```
help newff
 NEWFF Create a feed-forward backpropagation network.

  Syntax

    net = newff
    net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)

  Description

   NET = NEWFF creates a new network with a dialog box.

    NEWFF(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF) takes,
      PR  - Rx2 matrix of min and max values for R input elements.
      Si  - Size of ith layer, for Nl layers.
      TFi - Transfer function of ith layer, default = 'tansig'.
      BTF - Backprop network training function, default = 'trainlm'.
      BLF - Backprop weight/bias learning function, default = 'learngdm'.
      PF  - Performance function, default = 'mse'.
    and returns an N layer feed-forward backprop network.

    The transfer functions TFi can be any differentiable transfer
    function such as TANSIG, LOGSIG, or PURELIN.

    The training function BTF can be any of the backprop training
    functions such as TRAINLM, TRAINBFG, TRAINRP, TRAINGD, etc.

    *WARNING*: TRAINLM is the default training function because it
    is very fast, but it requires a lot of memory to run.  If you get
    an "out-of-memory" error when training try doing one of these:
```

(1) Slow TRAINLM training, but reduce memory requirements, by
   setting NET.trainParam.mem_reduc to 2 or more. (See HELP TRAINLM.)
(2) Use TRAINBFG, which is slower but more memory efficient than TRAINLM.
(3) Use TRAINRP which is slower but more memory efficient than TRAINBFG.

The learning function BLF can be either of the backpropagation learning functions such as LEARNGD, or LEARNGDM.

The performance function can be any of the differentiable performance functions such as MSE or MSEREG.

Examples

Here is a problem consisting of inputs P and targets T that we would like to solve with a network.

```
P = [0 1 2 3 4 5 6 7 8 9 10];
T = [0 1 2 3 4 3 2 1 2 3 4];
```

Here a two-layer feed-forward network is created.  The network's input ranges from [0 to 10].  The first layer has five TANSIG neurons, the second layer has one PURELIN neuron.  The TRAINLM network training function is to be used.

```
net = newff([0 10],[5 1],{'tansig' 'purelin'});
```

Here the network is simulated and its output plotted against the targets.

```
Y = sim(net,P);
 plot(P,T,P,Y,'o')
```

Here the network is trained for 50 epochs. Again the network's
output is plotted.

```
net.trainParam.epochs = 50;
net = train(net,P,T);
Y = sim(net,P);
 plot(P,T,P,Y,'o')
```

Algorithm

Feed-forward networks consist of Nl layers using the DOTPROD
weight function, NETSUM net input function, and the specified
transfer functions.

The first layer has weights coming from the input. Each subsequent
layer has a weight coming from the previous layer. All layers
have biases. The last layer is the network output.

Each layer's weights and biases are initialized with INITNW.

Adaption is done with TRAINS which updates weights with the
specified learning function. Training is done with the specified
training function. Performance is measured according to the specified
performance function.

See also newcf, newelm, sim, init, adapt, train, trains

Reference page in Help browser

```
doc newff
```

For example, the following command creates a two-layer network. There is one input vector with two elements. The values for the first element of the input vector range between -1 and 2, the values of the second element of the input vector range between 0 and 5. There are three neurons in the first layer and one neuron in the second (output) layer. The transfer function in the first layer is tan-sigmoid, and the output layer transfer function is linear. The training function is traingd (which is described in a later section). net=newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');

This command creates the network object and also initializes the weights and biases of the network; therefore the network is ready for training. There are times when you may want to reinitialize the weights, or to perform a custom initialization. The next section explains the details of the initialization process. Initializing Weights (init).   Before training a feedforward network, the weights and biases must be initialized. The newff command will automatically initialize the weights, but you may want to reinitialize them. This can be done with the command init. This function takes a network object as input and returns a network object with all weights and biases initialized. Here is how a network is initialized (or reinitialized): net = init(net);

Simulation (sim)The function sim simulates a network. sim takes the network input p, and the network object net, and returns the network outputs a. Here is how you can use sim to simulate the network we created above for a single input vector: p = [1;2];

```
a = sim(net,p)

a =

   -0.1011
```

(If you try these commands, your output may be different, depending on the state of your random number generator when the network was initialized.) Below, sim is called to calculate the outputs for a concurrent set of three input vectors. This is the batch mode form of simulation, in which all of the input vectors are place in one matrix. This is much more efficient than presenting the vectors one at a time. p = [1 3 2;2 4 1];

```
a=sim(net,p)

a =

  -0.1011   -0.2308    0.4955
```