MATLAB 6.1 includes in its collection of toolboxes a comprehensive API for developing neural networks. Along with console-based programming facilities, MATLAB 6.1 includes an easy-to-use GUI that encapsulates all functions available in the Neural Networks Toolbox. You will be introduced to this toolbox in this short tutorial.

NOTE: Bear in mind that functionality discussed here is limited to what you will need for the next lab. Since the toolbox is extensive, it would be beneficial to read up on its documentation (available via the link on the course web-site).

Using the NNET Toolbox

To introduce you to the extensive NNET add-ons that MATLAB includes as part of its package, we will start off with the function `newlin`.

Consider the simple feedforward network below.

The simulation data set consists of:

\[ x_A = (1,2) , \ x_B = (2,1) , \ x_C = (2,3) \ \text{and} \ x_D = (3,1) \]

We wish to provide these inputs to the network concurrently (all at once). For this, we present the input set as a single matrix:

\[ X = \begin{bmatrix} 1 & 2 & 2 & 3; 2 & 1 & 3 & 1 \end{bmatrix}; \]

To declare the network we use the following command:

\[ \text{net} = \text{newlin}([0 10; 0 10],1); \]

Try this command out without the semicolon and observe the architecture details displayed.
The weight matrix \( W \) and \( b \) are set to \([ 1 2 ]\) and \([0]\) respectively. This is achieved by:

\[
\text{net.IW\{1,1\} = [1 2];}
\]
\[
\text{net.b\{1\} = 0;}
\]

We now simulate/run the network to store outputs in \( A \) as:

\[
A = \text{sim}(\text{net},X)
\]
\[
A = 5 \quad 4 \quad 8 \quad 5
\]

Notice that \( A \) is a single matrix, not a cell array. This is because the input set is concurrent, i.e. a single matrix. To input data sequentially, we use cell arrays instead of a single matrix; each cell element consists of a single input vector instance.

**Training using ADAPT**

The ADAPT function is used for incremental training, i.e. weights and biases are updated after each time an input vector is presented. Suppose we are given the same input data set as above and are also given a set of corresponding target values as \( t_1 = 4, t_2 = 5, t_3 = 7, t_4 = 7 \). We set initial weights and biases to zero.

We present our input to the network as a series of inputs, given by:

\[
X = ([1;2] \quad [2;1] \quad [2;3] \quad [3;1])
\]

Notice that \( X \) is a cell array this time. What does this tell us about the method of input presentation? \( T \) is stored as:

\[
T = [4 5 7 7];
\]

We also need to set the learning rate for the network. You may have noticed in the description for net the term `learnParam`. This has as a member the value \( lr \), which is by default set to zero. We may change this by the following commands:

\[
\text{net.inputWeights\{1,1\}.learnParam.lr} = 0.1;
\]
\[
\text{net.biases\{1,1\}.learnParam.lr} = 0.1;
\]

Type in `net.inputWeights\{1,1\}` and then press TAB twice (similar to how we use the TAB on the UNIX/LINUX console). You will see a list of parameters:

\[
\text{delays} \quad \text{learn} \quad \text{learnParam} \quad \text{userdata}
\]
\[
\text{initFcn} \quad \text{learnFcn} \quad \text{size} \quad \text{weightFcn}
\]

As in C/C++, you may access these member variables using a `.` operator. Access the `learnFcn` viz:

\[
\text{net.inputWeights\{1,1\}.learnFcn}
\]
\[
>>ans = \text{learnwh}
\]

This tells you which learning technique is being used to update weights. The `learnwh` rule i.e. the *Widrow-Hoff* rule is also known as the *delta rule*.
Finally, the command to train the network is:

```
[net,a,e,pf] = adapt(net,X,T);
```

where a = output of network and e = error

This yields the output:

```
a = [0] [2] [6.0] [5.8]
e = [4] [3] [1.0] [1.2]
```

Notice how the error decreases as each input is processed. If the learning rate is set correctly, the error will eventually be minimized (to zero in the extreme case).

Try the above example out to see that it works.
The NNTool: A Neural Network GU Interface

This is a short sample-based tutorial that will get you started with using NNTool, a GUI tool included in the Neural Networks Toolbox 4.0.1.

Type in NNTool at the console to open up the Network/Data Manager GUI:

There are two ways of accessing input/target data using NNTool. The first is via Import; the other is via New Data.

Clicking on New Data opens the following Dialog box:
Values may be specified within the **Value** text area. **Data types** are also specified using the radio button set. If you wish to import data, click on **Import** to open the following dialog box:

![Import dialog box]

As you can see, variables that you have declared and initialized in your workspace are accessible for import to your network. Here we import the variable **IN** as **Input** to our network.

**Creating a new network**

Click on **New Network** to create a fresh network. This opens up a comprehensive dialog box which allows you to construct a network using the same functions available via console. Click on **Create** to add the new network to your Manager. Click on **View** for a graphical illustration of your network architecture.

![Create new network dialog box]
Using the NNTool to construct and train a sample network

Refer to the example problem presented in the earlier section. Given four input pairs, viz.

\[ \mathbf{x}_A = (1,2) , \mathbf{x}_B=(2,1), \mathbf{x}_C = (2,3) \text{ and } \mathbf{x}_D= (3,1) \]

And their corresponding targets:

\[ \mathbf{T} = [4 \ 5 \ 7 \ 7] ; \]

We first instantiate \( \mathbf{X} \), a single array holding all inputs, and \( \mathbf{T} \) (as shown above) in our workspace. After starting NNTool, we import both \( \mathbf{X} \) and \( \mathbf{T} \) as input and tgt respectively.

Our network is a linear single-layer with input ranges and learning rate set as shown below:
Now we are ready to **Train** the network.

You are required to specify the **inputs** and **targets** you wish to use for your network in training.
Other parameters are set using the available tabs.

After setting epochs and goal to the above, click Train Network.

We observe that our goal is reached after 252 epochs. The network sampleNetwork has been trained.
The **Initialize** options may be used to reset weights and input ranges for training. Use the **Simulate** option for testing your network.

---

**Exporting/Saving data**

Click on **Export** in the main **Network/Data Manager**. You may either save your variables to disk as a `.mat` file or export them to your workspace.
For further elaboration regarding the Neural Network toolbox, please try out a few examples using the NNET toolbox and browse through the documentation provided on the course web-site.

The coming lab will be based on usage of the toolbox as described above. It is your responsibility to come prepared. Good luck!