: [288.3529 16.2913 8.0020 4.9284 4.6983 3.5345 2.2811 2.2176 2.1223 2.0627 2.0449 2.0295
           vperf: [281.3739 16.2857 8.0013 4.7807 4.4620 3.3871 2.1977 2.1758 2.1794 2.1282 2.1490 2.1566
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03
        gradient: [640.0782 94.0387 27.2194 37.7623 36.1852 13.5559 8.6822 5.4272 5.2353 0.5967 0.9044 0.6
        val_fail: [0 0 0 0 0 0 0 0 1 0 1 2 3 4 5 6]
       best_perf: 2.0627
      best_vperf: 2.1282
      best_tperf: NaN
Rmse_train = 1×34
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×34
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:
```

```
        numInputs: 1
        numLayers: 2
       numOutputs: 1
   numInputDelays: 0
   numLayerDelays: 0
numFeedbackDelays: 0
numWeightElements: 456
       sampleTime: 1

   connections:

      biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

   subobjects:

            input: Equivalent to inputs{1}
           output: Equivalent to outputs{2}

           inputs: {1x1 cell array of 1 input}
           layers: {2x1 cell array of 2 layers}
          outputs: {1x2 cell array of 1 output}
           biases: {2x1 cell array of 2 biases}
     inputWeights: {2x1 cell array of 1 weight}
     layerWeights: {2x2 cell array of 1 weight}

   functions:

         adaptFcn: 'adaptwb'
       adaptParam: (none)
         derivFcn: 'defaultderiv'
        divideFcn: 'dividerand'
      divideParam: .trainRatio, .valRatio, .testRatio
       divideMode: 'sample'
          initFcn: 'initlay'
       performFcn: 'mse'
     performParam: .regularization, .normalization
         plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
       plotParams: {1x5 cell array of 5 params}
         trainFcn: 'trainlm'
       trainParam: .showWindow, .showCommandLine, .show, .epochs,
                   .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                   .mu_inc, .mu_max

   weight and bias values:

               IW: {2x1 cell} containing 1 input weight matrix
               LW: {2x2 cell} containing 1 layer weight matrix
                b: {2x1 cell} containing 2 bias vectors

   methods:

            adapt: Learn while in continuous use
        configure: Configure inputs & outputs
           gensim: Generate Simulink model
             init: Initialize weights & biases
          perform: Calculate performance
              sim: Evaluate network outputs given inputs
            train: Train network with examples
             view: View diagram
```

```
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [1 2 3 4 5 6 7 8 9 11 13 16 18 19 20 21 22 23 24 25 26 27 28 30 31 33 34 36 37 38 40 42
          valInd: [10 12 14 15 17 29 32 35 39 41 45 49 50 56 58 61 62 66 68 74 79 86 89 91 94 100 103 110
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 18
       trainMask: {[1 1 1 1 1 1 1 1 NaN 1 NaN 1 NaN NaN 1 NaN 1 1 1 1 1 1 1 1 1 1 NaN 1 1 NaN 1 1 NaN
         valMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN 1 1 NaN 1 NaN NaN NaN NaN NaN NaN NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 12
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18]
            time: [0.1025 0.1736 0.2254 0.2934 0.3689 0.4479 0.5009 0.5857 0.6798 0.7529 0.8331 0.9174 0.9
            perf: [95.1538 8.7209 7.9943 6.5441 2.4411 2.1342 2.0097 1.9679 1.9489 1.9311 1.9152 1.9003 1.
           vperf: [93.9578 9.5312 7.7791 6.8610 2.6335 2.5101 2.4254 2.4198 2.4174 2.4200 2.4167 2.4111 2.
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [255.6680 74.1746 27.4186 38.9863 8.1140 7.5826 1.5378 0.8525 1.4124 0.5699 0.4083 0.426
        val_fail: [0 0 0 0 0 0 0 0 0 1 0 0 0 1 2 3 4 5 6]
       best_perf: 1.8862
      best_vperf: 2.4078
      best_tperf: NaN
Rmse_train = 1×35
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×35
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 469
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:
```

```
          input: Equivalent to inputs{1}
         output: Equivalent to outputs{2}

         inputs: {1x1 cell array of 1 input}
         layers: {2x1 cell array of 2 layers}
        outputs: {1x2 cell array of 1 output}
         biases: {2x1 cell array of 2 biases}
   inputWeights: {2x1 cell array of 1 weight}
   layerWeights: {2x2 cell array of 1 weight}

  functions:

       adaptFcn: 'adaptwb'
     adaptParam: (none)
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
    divideParam: .trainRatio, .valRatio, .testRatio
     divideMode: 'sample'
        initFcn: 'initlay'
     performFcn: 'mse'
   performParam: .regularization, .normalization
       plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                  'plotregression', 'plotfit'}
     plotParams: {1x5 cell array of 5 params}
       trainFcn: 'trainlm'
     trainParam: .showWindow, .showCommandLine, .show, .epochs,
                 .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                 .mu_inc, .mu_max

  weight and bias values:

              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

  methods:

           adapt: Learn while in continuous use
       configure: Configure inputs & outputs
          gensim: Generate Simulink model
            init: Initialize weights & biases
         perform: Calculate performance
             sim: Evaluate network outputs given inputs
           train: Train network with examples
            view: View diagram
     unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
       trainFcn: 'trainlm'
     trainParam: [1×1 struct]
     performFcn: 'mse'
   performParam: [1×1 struct]
       derivFcn: 'defaultderiv'
      divideFcn: 'dividerand'
     divideMode: 'sample'
    divideParam: [1×1 struct]
       trainInd: [1 2 3 4 5 6 8 10 11 12 14 16 17 18 19 20 23 24 27 28 30 32 33 35 36 37 38 39 41 43 44 4
         valInd: [7 9 13 15 21 22 25 26 29 31 34 40 42 49 50 53 59 60 64 66 68 75 80 83 87 91 94 95 100 1
        testInd: [1×0 double]
           stop: 'Training finished: Met validation criterion'
     num_epochs: 11
      trainMask: {[1 1 1 1 1 1 NaN 1 NaN 1 1 1 NaN 1 NaN 1 1 1 1 1 NaN NaN 1 1 NaN NaN 1 1 NaN 1 NaN 1 1
        valMask: {[NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN 1 NaN 1 NaN NaN NaN NaN NaN 1 1 NaN NaN 1
       testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
```

```
      best_epoch: 5
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11]
            time: [0.1055 0.1794 0.2372 0.3015 0.3585 0.4184 0.4718 0.5327 0.5883 0.6527 0.7022 0.7604]
            perf: [47.0971 5.2013 2.6650 2.3384 2.2924 2.0907 2.0381 1.9966 1.9679 1.9349 1.8992 1.8755]
           vperf: [45.2452 4.9111 2.8433 2.5691 2.5498 2.4283 2.4529 2.4722 2.4879 2.5066 2.5180 2.5397]
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [132.5746 30.8131 9.3553 3.1085 4.3953 2.4473 1.0925 1.4027 3.0502 2.4156 1.2383 1.1586]
        val_fail: [0 0 0 0 0 0 1 2 3 4 5 6]
       best_perf: 2.0907
      best_vperf: 2.4283
      best_tperf: NaN
Rmse_train = 1×36
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×36
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 482
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
```

```
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 3 4 5 6 8 10 11 12 13 15 16 17 19 20 21 22 23 24 25 27 28 29 30 31 34 35 36 37 38 4
            valInd: [7 9 14 18 26 32 33 39 40 42 49 54 56 59 60 64 65 66 78 81 83 84 86 87 90 94 101 104 108
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 18
         trainMask: {[1 1 1 1 1 1 NaN 1 NaN 1 1 1 1 NaN 1 1 1 NaN 1 1 1 1 1 1 1 NaN 1 1 1 1 1 NaN NaN 1 1 1
           valMask: {[NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 12
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18]
              time: [0.1039 0.1806 0.2424 0.3244 0.3815 0.4549 0.5096 0.5764 0.6403 0.7037 0.7577 0.8224 0.8
              perf: [55.4945 12.4607 4.8240 2.6138 2.1236 2.0302 1.9685 1.9221 1.8845 1.8575 1.8275 1.8068 1
             vperf: [57.2255 12.3037 5.4810 2.9514 2.5646 2.5548 2.5474 2.5565 2.5573 2.5594 2.5520 2.5462 2
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
                mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
          gradient: [169.1250 48.0191 13.8170 8.1007 1.6397 2.2894 2.0487 1.0308 0.8591 2.7224 2.0249 1.8414
          val_fail: [0 0 0 0 0 0 0 1 2 3 4 0 0 1 2 3 4 5 6]
         best_perf: 1.7892
        best_vperf: 2.5443
        best_tperf: NaN
Rmse_train = 1×37
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×37
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
```

```
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 495
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:
```

```
             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
        trainFcn: 'trainlm'
      trainParam: [1×1 struct]
      performFcn: 'mse'
    performParam: [1×1 struct]
        derivFcn: 'defaultderiv'
       divideFcn: 'dividerand'
      divideMode: 'sample'
     divideParam: [1×1 struct]
        trainInd: [2 3 4 5 6 7 8 9 13 14 15 16 17 20 21 22 23 25 26 27 28 30 32 34 35 36 40 41 42 43 44 46
          valInd: [1 10 11 12 18 19 24 29 31 33 37 38 39 45 48 49 52 56 62 74 84 86 87 88 89 91 102 106 11
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 11
       trainMask: {[NaN 1 1 1 1 1 1 1 NaN NaN NaN 1 1 1 1 1 NaN NaN 1 1 1 1 NaN 1 1 1 1 NaN 1 NaN 1 NaN
         valMask: {[1 NaN NaN NaN NaN NaN NaN NaN NaN 1 1 1 NaN NaN NaN NaN NaN 1 1 NaN NaN NaN NaN 1 NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 5
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11]
            time: [0.1130 0.1801 0.2392 0.3170 0.3782 0.4575 0.5196 0.5975 0.6631 0.7360 0.8010 0.8748]
            perf: [103.2342 26.3342 6.0149 2.7487 2.4048 2.1010 2.0559 2.0029 1.9693 1.9243 1.8981 1.8773]
           vperf: [102.8001 26.2206 5.7689 2.8347 2.6209 2.3207 2.3513 2.3400 2.3556 2.3711 2.3928 2.3948]
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [186.8010 81.3579 31.1910 8.5310 13.0424 1.6912 3.8322 2.8573 3.9705 0.9575 1.0666 0.253
        val_fail: [0 0 0 0 0 0 1 2 3 4 5 6]
       best_perf: 2.1010
      best_vperf: 2.3207
      best_tperf: NaN
Rmse_train = 1×38
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×38
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 508
        sampleTime: 1

    connections:
```

```
       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
```

```
        trainInd: [1 3 4 5 6 8 9 10 12 13 14 15 22 23 24 25 26 27 28 29 31 32 33 34 35 36 38 39 40 44 45 4
          valInd: [2 7 11 16 17 18 19 20 21 30 37 41 42 43 52 53 54 56 57 58 60 62 66 69 70 75 81 85 90 92
         testInd: [1×0 double]
            stop: 'Training finished: Met validation criterion'
      num_epochs: 14
       trainMask: {[1 NaN 1 1 1 1 NaN 1 1 1 NaN 1 1 1 1 NaN NaN NaN NaN NaN NaN 1 1 1 1 1 1 1 NaN 1 1 1
         valMask: {[NaN 1 NaN NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN 1 1 1 1 1 1 NaN NaN NaN NaN NaN
        testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
      best_epoch: 8
            goal: 0
          states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
           epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
            time: [0.1216 0.2008 0.2704 0.3567 0.4277 0.5041 0.5645 0.6599 0.7222 0.8036 0.8730 0.9468 1.0
            perf: [54.2781 10.5608 6.1585 5.5608 2.3823 2.2305 2.0673 2.0060 1.9671 1.9323 1.9095 1.9048 1
           vperf: [53.4847 10.6107 6.2328 5.6913 2.6727 2.5119 2.3920 2.3527 2.3405 2.3500 2.3527 2.3767 2
           tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
              mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
        gradient: [144.0158 46.0989 23.5483 30.5787 4.5251 10.7475 2.9227 2.0400 1.4515 0.5581 0.6998 2.04
        val_fail: [0 0 0 0 0 0 0 0 0 1 2 3 4 5 6]
       best_perf: 1.9671
      best_vperf: 2.3405
      best_tperf: NaN
Rmse_train = 1×39
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×39
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 521
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:
```

```
          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                     'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:

                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

             adapt: Learn while in continuous use
         configure: Configure inputs & outputs
            gensim: Generate Simulink model
              init: Initialize weights & biases
           perform: Calculate performance
               sim: Evaluate network outputs given inputs
             train: Train network with examples
              view: View diagram
       unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [2 3 4 5 7 8 9 10 11 12 13 14 15 16 17 20 22 23 25 26 30 33 34 35 37 38 39 40 41 43 44 4
            valInd: [1 6 18 19 21 24 27 28 29 31 32 36 42 53 56 59 61 63 66 69 71 73 77 78 84 88 90 92 93 97
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 15
         trainMask: {[NaN 1 1 1 1 NaN 1 1 1 1 1 1 1 1 1 1 NaN NaN 1 NaN 1 1 NaN 1 1 NaN NaN NaN 1 NaN NaN
           valMask: {[1 NaN NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN 1 1 NaN 1 NaN NaN 1 Na
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 9
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
              time: [0.1205 0.2032 0.2688 0.3606 0.4418 0.5292 0.6217 0.7183 0.7958 0.8852 0.9420 1.0413 1.1
              perf: [37.3613 17.4483 14.0025 7.1481 5.7981 5.5736 3.9353 2.3109 2.0878 1.9931 1.9630 1.8210
             vperf: [37.9096 18.4887 13.4110 8.0104 5.5235 5.7310 4.5701 2.7192 2.6149 2.5199 2.6634 2.5692
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
                mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03
          gradient: [131.4421 88.1294 59.0066 29.7713 30.9341 46.5507 18.7390 11.3225 8.5259 8.4429 5.7314 0
          val_fail: [0 0 0 0 0 1 0 0 0 0 1 2 3 4 5 6]
```

```
       best_perf: 1.9931
      best_vperf: 2.5199
      best_tperf: NaN
Rmse_train = 1×40
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×40
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
net =

    Neural Network

              name: 'Function Fitting Neural Network'
          userdata: (your custom info)

    dimensions:

         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 534
        sampleTime: 1

    connections:

       biasConnect: [1; 1]
     inputConnect: [1; 0]
     layerConnect: [0 0; 1 0]
    outputConnect: [0 1]

    subobjects:

             input: Equivalent to inputs{1}
            output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

    functions:

          adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                    'plotregression', 'plotfit'}
        plotParams: {1x5 cell array of 5 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max

    weight and bias values:
```

```
                IW: {2x1 cell} containing 1 input weight matrix
                LW: {2x2 cell} containing 1 layer weight matrix
                 b: {2x1 cell} containing 2 bias vectors

    methods:

              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

tr = struct with fields:
          trainFcn: 'trainlm'
        trainParam: [1×1 struct]
        performFcn: 'mse'
      performParam: [1×1 struct]
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
        divideMode: 'sample'
       divideParam: [1×1 struct]
          trainInd: [1 2 4 5 8 9 10 12 13 14 15 16 17 20 21 23 24 25 26 27 28 29 30 31 32 34 35 36 37 38 39
            valInd: [3 6 7 11 18 19 22 33 40 44 49 50 51 53 55 59 66 69 73 74 75 77 88 92 96 104 107 108 119
           testInd: [1×0 double]
              stop: 'Training finished: Met validation criterion'
        num_epochs: 13
         trainMask: {[1 1 NaN 1 1 NaN NaN 1 1 1 NaN 1 1 1 1 1 1 NaN NaN 1 1 NaN 1 1 1 1 1 1 1 1 1 NaN 1 1
           valMask: {[NaN NaN 1 NaN NaN 1 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 1 NaN NaN 1 NaN NaN NaN
          testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN Na
        best_epoch: 7
              goal: 0
            states: {'epoch'  'time'  'perf'  'vperf'  'tperf'  'mu'  'gradient'  'val_fail'}
             epoch: [0 1 2 3 4 5 6 7 8 9 10 11 12 13]
              time: [0.1183 0.1921 0.2601 0.3559 0.4278 0.5167 0.5809 0.6664 0.7284 0.8081 0.8961 0.9972 1.0
              perf: [77.2682 7.1853 7.1008 2.7564 2.0960 2.0287 1.9541 1.9068 1.8795 1.8560 1.8304 1.8100 1.
             vperf: [77.3582 6.9974 7.5952 3.1509 2.4591 2.3904 2.3958 2.3820 2.3967 2.4067 2.4149 2.4209 2.
             tperf: [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
                mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03
          gradient: [284.2018 27.8051 45.3363 10.2742 3.6781 3.3407 3.2235 0.9622 1.6363 1.7523 1.3322 0.959
          val_fail: [0 0 1 0 0 0 1 0 1 2 3 4 5 6]
         best_perf: 1.9068
        best_vperf: 2.3820
        best_tperf: NaN
Rmse_train = 1×41
  559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ···
Rmse_val = 1×41
  559.7966  540.3455  581.9837  488.1037  449.9189  511.2969  342.2379  392.7326 ···
```

Selecting optimal number of neuron in hidden layer

```
plot(1:60,Rmse_train);hold on;
plot(1:60, Rmse_val);hold off;
```

Custom test data prediction

Predict with Indian csv files

```
mytest = readtable('India Dataset.csv');
test= removevars(mytest,{'Var1'});
test=test(~any(ismissing(test),2),:);
test = table2array(test(:,1:n-1));

% y = table2array(test(:,end));

[m,n] = size(test)

for i = 1:n
    test(:,i) = (test(:,i) - min(test(:,i)))/max(test(:,i)-min(test(:,i)));
end

xtest = test';
for i = 1:m
    Indian_output(i,1) = exp(net(xt(:,i)))-1;
end
plot(1:600,Indian_output(1:600,1),'o')
xlabel('No of Days')
ylabel('Predicted Solar power from features')
title('Predicted solar power in Indian dataset')
```

Predict with India Dataset.xlsx files

```
mytest = readtable('MALASYIA 2.csv');
test= removevars(mytest,{'Var1'});
test=test(~any(ismissing(test),2),:);
test = table2array(test(:,1:n-1));

% y = table2array(test(:,end));

[m,n] = size(test)

for i = 1:n
    test(:,i) = (test(:,i) - min(test(:,i)))/max(test(:,i)-min(test(:,i)));
end

xtest = test';
for i = 1:m
    Malasyia_output(i,1) = exp(net(xt(:,i)))-1;
end
plot(1:754,Malasyia_output,'o')
xlabel('No of Days')
ylabel('Predicted Solar power from features')
title('Predicted solar power in Malasyian dataset')
```