

Reading data

Importing the data

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

Data Preprocessing

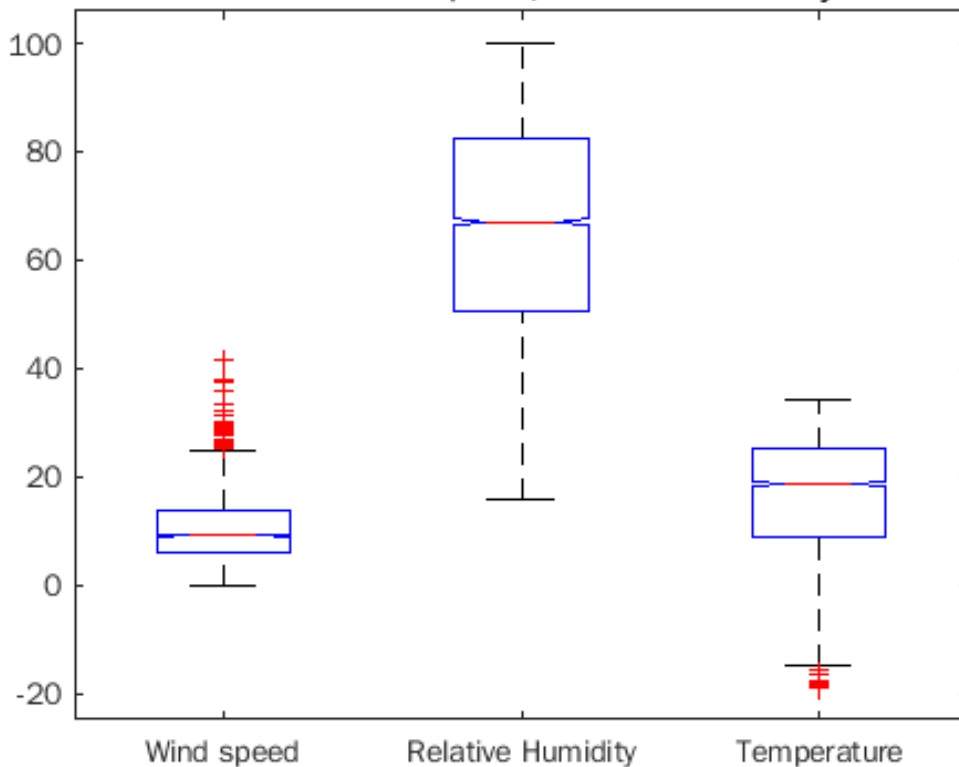
Removing missing values and dropping non relevant columns

```
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', {'V  
title('Detect outliers Based on Wind Speed, Relative Humidity and Temperature');
```

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 = 1:m  
    if data >= validRange(1) & data <= validRange(2)  
        final_table(i,:) = [];
```

```
end  
end
```

Warning: Colon operands must be real scalars.

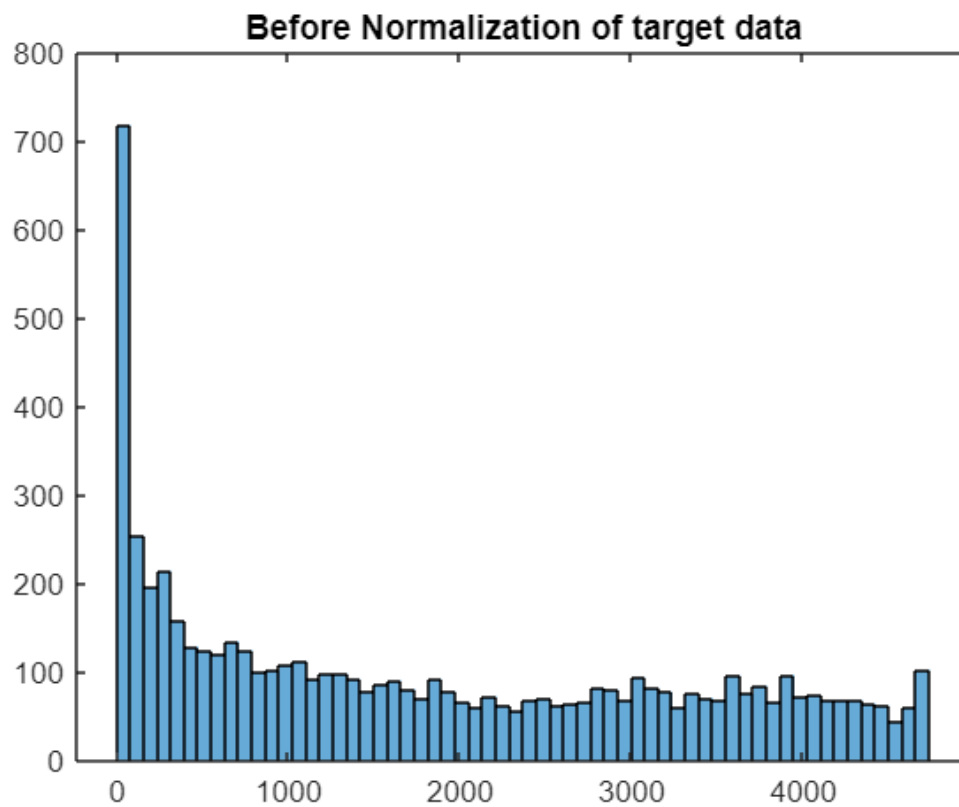
Splitting data as feature and target

```
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')
```



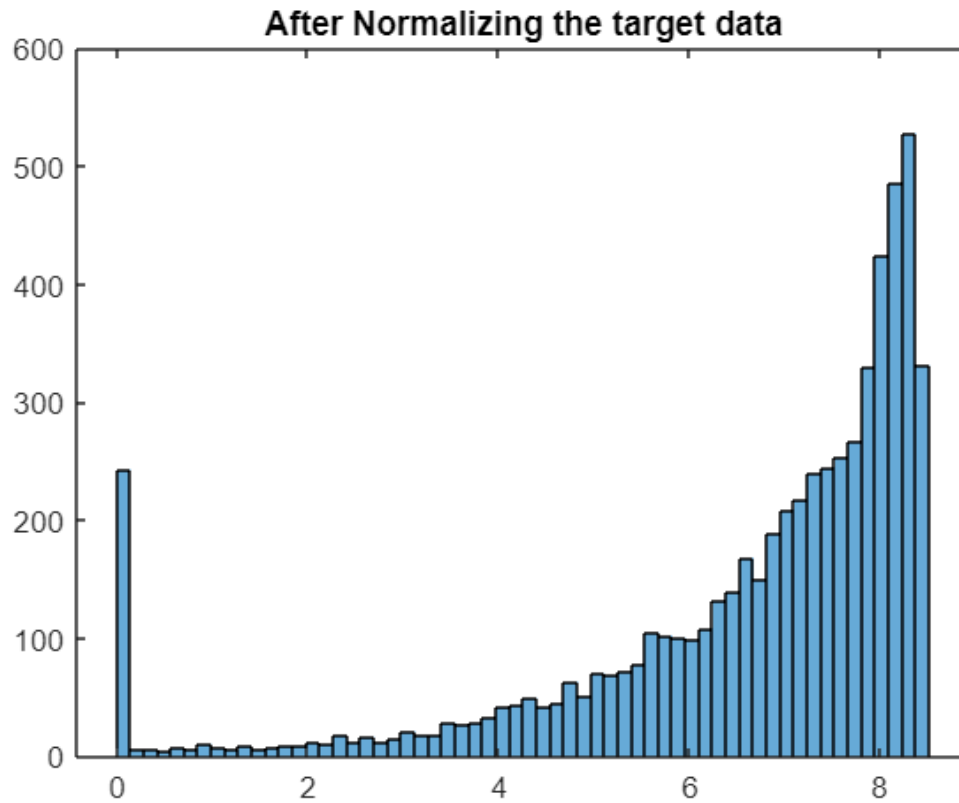
Normalize the Target and transform the output

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

```
y2 = 6028x1  
6.1098  
8.0604  
7.8778  
6.5670  
6.5243  
5.6166  
8.4538
```

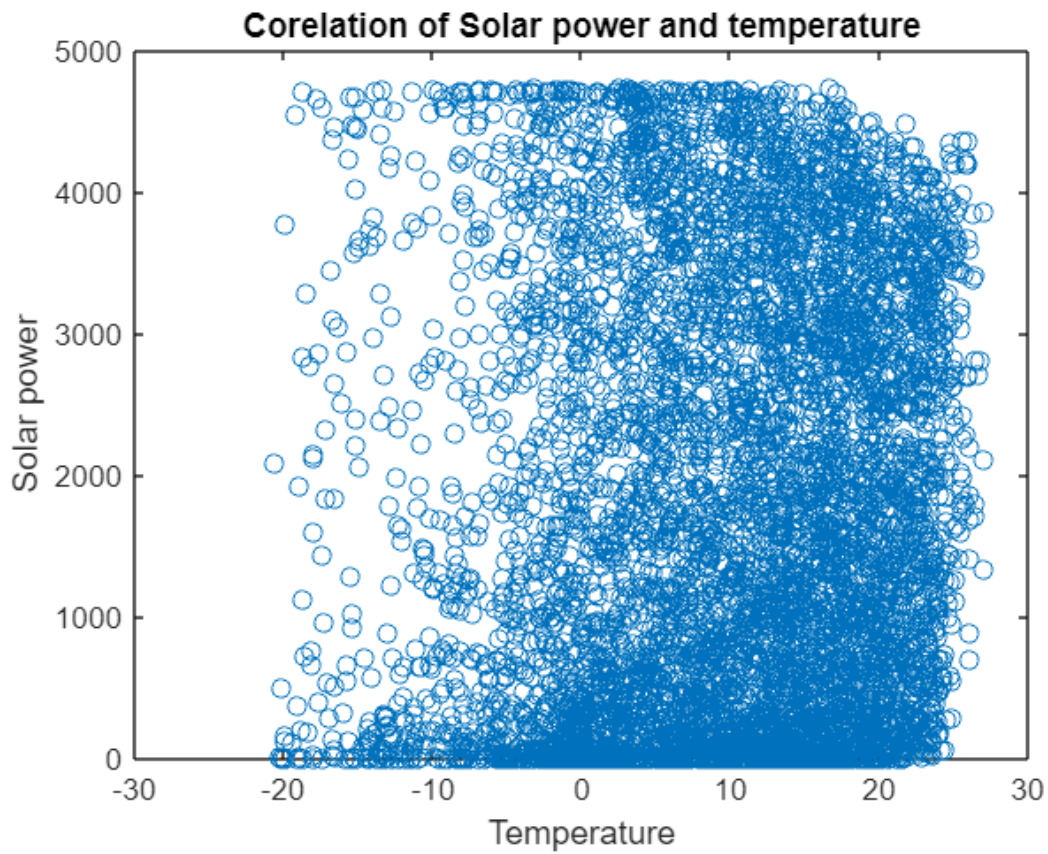
```
8.0067
8.0201
5.6996
⋮
```

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



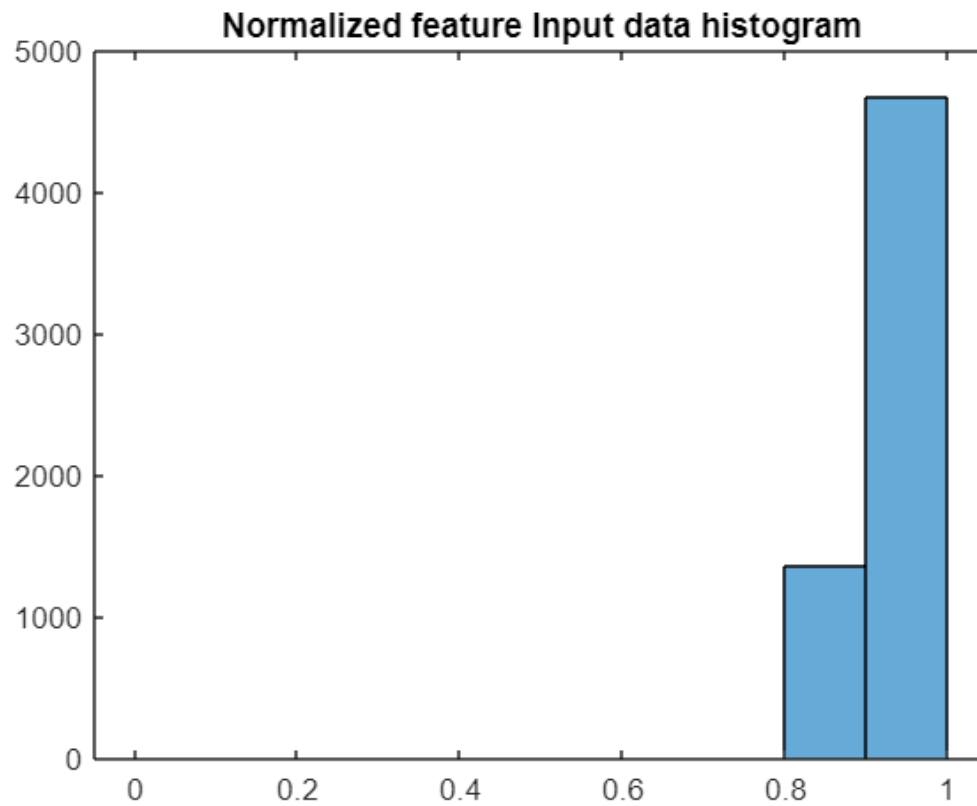
Correlation between solar power and temperature

```
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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'

```

```

    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 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 NaN]
    testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
    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 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 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)
```

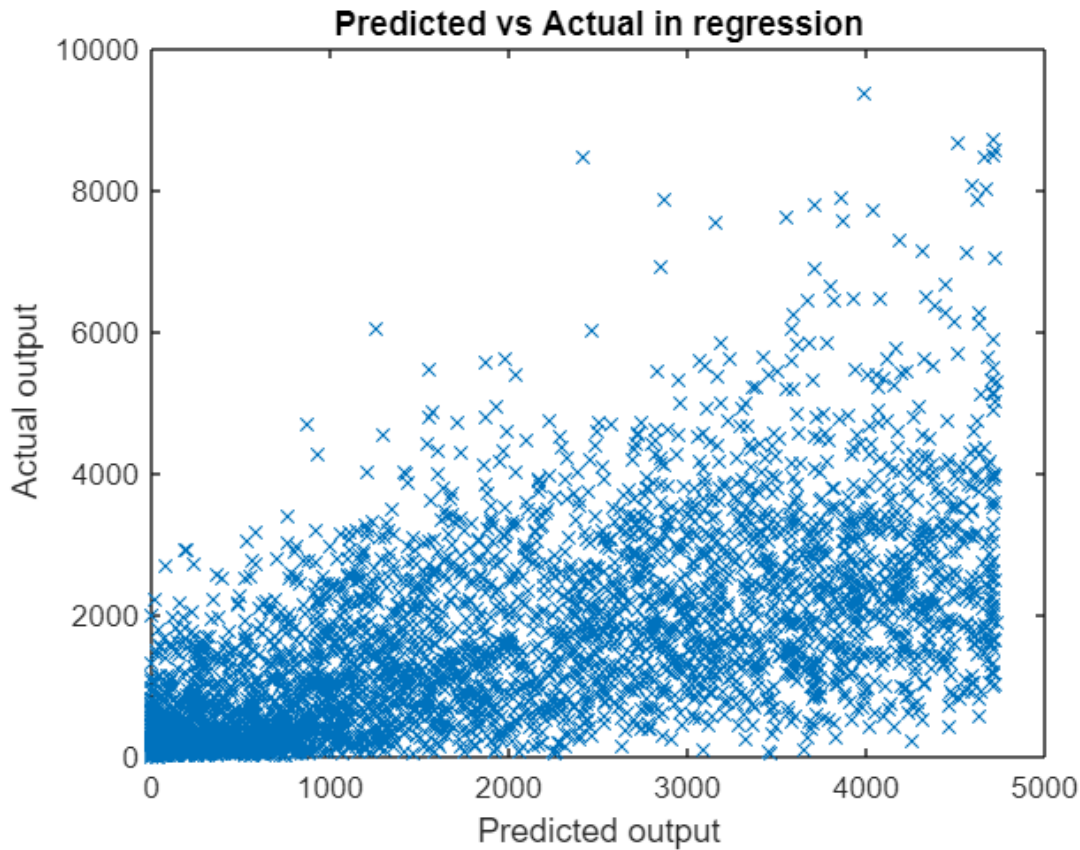
```
Rmse_val = 371.7834
```

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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN]
    testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]
    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 32]
    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 0.3842 0.3885 0.3927 0.3969 0.4011 0.4053 0.4095 0.4137 0.4179 0.4221 0.4263 0.4305 0.4347 0.4389 0.4431 0.4473 0.4515 0.4557 0.4599 0.4641 0.4683 0.4725 0.4767 0.4809 0.4851 0.4893 0.4935 0.4977 0.5019 0.5061 0.5103 0.5145 0.5187 0.5229 0.5271 0.5313 0.5355 0.5397 0.5439 0.5481 0.5523 0.5565 0.5607 0.5649 0.5691 0.5733 0.5775 0.5817 0.5859 0.5901 0.5943 0.5985 0.6027 0.6069 0.6111 0.6153 0.6195 0.6237 0.6279 0.6321 0.6363 0.6405 0.6447 0.6489 0.6531 0.6573 0.6615 0.6657 0.6699 0.6741 0.6783 0.6825 0.6867 0.6909 0.6951 0.6993 0.7035 0.7077 0.7119 0.7161 0.7203 0.7245 0.7287 0.7329 0.7371 0.7413 0.7455 0.7497 0.7539 0.7581 0.7623 0.7665 0.7707 0.7749 0.7791 0.7833 0.7875 0.7917 0.7959 0.8001 0.8043 0.8085 0.8127 0.8169 0.8211 0.8253 0.8295 0.8337 0.8379 0.8421 0.8463 0.8505 0.8547 0.8589 0.8631 0.8673 0.8715 0.8757 0.8799 0.8841 0.8883 0.8925 0.8967 0.9009 0.9051 0.9093 0.9135 0.9177 0.9219 0.9261 0.9303 0.9345 0.9387 0.9429 0.9471 0.9513 0.9555 0.9597 0.9639 0.9681 0.9723 0.9765 0.9807 0.9849 0.9891 0.9933 0.9975 1.0017 1.0059 1.0101 1.0143 1.0185 1.0227 1.0269 1.0311 1.0353 1.0395 1.0437 1.0479 1.0521 1.0563 1.0605 1.0647 1.0689 1.0731 1.0773 1.0815 1.0857 1.0899 1.0941 1.0983 1.1025 1.1067 1.1109 1.1151 1.1193 1.1235 1.1277 1.1319 1.1361 1.1403 1.1445 1.1487 1.1529 1.1571 1.1613 1.1655 1.1697 1.1739 1.1781 1.1823 1.1865 1.1907 1.1949 1.1991 1.2033 1.2075 1.2117 1.2159 1.2201 1.2243 1.2285 1.2327 1.2369 1.2411 1.2453 1.2495 1.2537 1.2579 1.2621 1.2663 1.2705 1.2747 1.2789 1.2831 1.2873 1.2915 1.2957 1.3000 1.3042 1.3084 1.3126 1.3168 1.3210 1.3252 1.3294 1.3336 1.3378 1.3420 1.3462 1.3504 1.3546 1.3588 1.3630 1.3672 1.3714 1.3756 1.3798 1.3840 1.3882 1.3924 1.3966 1.4008 1.4050 1.4092 1.4134 1.4176 1.4218 1.4260 1.4302 1.4344 1.4386 1.4428 1.4470 1.4512 1.4554 1.4596 1.4638 1.4680 1.4722 1.4764 1.4806 1.4848 1.4890 1.4932 1.4974 1.5016 1.5058 1.5100 1.5142 1.5184 1.5226 1.5268 1.5310 1.5352 1.5394 1.5436 1.5478 1.5520 1.5562 1.5604 1.5646 1.5688 1.5730 1.5772 1.5814 1.5856 1.5898 1.5940 1.5982 1.6024 1.6066 1.6108 1.6150 1.6192 1.6234 1.6276 1.6318 1.6360 1.6402 1.6444 1.6486 1.6528 1.6570 1.6612 1.6654 1.6696 1.6738 1.6780 1.6822 1.6864 1.6906 1.6948 1.6990 1.7032 1.7074 1.7116 1.7158 1.7200 1.7242 1.7284 1.7326 1.7368 1.7410 1.7452 1.7494 1.7536 1.7578 1.7620 1.7662 1.7704 1.7746 1.7788 1.7830 1.7872 1.7914 1.7956 1.7998 1.8040 1.8082 1.8124 1.8166 1.8208 1.8250 1.8292 1.8334 1.8376 1.8418 1.8460 1.8502 1.8544 1.8586 1.8628 1.8670 1.8712 1.8754 1.8796 1.8838 1.8880 1.8922 1.8964 1.9006 1.9048 1.9090 1.9132 1.9174 1.9216 1.9258 1.9300 1.9342 1.9384 1.9426 1.9468 1.9510 1.9552 1.9594 1.9636 1.9678 1.9720 1.9762 1.9804 1.9846 1.9888 1.9930 1.9972 2.0014 2.0056 2.0098 2.0140 2.0182 2.0224 2.0266 2.0308 2.0350 2.0392 2.0434 2.0476 2.0518 2.0560 2.0602 2.0644 2.0686 2.0728 2.0770 2.0812 2.0854 2.0896 2.0938 2.0980 2.1022 2.1064 2.1106 2.1148 2.1190 2.1232 2.1274 2.1316 2.1358 2.1400 2.1442 2.1484 2.1526 2.1568 2.1610 2.1652 2.1694 2.1736 2.1778 2.1820 2.1862 2.1904 2.1946 2.1988 2.2030 2.2072 2.2114 2.2156 2.2198 2.2240 2.2282 2.2324 2.2366 2.2408 2.2450 2.2492 2.2534 2.2576 2.2618 2.2660 2.2702 2.2744 2.2786 2.2828 2.2870 2.2912 2.2954 2.2996 2.3038 2.3080 2.3122 2.3164 2.3206 2.3248 2.3290 2.3332 2.3374 2.3416 2.3458 2.3500 2.3542 2.3584 2.3626 2.3668 2.3710 2.3752 2.3794 2.3836 2.3878 2.3920 2.3962 2.4004 2.4046 2.4088 2.4130 2.4172 2.4214 2.4256 2.4298 2.4340 2.4382 2.4424 2.4466 2.4508 2.4550 2.4592 2.4634 2.4676 2.4718 2.4760 2.4802 2.4844 2.4886 2.4928 2.4970 2.5012 2.5054 2.5096 2.5138 2.5180 2.5222 2.5264 2.5306 2.5348 2.5390 2.5432 2.5474 2.5516 2.5558 2.5600 2.5642 2.5684 2.5726 2.5768 2.5810 2.5852 2.5894 2.5936 2.5978 2.6020 2.6062 2.6104 2.6146 2.6188 2.6230 2.6272 2.6314 2.6356 2.6398 2.6440 2.6482 2.6524 2.6566 2.6608 2.6650 2.6692 2.6734 2.6776 2.6818 2.6860 2.6902 2.6944 2.6986 2.7028 2.7070 2.7112 2.7154 2.7196 2.7238 2.7280 2.7322 2.7364 2.7406 2.7448 2.7490 2.7532 2.7574 2.7616 2.7658 2.7700 2.7742 2.7784 2.7826 2.7868 2.7910 2.7952 2.7994 2.8036 2.8078 2.8120 2.8162 2.8204 2.8246 2.8288 2.8330 2.8372 2.8414 2.8456 2.8498 2.8540 2.8582 2.8624 2.8666 2.8708 2.8750 2.8792 2.8834 2.8876 2.8918 2.8960 2.9002 2.9044 2.9086 2.9128 2.9170 2.9212 2.9254 2.9296 2.9338 2.9380 2.9422 2.9464 2.9506 2.9548 2.9590 2.9632 2.9674 2.9716 2.9758 2.9800 2.9842 2.9884 2.9926 2.9968 3.0010 3.0052 3.0094 3.0136 3.0178 3.0220 3.0262 3.0304 3.0346 3.0388 3.0430 3.0472 3.0514 3.0556 3.0598 3.0640 3.0682 3.0724 3.0766 3.0808 3.0850 3.0892 3.0934 3.0976 3.1018 3.1060 3.1102 3.1144 3.1186 3.1228 3.1270 3.1312 3.1354 3.1396 3.1438 3.1480 3.1522 3.1564 3.1606 3.1648 3.1690 3.1732 3.1774 3.1816 3.1858 3.1900 3.1942 3.1984 3.2026 3.2068 3.2110 3.2152 3.2194 3.2236 3.2278 3.2320 3.2362 3.2404 3.2446 3.2488 3.2530 3.2572 3.2614 3.2656 3.2698 3.2740 3.2782 3.2824 3.2866 3.2908 3.2950 3.2992 3.3034 3.3076 3.3118 3.3160 3.3202 3.3244 3.3286 3.3328 3.3370 3.3412 3.3454 3.3496 3.3538 3.3580 3.3622 3.3664 3.3706 3.3748 3.3790 3.3832 3.3874 3.3916 3.3958 3.4000 3.4042 3.4084 3.4126 3.4168 3.4210 3.4252 3.4294 3.4336 3.4378 3.4420 3.4462 3.4504 3.4546 3.4588 3.4630 3.4672 3.4714 3.4756 3.4798 3.4840 3.4882 3.4924 3.4966 3.5008 3.5050 3.5092 3.5134 3.5176 3.5218 3.5260 3.5302 3.5344 3.5386 3.5428 3.5470 3.5512 3.5554 3.5596 3.5638 3.5680 3.5722 3.5764 3.5806 3.5848 3.5890 3.5932 3.5974 3.6016 3.6058 3.6100 3.6142 3.6184 3.6226 3.6268 3.6310 3.6352 3.6394 3.6436 3.6478 3.6520 3.6562 3.6604 3.6646 3.6688 3.6730 3.6772 3.6814 3.6856 3.6898 3.6940 3.6982 3.7024 3.7066 3.7108 3.7150 3.7192 3.7234 3.7276 3.7318 3.7360 3.7402 3.7444 3.7486 3.7528 3.7570 3.7612 3.7654 3.7696 3.7738 3.7780 3.7822 3.7864 3.7906 3.7948 3.7990 3.8032 3.8074 3.8116 3.8158 3.8200 3.8242 3.8284 3.8326 3.8368 3.8410 3.8452 3.8494 3.8536 3.8578 3.8620 3.8662 3.8704 3.8746 3.8788 3.8830 3.8872 3.8914 3.8956 3.8998 3.9040 3.9082 3.9124 3.9166 3.9208 3.9250 3.9292 3.9334 3.9376 3.9418 3.9460 3.9502 3.9544 3.9586 3.9628 3.9670 3.9712 3.9754 3.9796 3.9838 3.9880 3.9922 3.9964 4.0006 4.0048 4.0090 4.0132 4.0174 4.0216 4.0258 4.0300 4.0342 4.0384 4.0426 4.0468 4.0510 4.0552 4.0594 4.0636 4.0678 4.0720 4.0762 4.0804 4.0846 4.0888 4.0930 4.0972 4.1014 4.1056 4.1098 4.1140 4.1182 4.1224 4.1266 4.1308 4.1350 4.1392 4.1434 4.1476 4.1518 4.1560 4.1602 4.1644 4.1686 4.1728 4.1770 4.1812 4.1854 4.1896 4.1938 4.1980 4.2022 4.2064 4.2106 4.2148 4.2190 4.2232 4.2274 4.2316 4.2358 4.2400 4.2442 4.2484 4.2526 4.2568 4.2610 4.2652 4.2694 4.2736 4.2778 4.2820 4.2862 4.2904 4.2946 4.2988 4.3030 4.3072 4.3114 4.3156 4.3198 4.3240 4.3282 4.3324 4.3366 4.3408 4.3450 4.3492 4.3534 4.3576 4.3618 4.3660 4.3702 4.3744 4.3786 4.3828 4.3870 4.3912 4.3954 4.3996 4.4038 4.4080 4.4122 4.4164 4.4206 4.4248 4.4290 4.4332 4.4374 4.4416 4.4458 4.4500 4.4542 4.4584 4.4626 4.4668 4.4710 4.4752 4.4794 4.4836 4.4878 4.4920 4.4962 4.5004 4.5046 4.5088 4.5130 4.5172 4.5214 4.5256 4.5298 4.5340 4.5382 4.5424 4.5466 4.5508 4.5550 4.5592 4.5634 4.5676 4.5718 4.5760 4.5802 4.5844 4.5886 4.5928 4.5970 4.6012 4.6054 4.6096 4.6138 4.6180 4.6222 4.6264 4.6306 4.6348 4.6390 4.6432 4.6474 4.6516 4.6558 4.6600 4.6642 4.6684 4.6726 4.6768 4.6810 4.6852 4.6894 4.6936 4.6978 4.7020 4.7062 4.7104 4.7146 4.7188 4.7230 4.7272 4.7314 4.7356 4.7398 4.7440 4.7482 4.7524 4.7566 4.7608 4.7650 4.7692 4.7734 4.7776 4.7818 4.7860 4.7902 4.7944 4.7986 4.8028 4.8070 4.8112 4.8154 4.8196 4.8238 4.8280 4.8322 4.8364 4.8406 4.8448 4.8490 4.8532 4.8574 4.8616 4.8658 4.8700 4.8742 4.8784 4.8826 4.8868 4.8910 4.8952 4.8994 4.9036 4.9078 4.9120 4.9162 4.9204 4.9246 4.9288 4.9330 4.9372 4.9414 4.9456 4.9498 4.9540 4.9582 4.9624 4.9666 4.9708 4.9750 4.9792 4.9834 4.9876 4.9918 4.9960 5.0002 5.0044 5.0086 5.0128 5.0170 5.0212 5.0254 5.0296 5.0338 5.0380 5.0422 5.0464 5.0506 5.0548 5.0590 5.0632 5.0674 5.0716 5.0758 5.0800 5.0842 5.0884 5.0926 5.0968 5.1010 5.1052 5.1094 5.1136 5.1178 5.1220 5.1262 5.1304 5.1346 5.1388 5.1430 5.1472 5.1514 5.1556 5.1598 5.1640 5.1682 5.1724 5.1766 5.1808 5.1850 5.1892 5.1934 5.1976 5.2018 5.2060 5.2102 5.2144 5.2186 5.2228 5.2270 5.2312 5.2354 5.2396 5.2438 5.2480 5.2522 5.2564 5.2606 5.2648 5.2690 5.2732 5.2774 5.2816 5.2858 5.2900 5.2942 5.2984 5.3026 5.3068 5.3110 5.3152 5.3194 5.3236 5.3278 5.3320 5.3362 5.3404 5.3446 5.3488 5.3530 5.3572 5.3614 5.3656 5.3698 5.3740 5.3782 5.3824 5.3866 5.3908 5.3950 5.3992 5.4034 5.4076 5.4118 5.4160 5.4202 5.4244 5.4286 5.4328 5.4370 5.4412 5.4454 5.4496 5.4538 5.4580 5.4622 5.4664 5.4706 5.4748 5.4790 5.4832 5.4874 5.4916 5.4958 5.5000 5.5042 5.5084 5.5126 5.5168 5.5210 5.5252 5.5294 5.5336 5.5378 5.5420 5.5462 5.5504 5.5546 5.5588 5.5630 5.5672 5.5714 5.5756 5.5798 5.5840 5.5882 5.5924 5.5966 5.6008 5.6050 5.6092 5.6134 5.6176 5.6218 5.6260 5.6302 5.6344 5.6386 5.6428 5.6470 5.6512 5.6554 5.6596 5.6638 5.6680 5.6722 5.6764 5.6806 5.6848 5.6890 5.6932 5.6974 5.7016 5.7058 5.7100 5.7142 5.7184 5.7226 5.7268 5.7310 5.7352 5.7394 5.7436 5.7478 5.7520 5.7562 5.7604 5.7646 5.7688 5.7730 5.7772 5.7814 5.7856 5.7898 5.7940 5.7982 5.8024 5.8066 5.8108 5.8150 5.8192 5.8234 5.8276 5.8318 5.8360 5.8402 5.8444 5.8486 5.8528 5.8570 5.8612 5.8654 5.8696 5.8738 5.8780 5.8822 5.8864 5.8906 5.8948 5.8990 5.9032 5.9074 5.9116 5.9158 5.9200 5.9242 5.9284 5.9326 5.9368 5.9410 5.9452 5.9494 5.9536 5.9578 5.9620 5.9662 5.9704 5.9746 5.9788 5.9830 5.9872 5.9914 5.9956 6.0000]
    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 0.3842 0.3885 0.3927 0.3969 0.4011 0.4053 0.4095 0.4137 0.4179 0.4221 0.4263 0.4305 0.4347 0.4389 0.4431 0.4473 0.4515 0.4557 0.4599 0.4641 0.4683 0.4725 0.4767 0.4809 0.4851 0.4893 0.4935 0.4977 0.5019 0.5061 0.5103 0.5145 0.5187 0.5229 0.5271 0.5313 0.5355 0.5397 0.5439 0.5481 0.5523 0.5565 0.5607 0.5649 0.5691 0.5733 0.5775 0.5817 0.5859 0.5901 0.5943 0.5985 0.6027 0.6069 0.6111 0.6153 0.6195 0.6237 0.6279 0.6321 0.6363 0.6405 0.6447 0.6489 0.6531 0.6573 0.6615 0.6657 0.6699 0.6741 0.6783 0.6825 0.6867 0.6909 0.6951 0.6993 0.7035 0.7077 0.7119 0.7161 0.7203 0.7245 0.7287 0.7329 0.7371 0.7413 0.7455 0.7497 0.7539 0.7581 0.7623 0.7665 0.7707 0.7749 0.7791 0.7833 0.7875 0.7917 0.7959 0.8001 0.8043 0.8085 0.8127 0.8169 0.8211 0.8253 0.8295 0.8337 0.8379 0.8421 0.8463 0.8505 0.8547 0.8589 0.8631 0.8673 0.8715 0.8757 0.8799 0.8841 0.8883 0.8925 0.8967 0.9009 0.9051 0.9093 0.9135 0.9177 0.9219 0.9261 0.9303 0.9345 0.9387 0.9429 0.9471 0.9513 0.9555 0.9597 0.9639 0.9681 0.9723 0.9765 0.9807 0.9849 0.9891 0.9933 0.9975 1.0017 1.0059 1.0101 1.0143 1.0185 1.0227 1.0269 1.0311 1.0353 1.0395 1.0437 1.0479 1.0521 1.0563 1.0605 1.0647 1.0689 1.0731 1.0773 1.0815 1.0857 1.0899 1.0941 1.0983 1.1025 1.1067 1.1109 1.1151 1.1193 1.1235 1.1277 1.1319 1.1361 1.1403 1.1445 1.1487 1.1529 1.1571 1.1613 1.1655 1.1697 1.1739 1.1781 1.1823 1.1865 1.1907 1.1949 1.1991 1.2033 1.2075 1.2117 1.2159 1.2201 1.2243 1.2285 1.2327 1.2369 1.2411 1.2453 1.2495 1.2537 1.2579 1.2621 1.2663 1.2705 1.2747 1.2789 1.2831 1.2873 1.2915 1.2957 1.2999 1.3041 1.3083 1.3125 1.3167 1.3209 1.3251 1.3293 1.3335 1.3377 1.3419 1.3461 1.3503 1.3545 1.3587 1.3629 1.3671 1.3713 1.3755 1.3797 1.3839 1.3881 1.3923 1.3965 1.4007 1.4049 1.4091 1.4133 1.4175 1.4217 1.4259 1.4301 1.4343 1.4385 1.4427 1.4469 1.4511 1.4553 1.4595 1.4637 1.4679 1.4721 1.4763 1.4805 1.4847 1.4889 1.4931 1.4973 1.5015 1.5057 1.5099 1.5141 1.5183 1.5225 1.5267 1.5309 1.5351 1.5393 1.5435 1.5477 1.5519 1.5561 1.5603 1.5645 1.5687 1.5729 1.5771 1.5813 1.5855 1.5897 1.5939 1.5981 1.6023 1.6065 1.6107 1.6149 1.6191 1.6233 1.6275 1.6317 1.6359 1.6401 1.6443 1.6485 1.6527 1.6569 1.6611 1.6653 1.6695 1.6737 1.6779 1.6821 1.6863 1.6905 1.6947 1.6989 1.7031 1.7073 
```

```

    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 double]
    stop: 'Training finished: Met validation criterion'
    num_epochs: 16
    trainMask: {[1 1 NaN 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 NaN 1 NaN 1 1]
               [NaN NaN 1 NaN NaN NaN NaN NaN NaN NaN 1 NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN]
               [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]}
    valMask: {[NaN NaN 1 NaN NaN NaN NaN NaN NaN NaN 1 NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN]
              [NaN NaN NaN NaN NaN NaN NaN NaN NaN 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 NaN NaN]}

```

```

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 0.1880 0.1940 0.2000 0.2060 0.2120]
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 2.4900 2.4885 2.4870 2.4855 2.4840]
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 2.3510 2.3528 2.3510 2.3528 2.3510]
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-05 1.0000e-05 1.0000e-04 1.0000e-05 1.0000e-05 1.0000e-05 1.0000e-05 1.0000e-05 1.0000e-05 1.0000e-05 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 0.0075 0.0038 0.0019 0.0009 0.0005 0.0002]
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 = 1x2
559.6753 541.1303
Rmse_val = 1x2
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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN 1 1 1 NaN 1 NaN]}
    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 NaN]}
    testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]}
    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 0.1711 0.1781 0.1851 0.1921 0.1991 0.2061 0.2131 0.2201 0.2271 0.2341 0.2411 0.2481 0.2551 0.2621 0.2691 0.2761 0.2831 0.2901 0.2971 0.3041 0.3111 0.3181 0.3251 0.3321 0.3391 0.3461 0.3531 0.3601 0.3671 0.3741 0.3811 0.3881 0.3951 0.4021 0.4091 0.4161 0.4231 0.4301 0.4371 0.4441 0.4511 0.4581 0.4651 0.4721 0.4791 0.4861 0.4931 0.5001 0.5071 0.5141 0.5211 0.5281 0.5351 0.5421 0.5491 0.5561 0.5631 0.5701 0.5771 0.5841 0.5911 0.5981 0.6051 0.6121 0.6191 0.6261 0.6331 0.6401 0.6471 0.6541 0.6611 0.6681 0.6751 0.6821 0.6891 0.6961 0.7031 0.7101 0.7171 0.7241 0.7311 0.7381 0.7451 0.7521 0.7591 0.7661 0.7731 0.7801 0.7871 0.7941 0.8011 0.8081 0.8151 0.8221 0.8291 0.8361 0.8431 0.8501 0.8571 0.8641 0.8711 0.8781 0.8851 0.8921 0.8991 0.9061 0.9131 0.9201 0.9271 0.9341 0.9411 0.9481 0.9551 0.9621 0.9691 0.9761 0.9831 0.9901 0.9971 1.0041 1.0111 1.0181 1.0251 1.0321 1.0391 1.0461 1.0531 1.0601 1.0671 1.0741 1.0811 1.0881 1.0951 1.1021 1.1091 1.1161 1.1231 1.1301 1.1371 1.1441 1.1511 1.1581 1.1651 1.1721 1.1791 1.1861 1.1931 1.2001 1.2071 1.2141 1.2211 1.2281 1.2351 1.2421 1.2491 1.2561 1.2631 1.2701 1.2771 1.2841 1.2911 1.2981 1.3051 1.3121 1.3191 1.3261 1.3331 1.3401 1.3471 1.3541 1.3611 1.3681 1.3751 1.3821 1.3891 1.3961 1.4031 1.4101 1.4171 1.4241 1.4311 1.4381 1.4451 1.4521 1.4591 1.4661 1.4731 1.4801 1.4871 1.4941 1.5011 1.5081 1.5151 1.5221 1.5291 1.5361 1.5431 1.5501 1.5571 1.5641 1.5711 1.5781 1.5851 1.5921 1.5991 1.6061 1.6131 1.6201 1.6271 1.6341 1.6411 1.6481 1.6551 1.6621 1.6691 1.6761 1.6831 1.6901 1.6971 1.7041 1.7111 1.7181 1.7251 1.7321 1.7391 1.7461 1.7531 1.7601 1.7671 1.7741 1.7811 1.7881 1.7951 1.8021 1.8091 1.8161 1.8231 1.8301 1.8371 1.8441 1.8511 1.8581 1.8651 1.8721 1.8791 1.8861 1.8931 1.9001 1.9071 1.9141 1.9211 1.9281 1.9351 1.9421 1.9491 1.9561 1.9631 1.9701 1.9771 1.9841 1.9911 1.9981 2.0051 2.0121 2.0191 2.0261 2.0331 2.0401 2.0471 2.0541 2.0611 2.0681 2.0751 2.0821 2.0891 2.0961 2.1031 2.1101 2.1171 2.1241 2.1311 2.1381 2.1451 2.1521 2.1591 2.1661 2.1731 2.1801 2.1871 2.1941 2.2011 2.2081 2.2151 2.2221 2.2291 2.2361 2.2431 2.2501 2.2571 2.2641 2.2711 2.2781 2.2851 2.2921 2.2991 2.3061 2.3131 2.3201 2.3271 2.3341 2.3411 2.3481 2.3551 2.3621 2.3691 2.3761 2.3831 2.3901 2.3971 2.4041 2.4111 2.4181 2.4251 2.4321 2.4391 2.4461 2.4531 2.4601 2.4671 2.4741 2.4811 2.4881 2.4951 2.5021 2.5091 2.5161 2.5231 2.5301 2.5371 2.5441 2.5511 2.5581 2.5651 2.5721 2.5791 2.5861 2.5931 2.6001 2.6071 2.6141 2.6211 2.6281 2.6351 2.6421 2.6491 2.6561 2.6631 2.6701 2.6771 2.6841 2.6911 2.6981 2.7051 2.7121 2.7191 2.7261 2.7331 2.7401 2.7471 2.7541 2.7611 2.7681 2.7751 2.7821 2.7891 2.7961 2.8031 2.8101 2.8171 2.8241 2.8311 2.8381 2.8451 2.8521 2.8591 2.8661 2.8731 2.8801 2.8871 2.8941 2.9011 2.9081 2.9151 2.9221 2.9291 2.9361 2.9431 2.9501 2.9571 2.9641 2.9711 2.9781 2.9851 2.9921 2.9991 3.0061 3.0131 3.0201 3.0271 3.0341 3.0411 3.0481 3.0551 3.0621 3.0691 3.0761 3.0831 3.0901 3.0971 3.1041 3.1111 3.1181 3.1251 3.1321 3.1391 3.1461 3.1531 3.1601 3.1671 3.1741 3.1811 3.1881 3.1951 3.2021 3.2091 3.2161 3.2231 3.2301 3.2371 3.2441 3.2511 3.2581 3.2651 3.2721 3.2791 3.2861 3.2931 3.3001 3.3071 3.3141 3.3211 3.3281 3.3351 3.3421 3.3491 3.3561 3.3631 3.3701 3.3771 3.3841 3.3911 3.3981 3.4051 3.4121 3.4191 3.4261 3.4331 3.4401 3.4471 3.4541 3.4611 3.4681 3.4751 3.4821 3.4891 3.4961 3.5031 3.5101 3.5171 3.5241 3.5311 3.5381 3.5451 3.5521 3.5591 3.5661 3.5731 3.5801 3.5871 3.5941 3.6011 3.6081 3.6151 3.6221 3.6291 3.6361 3.6431 3.6501 3.6571 3.6641 3.6711 3.6781 3.6851 3.6921 3.6991 3.7061 3.7131 3.7201 3.7271 3.7341 3.7411 3.7481 3.7551 3.7621 3.7691 3.7761 3.7831 3.7901 3.7971 3.8041 3.8111 3.8181 3.8251 3.8321 3.8391 3.8461 3.8531 3.8601 3.8671 3.8741 3.8811 3.8881 3.8951 3.9021 3.9091 3.9161 3.9231 3.9301 3.9371 3.9441 3.9511 3.9581 3.9651 3.9721 3.9791 3.9861 3.9931 4.0001 4.0071 4.0141 4.0211 4.0281 4.0351 4.0421 4.0491 4.0561 4.0631 4.0701 4.0771 4.0841 4.0911 4.0981 4.1051 4.1121 4.1191 4.1261 4.1331 4.1401 4.1471 4.1541 4.1611 4.1681 4.1751 4.1821 4.1891 4.1961 4.2031 4.2101 4.2171 4.2241 4.2311 4.2381 4.2451 4.2521 4.2591 4.2661 4.2731 4.2801 4.2871 4.2941 4.3011 4.3081 4.3151 4.3221 4.3291 4.3361 4.3431 4.3501 4.3571 4.3641 4.3711 4.3781 4.3851 4.3921 4.3991 4.4061 4.4131 4.4201 4.4271 4.4341 4.4411 4.4481 4.4551 4.4621 4.4691 4.4761 4.4831 4.4901 4.4971 4.5041 4.5111 4.5181 4.5251 4.5321 4.5391 4.5461 4.5531 4.5601 4.5671 4.5741 4.5811 4.5881 4.5951 4.6021 4.6091 4.6161 4.6231 4.6301 4.6371 4.6441 4.6511 4.6581 4.6651 4.6721 4.6791 4.6861 4.6931 4.7001 4.7071 4.7141 4.7211 4.7281 4.7351 4.7421 4.7491 4.7561 4.7631 4.7701 4.7771 4.7841 4.7911 4.7981 4.8051 4.8121 4.8191 4.8261 4.8331 4.8401 4.8471 4.8541 4.8611 4.8681 4.8751 4.8821 4.8891 4.8961 4.9031 4.9101 4.9171 4.9241 4.9311 4.9381 4.9451 4.9521 4.9591 4.9661 4.9731 4.9801 4.9871 4.9941 5.0011 5.0081 5.0151 5.0221 5.0291 5.0361 5.0431 5.0501 5.0571 5.0641 5.0711 5.0781 5.0851 5.0921 5.0991 5.1061 5.1131 5.1201 5.1271 5.1341 5.1411 5.1481 5.1551 5.1621 5.1691 5.1761 5.1831 5.1901 5.1971 5.2041 5.2111 5.2181 5.2251 5.2321 5.2391 5.2461 5.2531 5.2601 5.2671 5.2741 5.2811 5.2881 5.2951 5.3021 5.3091 5.3161 5.3231 5.3301 5.3371 5.3441 5.3511 5.3581 5.3651 5.3721 5.3791 5.3861 5.3931 5.4001 5.4071 5.4141 5.4211 5.4281 5.4351 5.4421 5.4491 5.4561 5.4631 5.4701 5.4771 5.4841 5.4911 5.4981 5.5051 5.5121 5.5191 5.5261 5.5331 5.5401 5.5471 5.5541 5.5611 5.5681 5.5751 5.5821 5.5891 5.5961 5.6031 5.6101 5.6171 5.6241 5.6311 5.6381 5.6451 5.6521 5.6591 5.6661 5.6731 5.6801 5.6871 5.6941 5.7011 5.7081 5.7151 5.7221 5.7291 5.7361 5.7431 5.7501 5.7571 5.7641 5.7711 5.7781 5.7851 5.7921 5.7991 5.8061 5.8131 5.8201 5.8271 5.8341 5.8411 5.8481 5.8551 5.8621 5.8691 5.8761 5.8831 5.8901 5.8971 5.9041 5.9111 5.9181 5.9251 5.9321 5.9391 5.9461 5.9531 5.9601 5.9671 5.9741 5.9811 5.9881 5.9951 6.0021 6.0091 6.0161 6.0231 6.0301 6.0371 6.0441 6.0511 6.0581 6.0651 6.0721 6.0791 6.0861 6.0931 6.1001 6.1071 6.1141 6.1211 6.1281 6.1351 6.1421 6.1491 6.1561 6.1631 6.1701 6.1771 6.1841 6.1911 6.1981 6.2051 6.2121 6.2191 6.2261 6.2331 6.2401 6.2471 6.2541 6.2611 6.2681 6.2751 6.2821 6.2891 6.2961 6.3031 6.3101 6.3171 6.3241 6.3311 6.3381 6.3451 6.3521 6.3591 6.3661 6.3731 6.3801 6.3871 6.3941 6.4011 6.4081 6.4151 6.4221 6.4291 6.4361 6.4431 6.4501 6.4571 6.4641 6.4711 6.4781 6.4851 6.4921 6.4991 6.5061 6.5131 6.5201 6.5271 6.5341 6.5411 6.5481 6.5551 6.5621 6.5691 6.5761 6.5831 6.5901 6.5971 6.6041 6.6111 6.6181 6.6251 6.6321 6.6391 6.6461 6.6531 6.6601 6.6671 6.6741 6.6811 6.6881 6.6951 6.7021 6.7091 6.7161 6.7231 6.7301 6.7371 6.7441 6.7511 6.7581 6.7651 6.7721 6.7791 6.7861 6.7931 6.8001 6.8071 6.8141 6.8211 6.8281 6.8351 6.8421 6.8491 6.8561 6.8631 6.8701 6.8771 6.8841 6.8911 6.8981 6.9051 6.9121 6.9191 6.9261 6.9331 6.9401 6.9471 6.9541 6.9611 6.9681 6.9751 6.9821 6.9891 6.9961 7.0031 7.0101 7.0171 7.0241 7.0311 7.0381 7.0451 7.0521 7.0591 7.0661 7.0731 7.0801 7.0871 7.0941 7.1011 7.1081 7.1151 7.1221 7.1291 7.1361 7.1431 7.1501 7.1571 7.1641 7.1711 7.1781 7.1851 7.1921 7.1991 7.2061 7.2131 7.2201 7.2271 7.2341 7.2411 7.2481 7.2551 7.2621 7.2691 7.2761 7.2831 7.2901 7.2971 7.3041 7.3111 7.3181 7.3251 7.3321 7.3391 7.3461 7.3531 7.3601 7.3671 7.3741 7.3811 7.3881 7.3951 7.4021 7.4091 7.4161 7.4231 7.4301 7.4371 7.4441 7.4511 7.4581 7.4651 7.4721 7.4791 7.4861 7.4931 7.5001 7.5071 7.5141 7.5211 7.5281 7.5351 7.5421 7.5491 7.5561 7.5631 7.5701 7.5771 7.5841 7.5911 7.5981 7.6051 7.6121 7.6191 7.6261 7.6331 7.6401 7.6471 7.6541 7.6611 7.6681 7.6751 7.6821 7.6891 7.6961 7.7031 7.7101 7.7171 7.7241 7.7311 7.7381 7.7451 7.7521 7.7591 7.7661 7.7731 7.7801 7.7871 7.7941 7.8011 7.8081 7.8151 7.8221 7.8291 7.8361 7.8431 7.8501 7.8571 7.8641 7.8711 7.8781 7.8851 7.8921 7.8991 7.9061 7.9131 7.9201 7.9271 7.9341 7.9411 7.9481 7.9551 7.9621 7.9691 7.9761 7.9831 7.9901 7.9971 8.0041 8.0111 8.0181 8.0251 8.0321 8.0391 8.0461 8.0531 8.0601 8.0671 8.0741 8.0811 8.0881 8.0951 8.1021 8.1091 8.1161 8.1231 8.1301 8.1371 8.1441 8.1511 8.1581 8.1651 8.1721 8.1791 8.1861 8.1931 8.2001 8.2071 8.2141 8.2211 8.2281 8.2351 8.2421 8.2491 8.2561 8.2631 8.2701 8.2771 8.2841 8.2911 8.2981 8.3051 8.3121 8.3191 8.3261 8.3331 8.3401 8.3471 8.3541 8.3611 8.3681 8.3751 8.3821 8.3891 8.3961 8.4031 8.4101 8.4171 8.4241 8.4311 8.4381 8.4451 8.4521 8.4591 8.4661 8.4731 8.4801 8.4871 8.4941 8.5011 8.5081 8.5151 8.5221 8.5291 8.5361 8.5431 8.5501 8.5571 8.5641 8.5711 8.5781 8.5851 8.5921 8.5991 8.6061 8.6131 8.6201 8.6271 8.6341 8.6411 8.6481 8.6551 8.6621 8.6691 8.6761 8.6831 8.6901 8.6971 8.7041 8.7111 8.7181 8.7251 8.7321 8.7391 8.7461 8.7531 8.7601 8.7671 8.7741 8.7811 8.7881 8.7951 8.8021 8.8091 8.8161 8.8231 8.8301 8.8371 8.8441 8.8511 8.8581 8.8651 8.8721 8.8791 8.8861 8.8931 8.9001 8.9071 8.9141 8.9211 8.9281 8.9351 8.9421 8.9491 8.9561 8.9631 8.9701 8.9771 8.9841 8.9911 8.9981 9.0051 9.0121 9.0191 9.0261 9.0331 9.0401 9.0471 9.0541 9.0611 9.0681 9.0751 9.0821 9.0891 9.0961 9.1031 9.1101 9.1171 9.1241 9.1311 9.1381 9.1451 9.1521 9.1591 9.1661 9.1731 9.1801 9.1871 9.1941 9.2011 9.2081 9.2151 9.2221 9.2291 9.2361 9.2431 9.2501 9.2571 9.2641 9.2711 9.2781 9.2851 9.2921 9.2991 9.3061 9.3131 9.3201 9.3271 9.3341 9.3411 9.3481 9.3551 9.3621 9.3691 9.3761 9.3831 9.3901 9.3971 9.4041 9.4111 9.4181 9.4251 9.4321 9.4391 9.4461 9.4531 9.4601 9.4671 9.4741 9.4811 9.4881 9.4951 9.5021 9.5091 9.5161 9.5231 9.5301 9.5371 9.5441 9.5511 9.5581 9.5651 9.5721 9.5791 9.5861 9.5931 9.6001 9.6071 9.6141 9.6211 9.6281 9.6351 9.6421 9.6491 9.6561 9.6631 9.6701 9.6771 9.6841 9.6911 9.6981 9.7051 9.7121 9.7191 9.7261 9.7331 9.7401 9.7471 9.7541 9.7611 9.7681 9.7751 9.7821 9.7891 9.7961 9.8031 9.8101 9.8171 9.8241 9.8311 9.8381 9.8451 9.8521 9.8591 9.8661 9.8731 9.8801 9.8871 9.8941 9.9011 9.9081 9.9151 9.9221 9.9291 9.9361 9.9431 9.9501 9.9571 9.9641 9.9711 9.9781 9.9851 9.9921 10.0001 10.0071 10.0141 10.0211 10.0281 10.0351 10.0421 10.0491 10.0561 10.0631 10.0701 10.0771 10.0841 10.0911 10.0981 10.1051 10.1121 10.1191 10.1261 10.1331 10.1401 10.1471 10.1541 10.1611 10.1681 10.1751 10.1821 10.1891 10.1961 10.2031 10.2101 10.2171 10.2241 10.2311 10.2381 10.2451 10.2521 10.2591 10.2661 10.2731 10.2801 10.2871 10.2941 10.3011 10.3081 10.3151 10.3221 10.3291 10.3361 10.3431 10.3501 10.3571 10.3641 10.3711 10.3781 10.3851 10.3921 10.3991 10.4061 10.4131 10.4201 10.4271 10.4341 10.4411 10.4481 10.4551 10.4621 10.4691 10.4761 10.4831 10.4901 10.4971 10.5041 10.5111 10.5181 10.5251 10.5321 10.5391 10.5461 10.5531 10.5601 10.5671 10.5741 10.5811 10.5881 10.5951 10.6021 10.6091 10.6161 10.6231 10.6301 10.6371 10.6441 10.6511 10.6581 10.6651 10.6721 10.6791 10.6861 10.6931 10.7001 10.7071 10.7141 10.7211 10.7281 10.7351 10.7421 10.7491 10.7561 10.7631 10.7701 10.7771 10.7841 10.7911 10.7981 10.8051 10.8121 10.8191 10.8261 10.8331 10.8401 10.8471 10.8541 10.8611 10.8681 10.8751 10.8821 10.8891 10.8961 10.9031 10.9101 10.9171 10.9241 10.9311 10.9381 10.9451 10.9521 10.9591 10.9661 10.9731 10.9801 10.9871 10.9941 11.0011 11.0081 11.0151 11.0221 11.0291 11.0361 11.0431 11.0501 11.0571 11.0641 11.0711 11.0781 11.0851 11.0921 11.0991 11.1061 11.1131 11.1201 11.1271 11.1341 11.1411 11.1481 11.1551 11.1621 11.1691 11.1761 11.1831 11.1901 11.1971 11.2041 11.2111 11.2181 11.2251 11.2321 11.2391 11.2461 11.2531 11.2601 11.2671 11.2741 11.2811 11.2881 11.2951 11.3021 11.3091 11.3161 11.3231 11.3301 11.3371 11.3441 11.3511 11.3581 11.3651 11.3721 11.3791 11.3861 11.3931 11.4001 11.4071 11.4141 11.4211 11.4281 11.4
```

```

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:

```

```

tr = struct with fields:
    trainFcn: 'trainlm'
    trainParam: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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 96 109 110 111]
    testInd: [1x0 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 NaN 1 NaN NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN 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 NaN NaN NaN NaN NaN NaN NaN]
    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 0.3704 0.3788 0.3872 0.3956 0.4040 0.4124 0.4208 0.4292 0.4376 0.4460 0.4544 0.4628 0.4712 0.4796 0.4880 0.4964 0.5048 0.5132 0.5216 0.5300 0.5384 0.5468 0.5552 0.5636 0.5720 0.5804 0.5888 0.5972 0.6056 0.6140 0.6224 0.6308 0.6392 0.6476 0.6560 0.6644 0.6728 0.6812 0.6896 0.6980 0.7064 0.7148 0.7232 0.7316 0.7400 0.7484 0.7568 0.7652 0.7736 0.7820 0.7904 0.7988 0.8072 0.8156 0.8240 0.8324 0.8408 0.8492 0.8576 0.8660 0.8744 0.8828 0.8912 0.8996 0.9080 0.9164 0.9248 0.9332 0.9416 0.9500 0.9584 0.9668 0.9752 0.9836 0.9920 1.0004 1.0088 1.0172 1.0256 1.0340 1.0424 1.0508 1.0592 1.0676 1.0760 1.0844 1.0928 1.1012 1.1096 1.1180 1.1264 1.1348 1.1432 1.1516 1.1600 1.1684 1.1768 1.1852 1.1936 1.2020 1.2104 1.2188 1.2272 1.2356 1.2440 1.2524 1.2608 1.2692 1.2776 1.2860 1.2944 1.3028 1.3112 1.3196 1.3280 1.3364 1.3448 1.3532 1.3616 1.3700 1.3784 1.3868 1.3952 1.4036 1.4120 1.4204 1.4288 1.4372 1.4456 1.4540 1.4624 1.4708 1.4792 1.4876 1.4960 1.5044 1.5128 1.5212 1.5296 1.5380 1.5464 1.5548 1.5632 1.5716 1.5800 1.5884 1.5968 1.6052 1.6136 1.6220 1.6304 1.6388 1.6472 1.6556 1.6640 1.6724 1.6808 1.6892 1.6976 1.7060 1.7144 1.7228 1.7312 1.7396 1.7480 1.7564 1.7648 1.7732 1.7816 1.7900 1.7984 1.8068 1.8152 1.8236 1.8320 1.8404 1.8488 1.8572 1.8656 1.8740 1.8824 1.8908 1.8992 1.9076 1.9160 1.9244 1.9328 1.9412 1.9496 1.9580 1.9664 1.9748 1.9832 1.9916 2.0000 2.0084 2.0168 2.0252 2.0336 2.0420 2.0504 2.0588 2.0672 2.0756 2.0840 2.0924 2.1008 2.1092 2.1176 2.1260 2.1344 2.1428 2.1512 2.1596 2.1680 2.1764 2.1848 2.1932 2.2016 2.2100 2.2184 2.2268 2.2352 2.2436 2.2520 2.2604 2.2688 2.2772 2.2856 2.2940 2.3024 2.3108 2.3192 2.3276 2.3360 2.3444 2.3528 2.3612 2.3696 2.3780 2.3864 2.3948 2.4032 2.4116 2.4200 2.4284 2.4368 2.4452 2.4536 2.4620 2.4704 2.4788 2.4872 2.4956 2.5040 2.5124 2.5208 2.5292 2.5376 2.5460 2.5544 2.5628 2.5712 2.5796 2.5880 2.5964 2.6048 2.6132 2.6216 2.6300 2.6384 2.6468 2.6552 2.6636 2.6720 2.6804 2.6888 2.6972 2.7056 2.7140 2.7224 2.7308 2.7392 2.7476 2.7560 2.7644 2.7728 2.7812 2.7896 2.7980 2.8064 2.8148 2.8232 2.8316 2.8400 2.8484 2.8568 2.8652 2.8736 2.8820 2.8904 2.8988 2.9072 2.9156 2.9240 2.9324 2.9408 2.9492 2.9576 2.9660 2.9744 2.9828 2.9912 2.9996 3.0080 3.0164 3.0248 3.0332 3.0416 3.0500 3.0584 3.0668 3.0752 3.0836 3.0920 3.1004 3.1088 3.1172 3.1256 3.1340 3.1424 3.1508 3.1592 3.1676 3.1760 3.1844 3.1928 3.2012 3.2096 3.2180 3.2264 3.2348 3.2432 3.2516 3.2600 3.2684 3.2768 3.2852 3.2936 3.3020 3.3104 3.3188 3.3272 3.3356 3.3440 3.3524 3.3608 3.3692 3.3776 3.3860 3.3944 3.4028 3.4112 3.4196 3.4280 3.4364 3.4448 3.4532 3.4616 3.4700 3.4784 3.4868 3.4952 3.5036 3.5120 3.5204 3.5288 3.5372 3.5456 3.5540 3.5624 3.5708 3.5792 3.5876 3.5960 3.6044 3.6128 3.6212 3.6296 3.6380 3.6464 3.6548 3.6632 3.6716 3.6800 3.6884 3.6968 3.7052 3.7136 3.7220 3.7304 3.7388 3.7472 3.7556 3.7640 3.7724 3.7808 3.7892 3.7976 3.8060 3.8144 3.8228 3.8312 3.8396 3.8480 3.8564 3.8648 3.8732 3.8816 3.8900 3.8984 3.9068 3.9152 3.9236 3.9320 3.9404 3.9488 3.9572 3.9656 3.9740 3.9824 3.9908 3.9992 4.0076 4.0160 4.0244 4.0328 4.0412 4.0496 4.0580 4.0664 4.0748 4.0832 4.0916 4.1000 4.1084 4.1168 4.1252 4.1336 4.1420 4.1504 4.1588 4.1672 4.1756 4.1840 4.1924 4.2008 4.2092 4.2176 4.2260 4.2344 4.2428 4.2512 4.2596 4.2680 4.2764 4.2848 4.2932 4.3016 4.3100 4.3184 4.3268 4.3352 4.3436 4.3520 4.3604 4.3688 4.3772 4.3856 4.3940 4.4024 4.4108 4.4192 4.4276 4.4360 4.4444 4.4528 4.4612 4.4696 4.4780 4.4864 4.4948 4.5032 4.5116 4.5200 4.5284 4
```

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

```

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

```

15

```

    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 struct]

```



```

        best_perf: 2.3221
        best_vperf: 2.3635
        best_tperf: NaN
Rmse_train = 1x6
    559.6753    541.1303    539.4591    460.8964    417.2784    509.5563
Rmse_val = 1x6
    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: [1x1 struct]
  performFcn: 'mse'
  performParam: [1x1 struct]
  derivFcn: 'defaultderiv'
  divideFcn: 'dividerand'
  divideMode: 'sample'
  divideParam: [1x1 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: [1x0 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 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 NaN
  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 0 1 0 0 0 1 2 3 0 0 1 0 1 0 0 0 0 1 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 = 1x7
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847
  Rmse_val = 1x7
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 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 1 1 1 NaN
    testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
    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
    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 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 = 1x8
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845
    Rmse_val = 1x8
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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]
               [NaN NaN 1 1 NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN 1 NaN]
               [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]}
    valMask: {[NaN NaN 1 1 NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN 1 NaN]
              [NaN NaN NaN NaN NaN NaN NaN NaN 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 NaN]}
    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]
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 = 1x9
559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x9
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: [1x1 struct]
    performFcn: 'mse'
performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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 98 100]
    testInd: [1x0 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 NaN
    testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
    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
    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 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 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 = 1x10
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x10
    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

```



```
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: [1x1 struct]  
    performFcn: 'mse'  
    performParam: [1x1 struct]  
    derivFcn: 'defaultderiv'  
    divideFcn: 'dividerand'  
    divideMode: 'sample'  
    divideParam: [1x1 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: [1x0 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 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 NaN
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]
mu: [1.0000e-03 1.0000e-04 1.0000e-04 1.0000e-03 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 = 1x12
559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x12
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'

```



```

559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ...
Rmse_val = 1x13
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

```



```

    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'

```

```

    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 1 NaN 1 1 1 1 1 NaN
        valMask: {[1 1 NaN NaN NaN NaN NaN NaN NaN NaN 1 1 1 NaN 1 NaN NaN NaN NaN NaN 1 1 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 NaN NaN NaN NaN NaN
    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 1 2 0 0 0 1 2 3 4 5 6]
    best_perf: 2.1878
    best_vperf: 2.2609
    best_tperf: NaN
Rmse_train = 1x15
    559.6753    541.1303    539.4591    460.8964    417.2784    509.5563    368.4847    400.2845 ...
Rmse_val = 1x15
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 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
    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 NaN]
```



```

        .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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 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 NaN
    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 1 0 0 0 1 0 1 2 3 4 5 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 = 1x17
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x17
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN
    testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
    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 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 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 = 1x18
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x18
    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]

subjects:

```

```

    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN NaN]

```



```

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 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 0 1 2 3 0 1 2 3 4 5 6]
best_perf: 1.9809
best_vperf: 2.4324
best_tperf: NaN
Rmse_train = 1x19
559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x19
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: [1x1 struct]
performFcn: 'mse'
performParam: [1x1 struct]
derivFcn: 'defaultderiv'
divideFcn: 'dividerand'
divideMode: 'sample'
divideParam: [1x1 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: [1x0 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
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 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 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 1 2 3 4 5 6]
best_perf: 2.1804
best_vperf: 2.2258
best_tperf: NaN
Rmse_train = 1x20
559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x20
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: [1x1 struct]
performFcn: 'mse'
performParam: [1x1 struct]
derivFcn: 'defaultderiv'
divideFcn: 'dividerand'
divideMode: 'sample'
divideParam: [1x1 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 49]
    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: [1x0 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 NaN]}
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 0.4784 0.5108 0.5432 0.5756]
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 2.0272 2.0088 1.9904 1.9720]
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 2.4820 2.4970 2.5120 2.5270]
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-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 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 1.0000 0.8125 0.6250 0.4375 0.2500]
val_fail: [0 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 = 1x21
    559.6753   541.1303   539.4591   460.8964   417.2784   509.5563   368.4847   400.2845 ...
Rmse_val = 1x21
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 struct]

```



```

        best_perf: 2.1966
        best_vperf: 2.5981
        best_tperf: NaN
Rmse_train = 1x23
    559.6753  541.1303  539.4591  460.8964  417.2784  509.5563  368.4847  400.2845 ...
Rmse_val = 1x23
    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: [1x1 struct]
  performFcn: 'mse'
  performParam: [1x1 struct]
  derivFcn: 'defaultderiv'
  divideFcn: 'dividerand'
  divideMode: 'sample'
  divideParam: [1x1 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: [1x0 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 NaN
    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 NaN
    testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
  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 = 1x24

559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...

Rmse_val = 1x24

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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 double]
    stop: 'Training finished: Met validation criterion'
    num_epochs: 15
    trainMask: {[NaN 1 1 NaN NaN 1 1 1 1 1 NaN NaN 1 1 1 NaN 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 NaN]}
    testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]}
    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 0.5181 0.5511 0.5841]
    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 2.1207 2.1000 2.0792]
    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 2.2836 2.2883 2.2930]
    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-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 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 0.6700 0.5833 0.5000 0.4167]
    val_fail: [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 = 1x25
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
    Rmse_val = 1x25
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN NaN 1 1 1 1 NaN NaN]
               [NaN NaN 1 NaN NaN 1 1 NaN 1 NaN 1 1 1 NaN 1 NaN 1 NaN NaN NaN NaN NaN NaN NaN 1 1 NaN NaN]
               [NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]}
    valMask: {[NaN NaN 1 NaN NaN 1 1 NaN 1 NaN 1 1 1 NaN 1 NaN 1 NaN NaN NaN NaN NaN NaN NaN NaN 1 1 NaN NaN]
              [NaN NaN NaN NaN NaN NaN NaN NaN NaN 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 NaN NaN]}
    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]
        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 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 1 2 3 4 5 6]
        best_perf: 2.0842
        best_vperf: 2.3230
        best_tperf: NaN
Rmse_train = 1x26
    559.6753    541.1303    539.4591    460.8964    417.2784    509.5563    368.4847    400.2845 ...
Rmse_val = 1x26
    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: [1x1 struct]
    performFcn: 'mse'
performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 double]
    stop: 'Training finished: Met validation criterion'
num_epochs: 25
    trainMask: {[1 1 1 1 1 1 1 1 1 1 1 1 1 NaN 1 NaN 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 NaN 1 NaN 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 NaN NaN]
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 1 0 1 0 1 0 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 = 1x27
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x27
    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

```

[illegible]


```
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: [1x1 struct]  
    performFcn: 'mse'  
    performParam: [1x1 struct]  
    derivFcn: 'defaultderiv'  
    divideFcn: 'dividerand'  
    divideMode: 'sample'  
    divideParam: [1x1 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: [1x0 double]  
    stop: 'Training finished: Met validation criterion'
```



```

    divideFcn: 'dividerand'
    divideParam: .trainRatio, .valRatio, .testRatio
    divideMode: 'sample'
        initFcn: 'initlay'
    performFcn: 'mse'
performParam: .regularization, .normalization
    plotFcns: {'plotperform', 'plottrainstate', 'plotterrhist',
               '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: [1x1 struct]
performFcn: 'mse'
performParam: [1x1 struct]
derivFcn: 'defaultderiv'
divideFcn: 'dividerand'
divideMode: 'sample'
divideParam: [1x1 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: [1x0 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 NaN NaN 1 1 1 1 NaN 1 1 1 NaN]}
valMask: {[1 1 NaN NaN NaN NaN 1 NaN 1 NaN NaN NaN NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN NaN 1 1 NaN]}
testMask: {[NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN]}
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 0.7594 0.8025]
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 1.9801 1.9628]
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 2.4151 2.4074]
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-04 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 1.0000e-03 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 0.3928 0.2485 0.1607]
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
e train = 1x30

```

```

559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x30
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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN 1 NaN NaN 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
    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 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 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 = 1x31
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x31
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'

```

```

    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 double]
        stop: 'Training finished: Met validation criterion'
    num_epochs: 14
    trainMask: {[1 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 NaN NaN 1 1 NaN 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 NaN NaN NaN NaN]
    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 0.6899 0.7362 0.7825]
    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 1.9556 1.9313 1.9070]
    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 2.1033 2.1122 2.1211]
    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 1.0000e-03 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 0.7825 0.9361 1.0897 1.2433]
    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 = 1x32
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x32
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN 1 1 NaN]
    valMask: {[NaN 1 NaN 1 1 NaN 1 NaN NaN 1 NaN NaN NaN 1 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]
    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]
```



```

        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 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 = 1x33
    559.6753    541.1303    539.4591    460.8964    417.2784    509.5563    368.4847    400.2845 ...
Rmse_val = 1x33
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 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 NaN]
    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]
    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 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 1 0 1 2 3 4 5 6]
    best_perf: 2.0627
    best_vperf: 2.1282
    best_tperf: NaN
Rmse_train = 1x34
    559.6753    541.1303    539.4591    460.8964    417.2784    509.5563    368.4847    400.2845 ...
Rmse_val = 1x34
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 double]
    stop: 'Training finished: Met validation criterion'
    num_epochs: 18
    trainMask: {[1 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 NaN 1 NaN 1 NaN 1 1 NaN 1 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 NaN NaN NaN
    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 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 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 1 0 0 0 1 2 3 4 5 6]
    best_perf: 1.8862
    best_vperf: 2.4078
    best_tperf: NaN
Rmse_train = 1x35
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x35
    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]

```

subjects:

```

    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN 1 NaN 1 NaN NaN NaN 1 NaN 1 NaN 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 NaN NaN

```

```

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 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 = 1x36
    559.6753    541.1303    539.4591    460.8964    417.2784    509.5563    368.4847    400.2845 ...
Rmse_val = 1x36
    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: [1x1 struct]
performFcn: 'mse'
performParam: [1x1 struct]
derivFcn: 'defaultderiv'
divideFcn: 'dividerand'
divideMode: 'sample'
divideParam: [1x1 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: [1x0 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 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 NaN 1 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 NaN NaN
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 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 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 = 1x37
559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x37
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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 double]
    stop: 'Training finished: Met validation criterion'
    num_epochs: 11
    trainMask: {[NaN 1 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 NaN 1 1 1 NaN NaN NaN NaN NaN NaN 1 1 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 NaN NaN NaN]}
    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 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 = 1x38
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x38
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 NaN 1 NaN NaN NaN 1 NaN NaN NaN NaN NaN 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 NaN NaN
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 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 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 = 1x39
559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
Rmse_val = 1x39
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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 1 1 NaN 1 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
    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]
    mu: [1.0000e-03 1.0000e-04 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 1 0 0 0 0 1 2 3 4 5 6]

```

```

        best_perf: 1.9931
        best_vperf: 2.5199
        best_tperf: NaN
Rmse_train = 1x40
    559.6753   541.1303   539.4591   460.8964   417.2784   509.5563   368.4847   400.2845 ...
Rmse_val = 1x40
    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: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
    derivFcn: 'defaultderiv'
    divideFcn: 'dividerand'
    divideMode: 'sample'
    divideParam: [1x1 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: [1x0 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 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]
    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 1.8]
    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 2.4]
    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 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 = 1x41
    559.6753 541.1303 539.4591 460.8964 417.2784 509.5563 368.4847 400.2845 ...
    Rmse_val = 1x41
    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 Malaysian dataset')

```