Neural Networks in Matlab

Matlab has a suite of programs designed to build neural networks (the Neural Networks Toolbox). Additionally, there are demonstrations available through Matlab's help feature.

In this lab, we will only work with three layer "feed forward" nets (these are the nets we discussed in class).

There are three steps to using neural networks: Design, Training, and Testing (the same for modeling in general!).

- **Design:**
  - Define the number of nodes in each of the three layers (input, hidden, output).
  - Define what σ is for each of the nodes (usually all nodes in a layer will all use the same σ). These are called the *transfer functions*.
  - Tell Matlab what optimization (or training) routine to use. Generally, we will use either `traingdx`, which is gradient descent, or `trainlm` (for Levenburg-Marquardt, which is a combination of gradient descent and Newton's Method).
  - Optional parts of the design: Error function (Mean Square Error is the default), plot the progress of training, etc.
- **Training:** Once the network has been designed, we "train" the network (by optimizing the error function).

  This process determines the "best" set of weights and biases for our data set.
- **Testing:** We need to test our network to see if it has found a good balance between memorization (accuracy) and generalization.

Sample Design and Training Session 1:

```
P = [0 1 2 3 4 5 6 7 8 9 10];  %Training Patterns (domain values)
T = [0 1 2 3 4 3 2 1 2 3 4];   %Training Targets (range values)

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


%Plot the original data points and the untrained output
  Y = sim(net,P);
  figure(1)
  plot(P,T,P,Y,'o')
  title('Data and Untrained Network Output')

%Train the network and plot the results
  net.trainParam.goal=0.01;  %0 is the default- too small!
  net.trainParam.epochs = 50; %For our sample, don't train too long
  net = train(net,P,T);
  X = linspace(0,10);  %New Domain Points
  Y = sim(net,X);      %Network Output
  figure(2)
  plot(P,T,'ko',X,Y)

%An alternative way to test training: postreg
  figure(3)
  Tout=sim(net,P);  %Get network output for the training domain
  [m,b,r]=postreg(T,Tout);  %Performs a linear regression
```

Training Session 1 explanation:

- Notice that the domain runs between a minimum of 0 and a maximum of 10, which are the numbers appearing in the first argument of `newff`. Notice also that the sizes of the domain and range sets are given as $m \times n$, where m =dimension and n =number of data points.
- The arguments shown in `newff` are required; notice that the first argument gives the minimum and maximum values in the domain set. Normally, one uses `minmax(P)` in place of actually typing these values in. This also implicitly defines the number of nodes in the input layer (which is the domain dimension).

  The second argument in `newff`, `[5,1]` defines the number of nodes in the hidden layer and in the output layer. This vector also implicity defines how many layers you have by counting the size of this vector. Thus, we can have as many layers as we wish.

  The last required argument is the type of transfer function, $\sigma(x)$, that will be used. The `tansig` function is the inverse tangent function, which gives the same performance as our standard sigmoidal function. We will always use a linear layer in the output, so this argument is always `purelin`.
- The default training (optimization) method is Levenburg-Marquardt (which is the default- it is a combination of gradient descent and Newton's Method). In the next training/testing session, we will use a different method.

Sample Design and Training Session 2:

This session uses a "validation" data set to test the training. That is, we retain some of our data to see if we're getting a good balance between memorization and generalization.

```
%Create training set
p=[-1:0.05:1];
t=sin(2*pi*p)+0.1*randn(size(p));

%Create a validation set
v.P=[-0.975:0.05:0.975];
v.T=[sin(2*pi*v.P)+0.1*randn(size(v.P))];

%Create a network, and train using the validation
net=newff(minmax(p),[20,1]);
net.trainParam.show=25;
net.trainParam.epochs=300;
[net,tr]=train(net,p,t,[],[],v);

%Plot the data and the network output:
X=linspace(-1,1);
Y=sim(net,X);
plot(p,t,'k*',v.P,v.T,'ko',X,Y,'r-');
legend('Training Data','Testing Data','Network')
```

In this graph, you might see a rather bad fit- Probably too many nodes in the hidden layer. Modify that number and re-run the script until you get a nice fit.

Sample Design and Training Session 3: Character Recognition

To run this code, first write the function `plotletters.m` on the next page so we can visualize the characters.

```
[alphabet,targets] = prprob; %Built in data set in Matlab
plotletters(alphabet);  %We wrote this function to display the characters
```

The following defines a network with 10 nodes in the hidden layer, and 26 nodes in the output layer. We'll use $\sigma(x) = \frac{1}{1+e^{-x}}$ (which is `logsig`) in both the hidden and output layers. We'll use a gradient descent algorithm for training:

```
net = newff(minmax(alphabet),[10 26],{'logsig' 'logsig'},'traingdx');

%These values are set to give better training results (initialization of
%the weights and biases should be small here)
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;

net.performFcn = 'sse';        % Sum-Squared Error performance function
net.trainParam.goal = 0.1;     % Sum-squared error goal.
net.trainParam.show = 20;      % Frequency of progress displays (in epochs).
net.trainParam.epochs = 5000;  % Maximum number of epochs to train.
net.trainParam.mc = 0.95;      % Momentum constant.

%    Training begins...please wait...

P = alphabet;
noisyP=alphabet+randn(size(alphabet))*0.4;
T = targets;

[net,tr] = train(net,P,T);
[net,tr] = train(net,noisyP,T);

%Test on new noisy data
noisyP = alphabet+randn(size(alphabet)) * 0.2;
plotletters(noisyP);

A2 = sim(net,noisyP);
for j=1:26 %Number of noisy letters
  A3 = compet(A2(:,j));
  answer(j) = find(compet(A3) == 1);
end
NetLetters=alphabet(:,answer);
plotletters(NetLetters);
```

This is the function `plotletters` so we can visualize the characters...

```
function plotletters(alphabet)

fprintf('plotletters is plotting the first 25 letters\n');

[m,n]=size(alphabet);
if m~=35
    error('plotletters needs columns 35 numbers long');
end

figure
MM=colormap(gray);
MM=MM(end:-1:1,:);
colormap(MM);
nn=min([n,25]);
for j=1:nn
    subplot(5,5,j)
    imagesc(reshape(alphabet(:,j),5,7)');
    axis equal
    axis off
end
```