1 The Options and Structures in the Neural Net

These notes are broken into several parts to give you one place to look for the most salient features of a neural network.

1.1 Initialize the Neural Network

There are three main commands we'll look at to build neural nets: `linearlayer` (to build a linear network), `feedforwardnet` (to build a nonlinear network), and `nnstart` (which gives a dialog box to lead you through the nonlinear network).

Here are some examples of each:

1. Construct a linear neural network with learning rate (Widrow-Hoff) set to 0.01, then train:

```
net1=linearlayer(0,0.01);
X=[2,1,-2,-1;2,-2,2,1];
T=[-1,1,-1,1];
net1=train(net1,X,T);
```

2. Build a 3 layer network with 10 nodes in the (single) hidden layer:

```
net=feedforwardnet(10);
```

3. Build a 4 layer network with 5 nodes in the first hidden layer and 3 nodes in the second hidden layer:

```
net=feedforwardnet([5,3]);
```

1.2 Define the transfer function for each layer (Optional)

The default transfer function for the hidden layers are `tansig` (which is the hyperbolic tangent). Other common options are `logsig`, and `purelin`:

$$\text{logsig}(r) = \frac{1}{1+\mathrm{e}^{-r}} \qquad \text{purelin}(r) = r$$

To see the information for each layer, type:

```
net=feedforwardnet(10);
net.layers{1}
net.layers{2}
```

To change the transfer function to the exponential, we would type:

```
net.layers{1}.transferFcn='logsig';
```

1.3 The Weights and Biases

There are two sets of weights, `IW` and `LW`. The matrix associated with `IW` is for the initial layer (from input layer to hidden layer), and `LW` is going from hidden to output layer, or to the next hidden layer.

Here's an example, where we build a $2-5-3-1$ network using some data we've seen before in the linear net, and examine the weights and biases:

```
net2=feedforwardnet([5,3]);
X=[1,0.5,-1,-0.5;1,-1,1,0.5];
T=[-1,1,-1,1];
net2=train(net2,X,T);
```

The initial matrix of weights is the 5×2 matrix: `net2.IW{1}`

The bias vector for that is `net2.b{1}`.

The second matrix of weights is the 3×5 matrix: `net2.LW{2,1}` with bias `net2.b{2}`

The third matrix of weights is the 1×3 matrix: `net2.LW{3,2}` with bias `net2.b{3}`

We can manually compute the output of the network:

```
P1=net2.IW{1}*X(:,1)+net2.b{1};  %Prestate of first hidden layer
S1=tansig(P1);                   %State of the first hidden layer
P2=net2.LW{2,1}*S1+net2.b{2};    %Prestate of the second hidden layer
S2=tansig(P2);                   %State of the second hidden layer
P3=net2.LW{3,2}*S2+net2.b{3};    %Output of the network
```

Important Note: The network will automatically scale each dimension of the input data to be between -1 and 1 (to work with the `tansig` function). These parameters are in the structure `net2.input` for the previous example.

1.4 Options in Training Algorithms

We'll typically stay with the default algorithm, but it's good to know where it is defined. Matlab gives 12 algorithms to choose from (See pages 2-18, 2-19 in the User's Guide). Here are some typical choices:

- `trainlm` is Levenberg-Marquardt training, and is an excellent general algorithm. It is also Matlab's default.
- `trainrp` is called "Resilient Backpropagation", and can be much faster than `trainlm` when you have a large network.
- `traingdx` is variable learning rate Gradient Descent.

Many of the training parameters are in the `trainParam` structure. To see these, we might type something like:

```
net=feedforwardnet(10);
net.trainFcn
net.trainParam
```

We see that the default training function is `trainlm` (which is an excellent multi-purpose algorithm).

To change the goal of training from 0 to something more realistic, like 0.05, and to change the training to `trainrp`, we would type:

```
net.trainParam.goal=0.05;
net.trainFcn='trainrp';
```

2 Commands on the Networks

We can `train` the network (meaning we find the best weights and biases for a given data set), and we can `sim` or simulate the network using new data to get the network's output.

Here's an example, where we train the network on some simple data, then we construct some new data to put into the network and plot the results:

```
net=feedforwardnet(10);
X=linspace(-2,2,10);
T=sin(X)+0.01*randn(1,10);
net=train(net,X,T);
xx=linspace(-2,2,100);
yy=sim(net,xx);
plot(X,T,'*',xx,yy);
```

Analyzing the results

It may be useful to use a different version of the `train` command so that we can plot the training record separately. Here's an example of that- We define a new structure, `tr`

```
net=feedforwardnet(10);
X=linspace(-2,2,10);
T=sin(X)+0.01*randn(1,10);
net=train(net,X,T);
[net,tr]=train(net,X,T);
```

And now we can plot the training record:

```
plotperf(tr);
```

We can also look at regression plots between the targets and the outputs, but that would be outside the scope of this class (see the User's Guide for more information).

Matlab by Example: Linear Network and Widrow-Hoff

In Example 1 of the linear NN handout, we had the input and target data:

$$X = \begin{bmatrix} 2 & 1 & -2 & -1 \\ 2 & -2 & 2 & 1 \end{bmatrix} \qquad T = [-1, 1, -1, 1];$$

and using the batch algorithm (Matlab's slash command), we got the weight matrix W and bias vector $\mathbf{b}$:

$$W = [-0.1523, -0.5076] \qquad b = 0.3807$$

To do the same training in Matlab, here is the set of commands and where the parameters are stored. The `linearlayer` command takes two inputs: The first is for delayed input (which we won't do), so we input 0. The second input is the learning rate for Widrow-Hoff. In this example, we'll set it to 0.01:

```
net1=linearlayer(0,0.01);
X=[2,1,-2,-1;2,-2,2,1];
T=[-1,1,-1,1];
net1=train(net1,X,T);
net1.IW{1}
net1.b{1}
```

Matlab by Example: Solution to Exercise 1, pg. 13

We'll do this two ways- Using the program `wid_hoff1` that we wrote, then using Matlab's neural network toolbox:

- Using our built-in function:

```
X=[1,1,2,2,-1,-2,-1,-2;1,2,-1,0,2,1,-1,-2];
T=[-1,-1,-1,-1,1,1,1,1;-1,-1,1,1,-1,-1,1,1];
lr=0.04;
iters=60;
[W,b,EpochErr]=wid_hoff1(X',T',lr,iters);
```

- Using Matlab's toolbox:

```
X=[1,1,2,2,-1,-2,-1,-2;1,2,-1,0,2,1,-1,-2];
T=[-1,-1,-1,-1,1,1,1,1;-1,-1,1,1,-1,-1,1,1];
net2=linearlayer(0,0.04);
net2=train(net2,X,T);
net2.IW{1}
net2.b{1}
```

In each case, W and b are approximately:

$$W = \begin{bmatrix} -0.57 & -0.02 \\ 0.20 & -0.65 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 0.018 \\ 0.20 \end{bmatrix}$$

For the graphics commands, type:

```
tt=linspace(-2,2);
y1=(-W(1,1)*tt-b(1))/W(1,2);
y2=(-W(2,1)*tt-b(2))/W(2,2);
plot(X(1,:),X(2,:),'x',tt,y1,tt,y2);
axis([-2,2,-2,2]);  %Fix the viewing window
```

These commands are all summarized in the script file `LinearExample.m`.

3 Matlab by Example: Character Recognition

These notes explain the commands used in the Matlab example file `appcr1.m`. If you would like to follow along, you can view these commands by typing: `edit appcr1.m` in the Matlab command window.

3.1 Part 1: Get the data

The data consists of the 26 characters of the alphabet. Each letter is actually a 7×5 grid of numbers, where 1 is black and 0 is white. In Matlab, there is a program that will create the letters and the targets for this character recognition example:

```
[X,T]=prprob;
```

Now the array X is 35×26 and T is 26×26. If you want to view a certain character, there is a built-in program for this example. To see the letter A for example, type

```
plotchar(X(:,1))
```

3.2 Part 2: Define the network

In this case, we'll construct a feedforward network with 25 nodes in the hidden layer (you would either guess at the number you need, or be told that number in the problem).

The key command here is `feedforwardnet`:

```
net1=feedforwardnet(25);
```

3.3 Part 3: Train the network

Now, we want to train the network. There is some default stuff happening in the background: (1) The data is scaled to match the transfer function, (2) the data is separated into three sets- Training, Testing and Validation, and then (3) training begins.

The `appcr1.m` script also uses an optional argument in the training, `nnMATLAB` (this optimizes some of the training, so it can be useful, but not mandatory):

```
net1=train(net1,X,T,nnMATLAB);
```

3.4 Part 4: Make the network robust

Often we do not do this step, but it can be advantageous to train the network on noisy copies of the data. In this case, we create Xn, Tn as noisy copies of X and T, and re-train:

```
numNoise=30;
Xn= min(max(repmat(X,1,numNoise)+randn(35,26*numNoise)*0.2,0),1);
Tn = repmat(T,1,numNoise);

net2=feedforwardnet(25);  %Create a new network to compare
net2=train(net2,Xn,Tn,nnMATLAB);
```

3.5 Analyze the Results

In the script file, you'll see a test using noisy versions of the data. Typically we don't re-run the networks like this- We'll look at ways of analyzing our results as time permits.

Here is a short script so you can see if you can beat the neural network. You should run this after running `appcr1.m`. This file is on our class website as `testappcr1.m`:

```
N=5;                    % Run the game N times
noiseLevel=0.4;         % Use this much noise (between 0 and 1)
randidx=randi(26,N,1);  % Randomly select letters
for i=1:N
  Letter=X(:,randidx(i))+noiseLevel*randn(35,1);
  t=sim(net2,Letter);
  [maxval,maxidx]=max(t);
  figure(1)
  plotchar(Letter)
  title('What Matlab Sees');
  pause
  figure(2)
  plotchar(X(:,maxidx))
  title('Matlab Predicts');
  pause
```

```
  figure(3)
  plotchar(X(:,randidx(i)));
  title('Actual Character');
  pause
end
```

3.6 Bottom Line

Notice that if I simply wanted to create the data, create the network, and train the neural network, this can be done very quickly:

```
[X,T]=prprob;
net1=feedforwardnet(25);
net1=train(net1,X,T);
```

4 Sample Questions

1. Build a 1-10-1 network to model the `simplefit _dataset` that comes with Matlab. That is, in Matlab type:

   ```
   [x,t]=simplefit_dataset;  %Data comes with Matlab
   plot(x,t);
   %Initialize the network using 10 nodes in the hidden layer.
   net=feedforwardnet(10);
   net=train(net,x,t);
   ```

   At this point, a dialog box comes up to illustrate how the training is proceeding. Once done, you might take a look at some of the plots- In particular, the "Performance" plot and the "Error Histogram" are kind of interesting.

   Try using a different form of the training command, and try reducing the number of hidden nodes.

   ```
   net=feedforwardnet(5);
   [net,tr]=train(net,x,t);
   ```

   The output `tr` is a Matlab structure that contains a lot of information. To see the plot from the training, you can type:

   ```
   h=length(tr.perf);
   plot(1:h,log(tr.perf),1:h,log(tr.vperf),1:h,log(tr.tperf))
   ```

2. PROBLEM: Build an 8-15-15-2 feedforward neural network that uses `trainrp` for the training method, and uses a training goal of 0.05. Train the data on the Matlab file on our class website, `diabetes1.mat`

   SOLUTION:

```
net=feedforwardnet([15 15]);
net.trainParam.goal=0.05;
net.trainFcn='trainrp';
load diabetes1;  %You have to download this
net=train(net,P',T');  %The data needs to be transposed.
```

3. PROBLEM: Get some random data for your input and output- Say 350 points in $\mathbb{R}^3$ for the domain, and 350 points in $\mathbb{R}^2$ for the range. Build a $3-8-5-2$ network and train it (output `tr` as well as `net`).

   (a) Plot the performance measures from the training output (not the dialog box).

   (b) There are three sets of weights. Where are they in the network structure?

   (c) There are three bias vectors. Where are they in the network structure?

   (d) Find where the transfer function is defined for each layer (Hint: `layers`) and change all of them to `logsig` (except input and output layers). Re-train.

   (e) Investigate the "Plot Regression" option. In particular, the program finds the linear relationship between the "outputs" and the "targets" (the closer that R is to 1, the better). Use `plotregression` directly to see what relationship you get.

4. Use the `simpleclass_dataset` (a built in data set) to try a pattern classifier:

```
load simpleclass_dataset;
net=feedforwardnet(20);
net=train(net,simpleclassInputs, simpleclassTargets);
SampleOut=sim(net,simpleclassInputs);
plotconfusion(simpleclassTargets,SampleOut);
```

5. Did your previous network perform any pre-processing of the data? To see and/or change the option, go to:

```
net.inputs{1}
net.outputs{1}
```

   The preprocessing options are: `mapminmax`, `mapstd`, `processpca`, `fixunknowns`, and `removeconstantrows`