CSG2341 Intelligent Systems

Workshop 4: Predicting risk of diabetes

Related Objectives from the Unit outline:

- Identify appropriate intelligent system solutions for a range of computational intelligence tasks.
- Demonstrate the ability to apply computational intelligence techniques to a range of tasks normally considered to require computational intelligence.

Learning Outcomes:

After completing this workshop, you should be able to demonstrate an understanding of the resources required to implement a problem solution based on a multi-layer perceptron, to describe and analyse the steps in designing a neural network to solve a prediction problem and use a computer package to develop such a prediction system.

Background

Multi-layer perceptrons are a type of artificial neural network that can be used to solve problems of prediction and classification. Using a dataset of training examples, a multi-layer perceptron can be "trained" to solve such problems, using the back-propagation algorithm, or some other learning algorithm.

In designing and training such a network, there are various parameters that must be specified, including

- The mapping of input values to input neurons;
- The mapping of output neuron values to predictions or classifications;
- The number of layers of neurons;
- The number of hidden neurons in each hidden layer;
- The learning rate;
- The momentum rate;
- The number of epochs or other stopping conditions to use.

Once the network design and training parameters are selected, it is also important to accurately estimate the accuracy of the trained prediction or classification system.

One way to do these things in a principled manner is to use **cross-validation testing**. This method will be explained in next week's lectures.

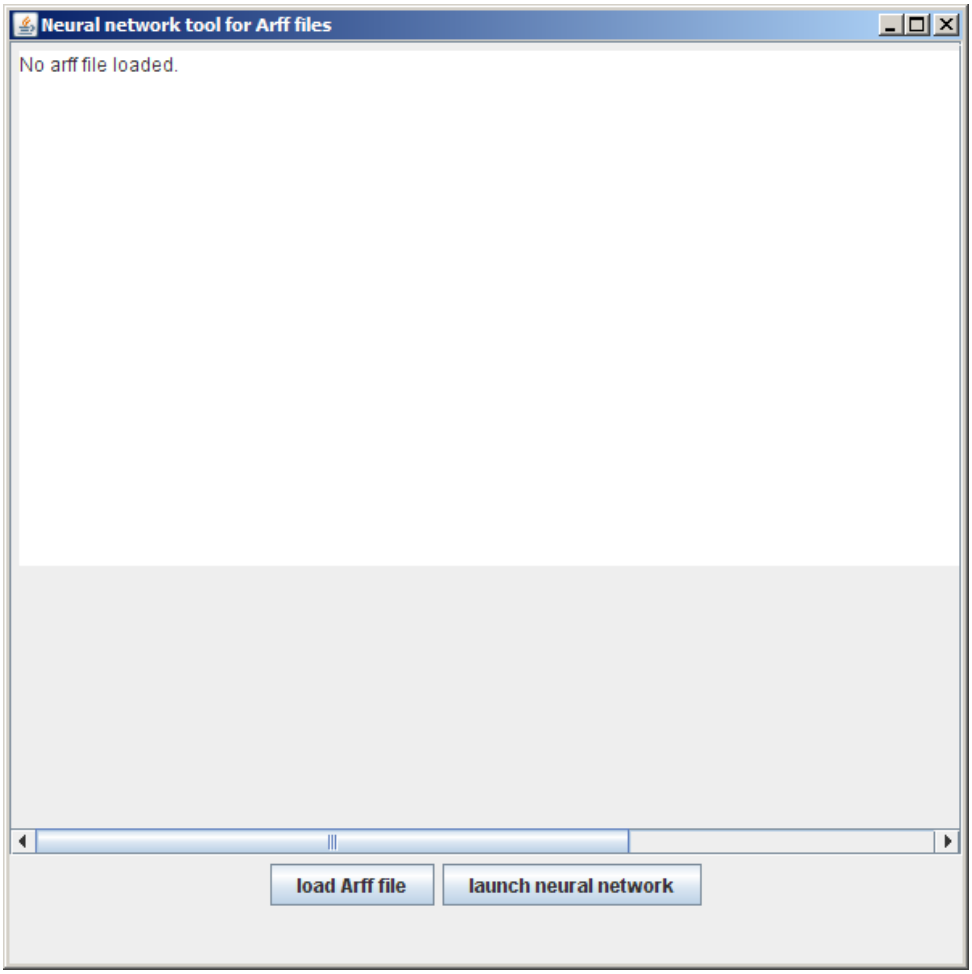
Task:

In this workshop, you will use a tool (built using the IS library) that lets you design, train and test a neural network to solve a prediction problem, using cross-validation testing.

The task is to predict a measure of mortality rates from population data and pollution levels.

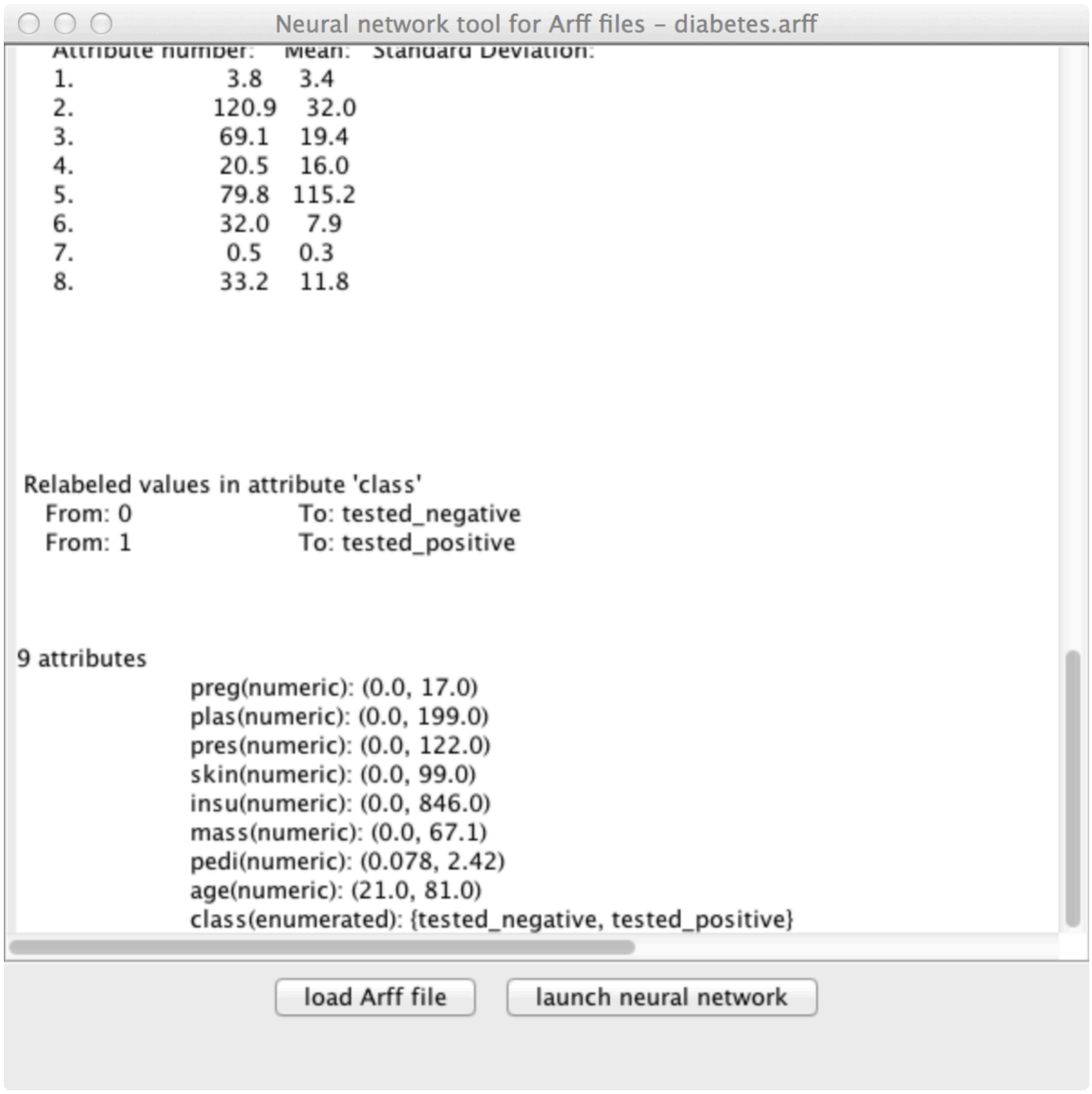
Step 1

Download *NNArffToolDist.zip* and *diabetes.arff* from BlackBoard. Unzip the zip file to a working folder. Inside you will find a jar file called *NNArffTool.jar*. Double-click on it to run. You should see a window like this:



Step 2

Click on “load Arff file”. A file selection dialog will open. Navigate to diabetes.arff and select it. You should then see something like this:

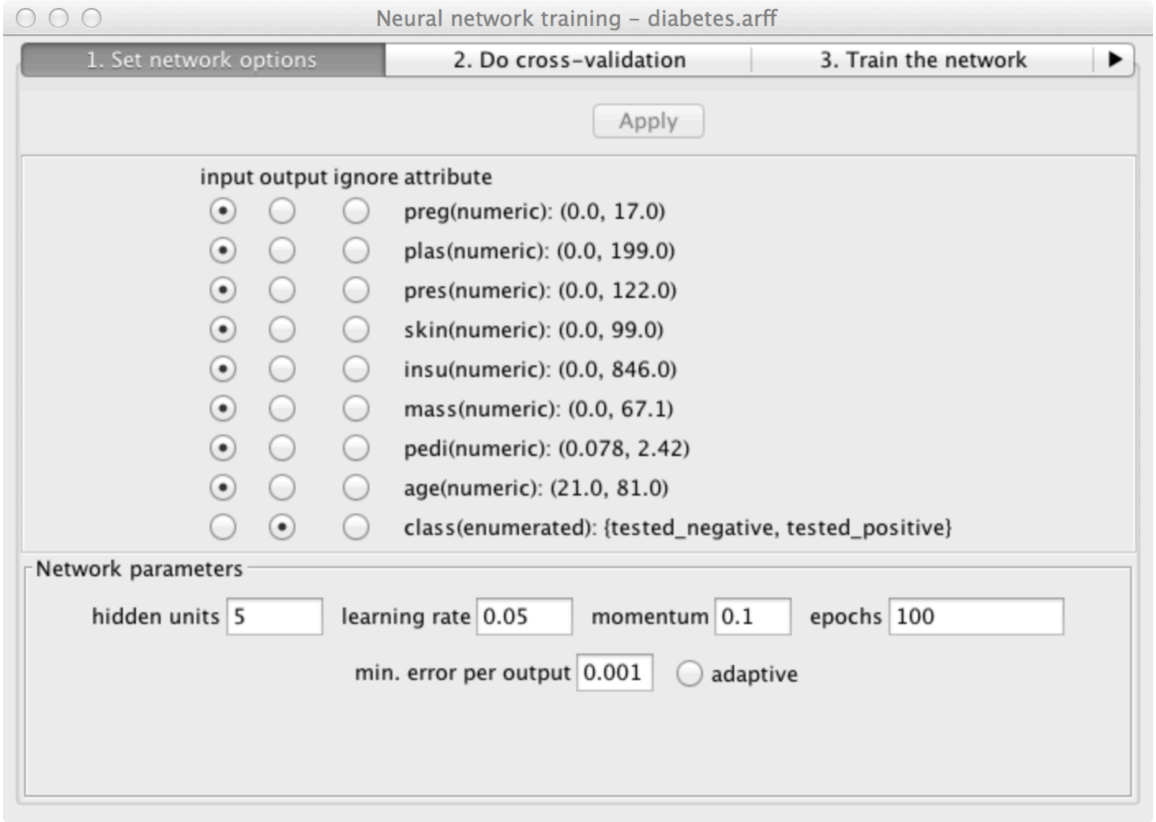


The text here describes the dataset that has been read in. There are 768 instances (training patterns). Each instance has 9 attributes, 8 numeric ones, plus a categorical one. The first one is called preg, and the minimum value read for preg was 0, and the maximum was 17.0. This is one of the inputs for the prediction problem. So are the other attributes, down to age. The final one, class, is the output, i.e. this is the value that we are trying to predict based on the 8 input values.

If you scroll up to the top of the window, you will find some additional description about this dataset and the attributes.

Step 3

Now click on “launch neural network”. Another window should open that looks like this:



Notice the numbered tabs across the top. These are used to switch between various tasks that you can now perform. Initially, you are on the “Set network options” tab. Here you can set the various parameters for network design and training.

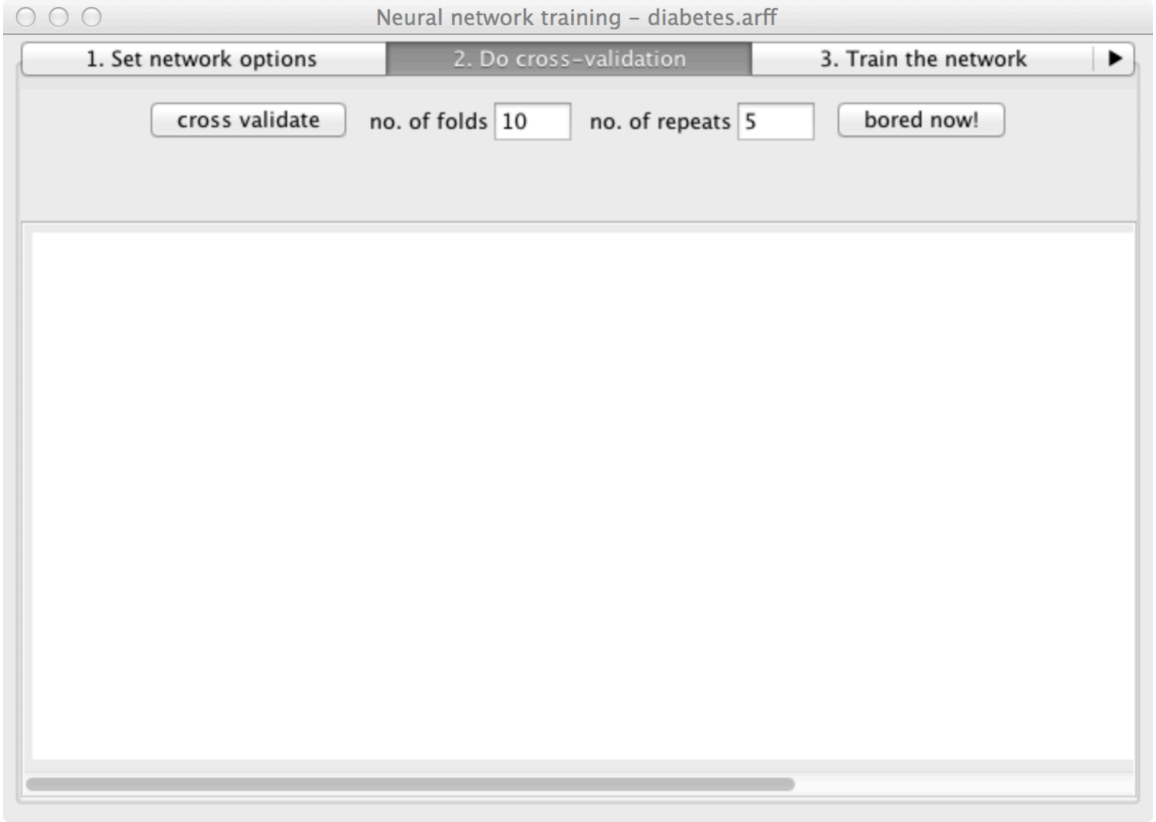
NNArffTool always creates a multi-layer perceptron (MLP) with one input layer, one hidden layer, and one output layer. (Difficult problems may need more layers, but you can’t create these with NNArffTool.)

Initially, the last attribute is assumed to be the output, and the rest are assumed to be inputs. There are also default values for the number of hidden units, learning rate, momentum, number of epochs, target error per output, and whether adaptive learning rates are used.

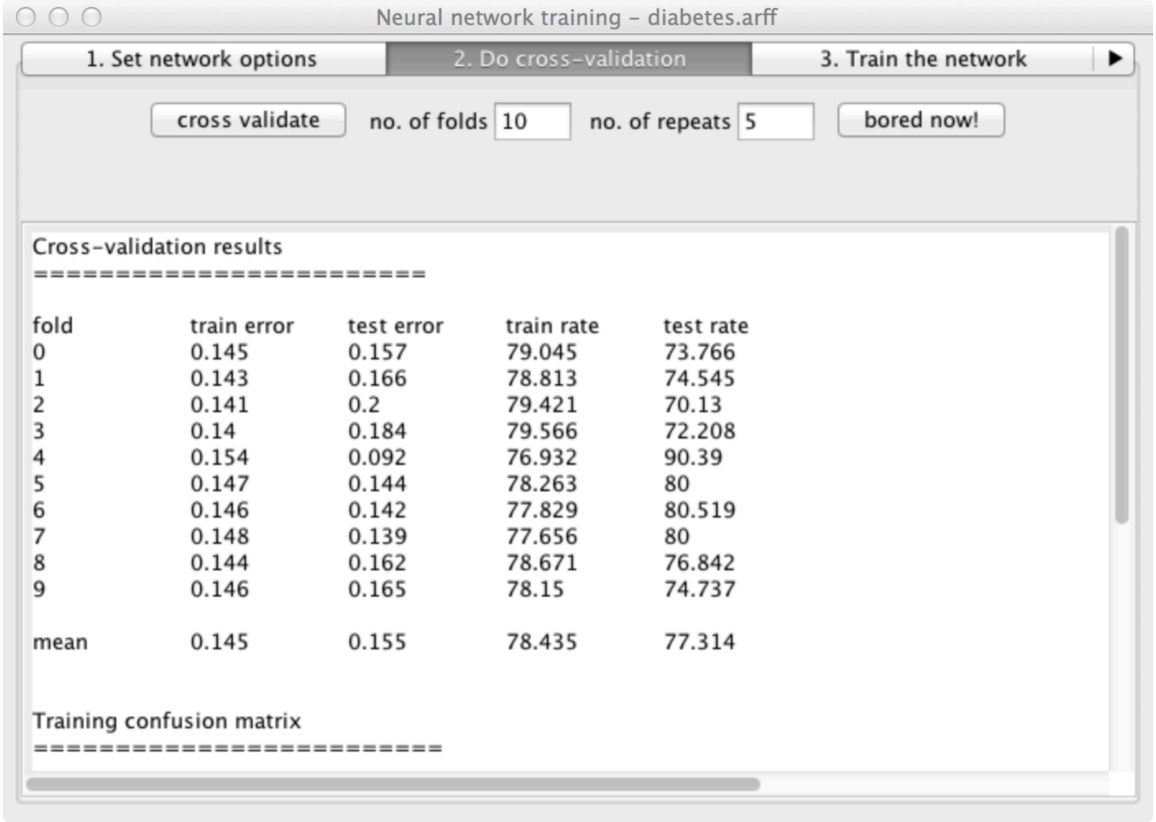
We will accept all these default values at the moment except for “epochs”, which we will change to 500. (This is the number of training cycles used to train the network).

Step 4

Now click on the “Do cross-validation” tab. You should see this:



Click on “cross validate”. You will see text appearing in the centre panel. You should wait for a while, until the window looks something like this:



Next week in lectures, you will learn what this all means! For the moment, we are only interested in the number in the bottom-right corner, 77.314 in the example.

The program has created and trained a large number of neural networks to predict class from the input values, using the parameters we chose. The figure 77.314 is the average % of time that the network correctly predicted the class. Can we do better?

We are going to try to do better by changing “epochs” and “hidden units”. Create a table like this and enter your average error:

hidden\epochs	100	500	1000
3			
5		77.314	
10			

Step 5

Now go back to the “Set network options” tab and change “epochs” to 1000. Then return to “Do cross validation” and click “cross validate” again. It will take longer to complete this time. Add the new average error to your table, like this:

hidden\epochs	100	500	1000

3			
5		77.314	77.567
10			

Now do the same with 100 epochs. Your table might look something like:

hidden\epochs	100	500	1000
3			
5	76.77	77.314	77.567
10			

Step 6

Now go back to the “Set network options” tab and change “epochs” to 100 and “hidden” to 10 (this is the number of hidden units in the network). Cross validate and add the mean error to your table like this:

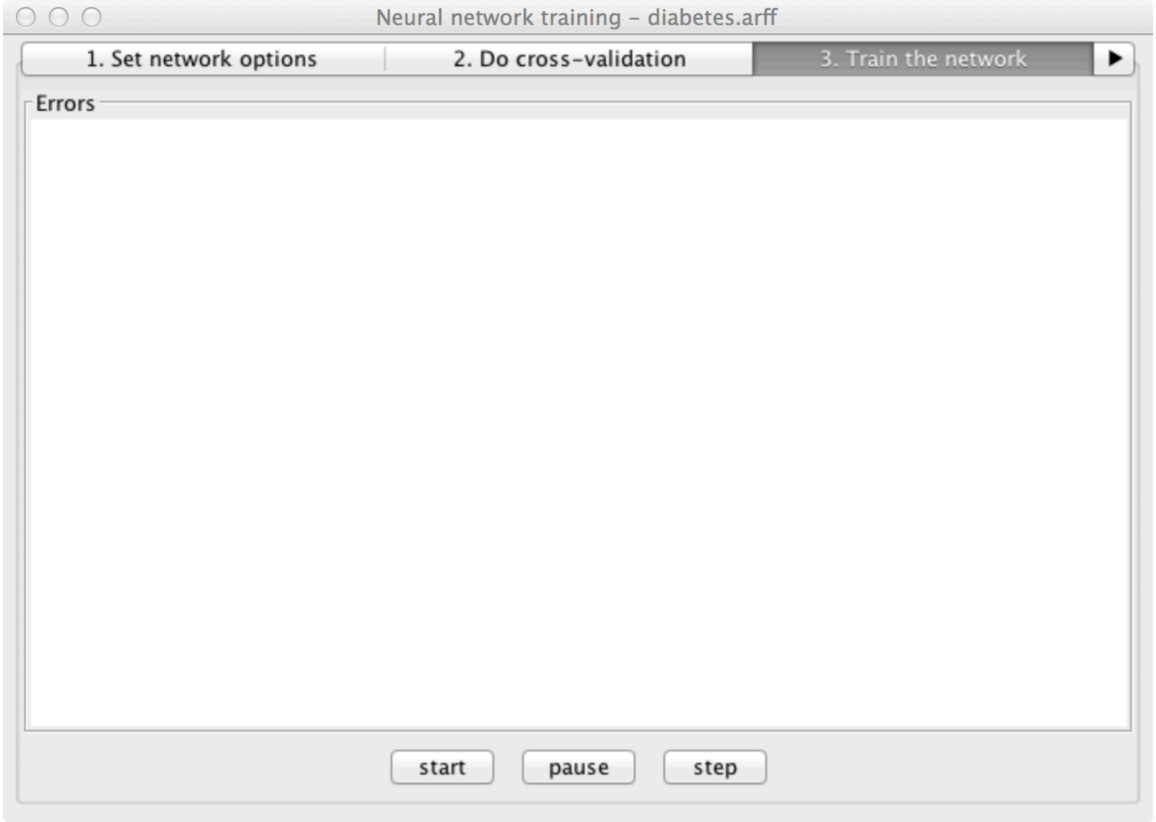
hidden\epochs	100	500	1000
3			
5	76.77	77.314	77.567
10	76.489		

Complete the rest of the table in this way. Of course, we could try more options than 3, 5, or 10 for the number of hidden units, and more options than 100, 500, or 1000 for the number of epochs. And we could try different values for the other parameters too.

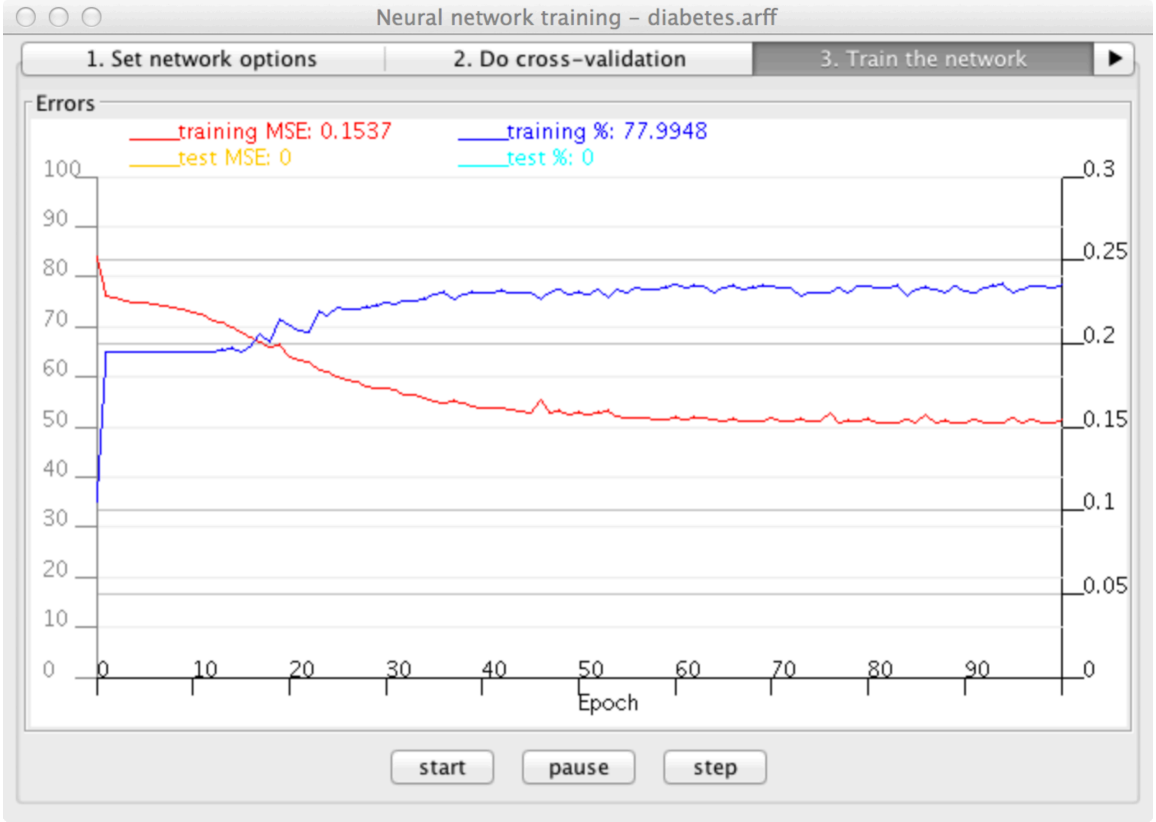
But for now, reset the number of epochs and the number of hidden units to whatever combination gives the best test rate.

Step 7

Click on the “Train the network” tab. You should see this:



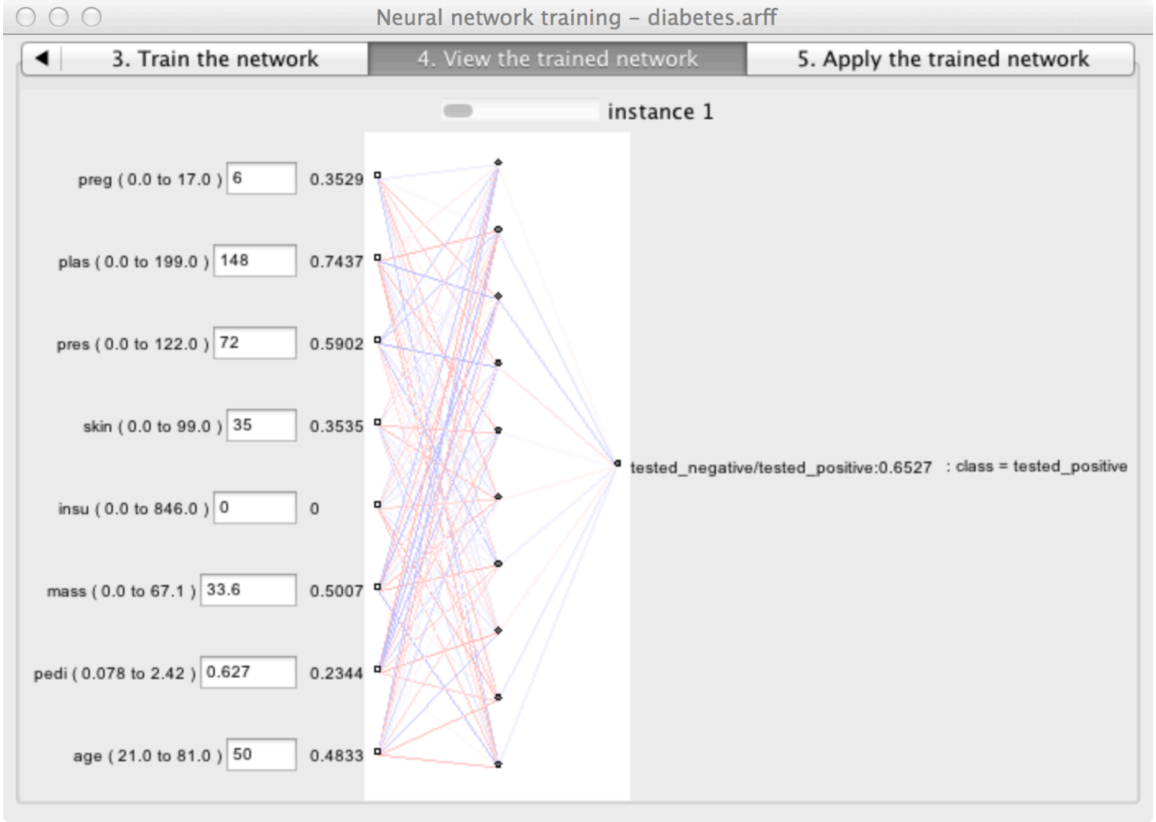
Click on “start”. This will create and train a network with the chosen parameters. You should see something like this:



Here you can see that the MSE (mean-squared error) starts around 0.25 and drops to about 0.15 after 100 epochs, while the training % starts around 35% and goes up to just below 80%.

Step 8

Now click on the “View the trained network” tab. You should see something like:

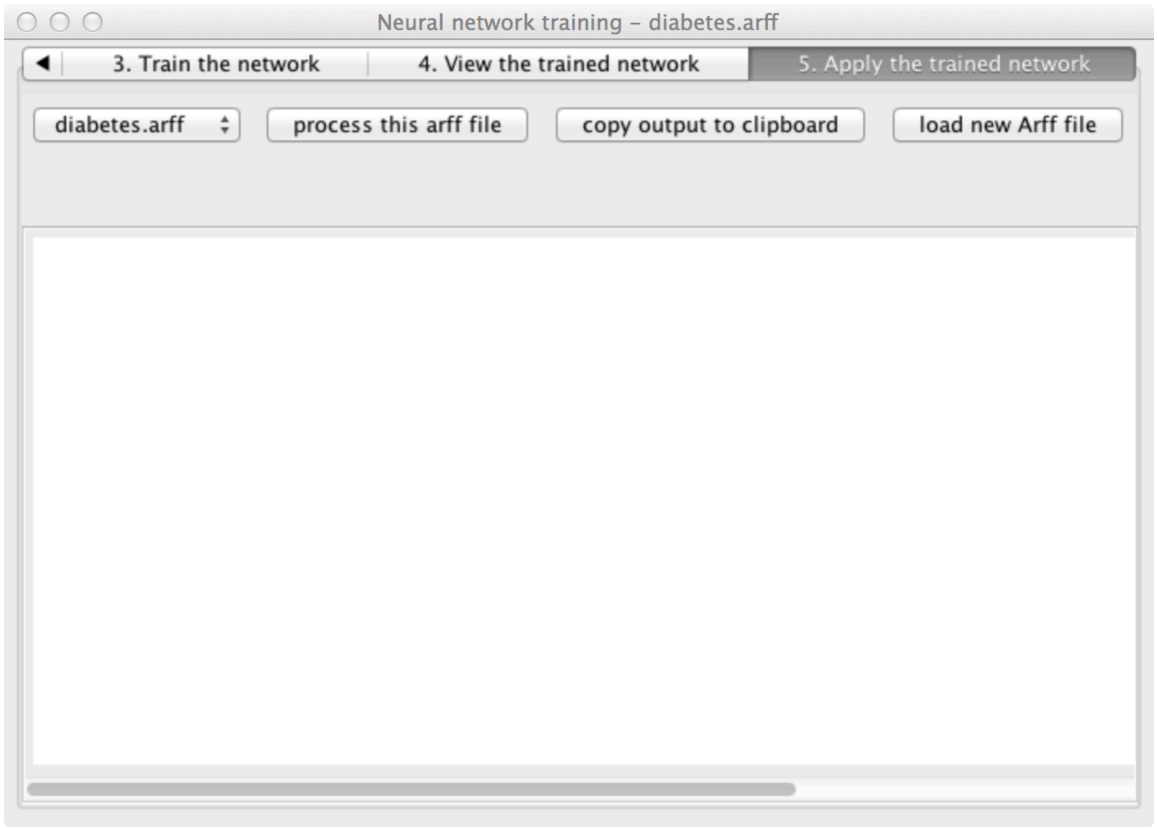


This is a diagram of the neural network. On the left are the input values for the first training pattern (instance 1). You can look at the other patterns using the scroller near the top. The raw input values are in the editable text fields. The pre-processed values are shown just to the right (e.g. 6 is mapped to 0.3529), and these values are used to determine how light or dark the input neurons (the small rectangles) appear. In the middle of the page are the hidden units (the small circles), and their output values determine how light or dark they appear. On the right hand side are the output neurons (in this case, just one), and to the right of that is its output value (0.6524) and the final predicted class.

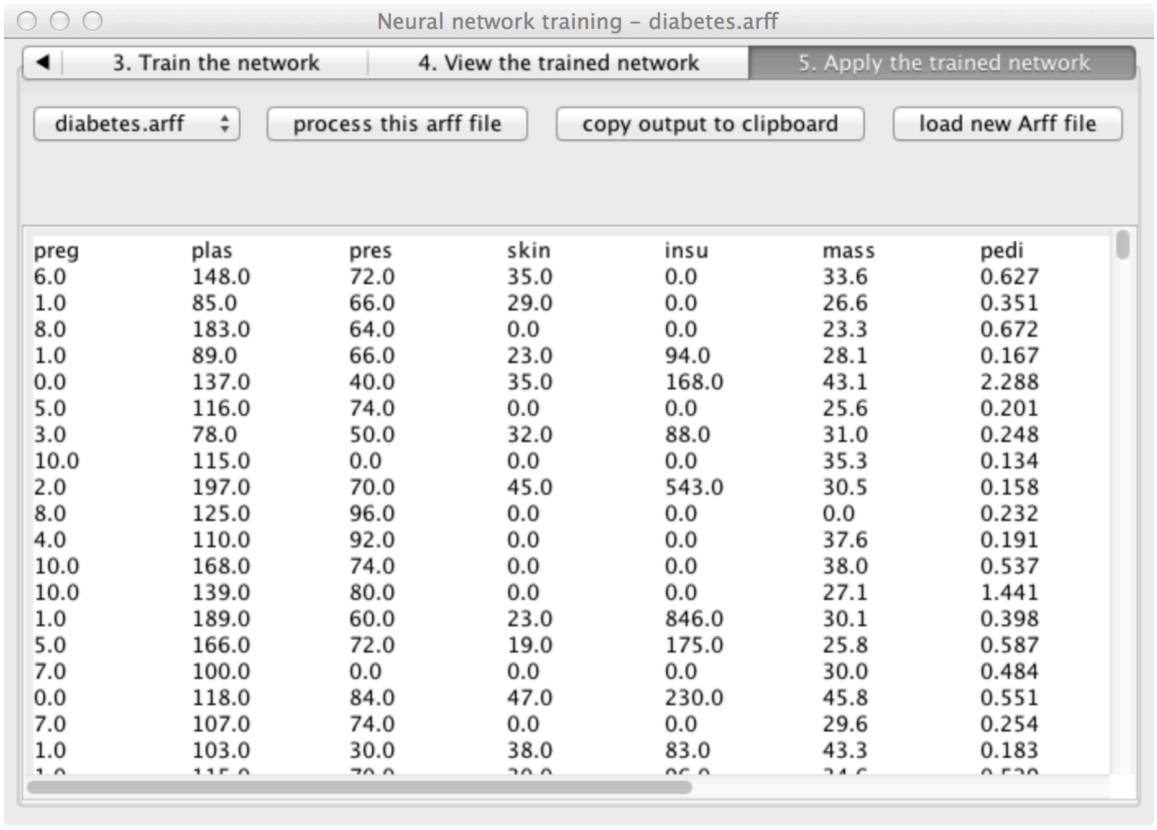
You can also see coloured lines connecting the neurons. These are colour coded according to their weight values: negative ones are red and positive ones are blue. Light colours are used for weights that are near to zero, and stronger colours indicate weights far from zero. These are the more important connections.

Step 9

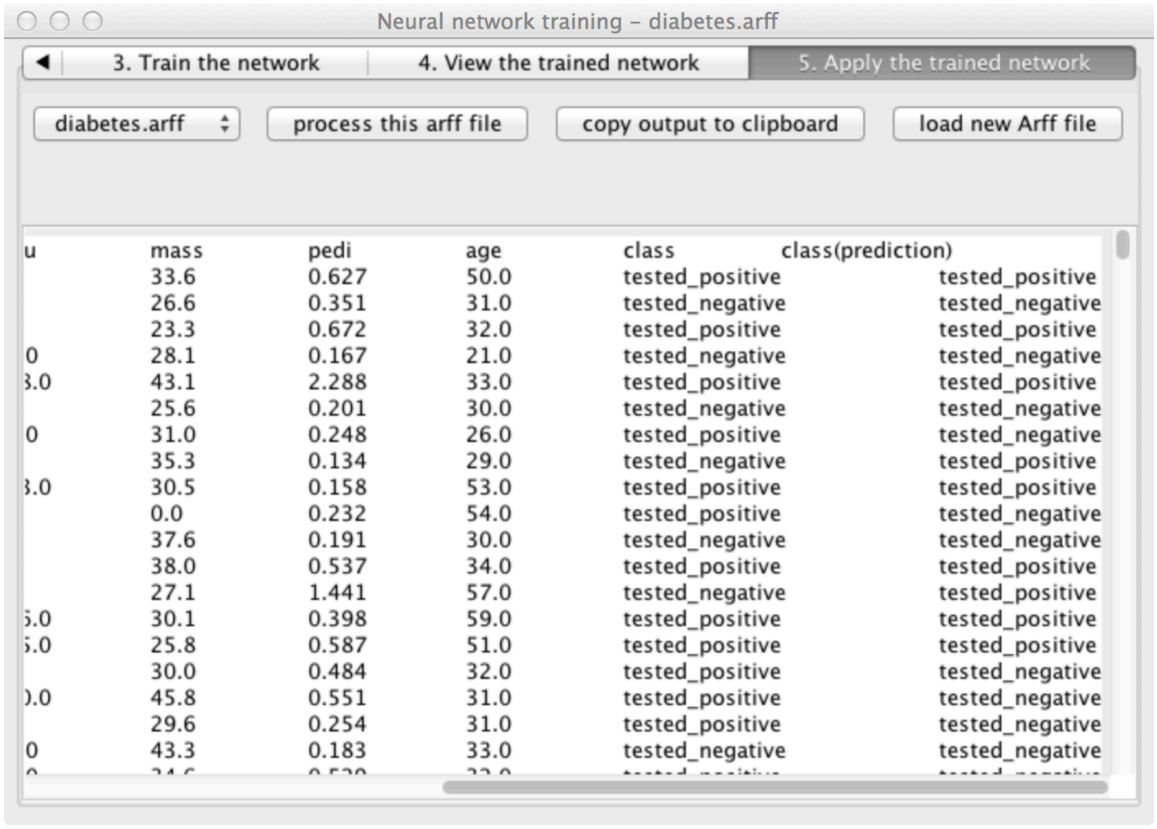
Click on “Apply the trained network”. You should see this:



Click on “process this arff file” and you should see something like:



Scroll across to the right and you should see:



u	mass	pedi	age	class	class(prediction)
	33.6	0.627	50.0	tested_positive	tested_positive
	26.6	0.351	31.0	tested_negative	tested_negative
	23.3	0.672	32.0	tested_positive	tested_positive
0	28.1	0.167	21.0	tested_negative	tested_negative
3.0	43.1	2.288	33.0	tested_positive	tested_positive
	25.6	0.201	30.0	tested_negative	tested_negative
0	31.0	0.248	26.0	tested_positive	tested_negative
	35.3	0.134	29.0	tested_negative	tested_positive
3.0	30.5	0.158	53.0	tested_positive	tested_positive
	0.0	0.232	54.0	tested_positive	tested_negative
	37.6	0.191	30.0	tested_negative	tested_negative
	38.0	0.537	34.0	tested_positive	tested_positive
	27.1	1.441	57.0	tested_negative	tested_positive
5.0	30.1	0.398	59.0	tested_positive	tested_positive
5.0	25.8	0.587	51.0	tested_positive	tested_positive
	30.0	0.484	32.0	tested_positive	tested_negative
).0	45.8	0.551	31.0	tested_positive	tested_negative
	29.6	0.254	31.0	tested_positive	tested_negative
0	43.3	0.183	33.0	tested_negative	tested_negative

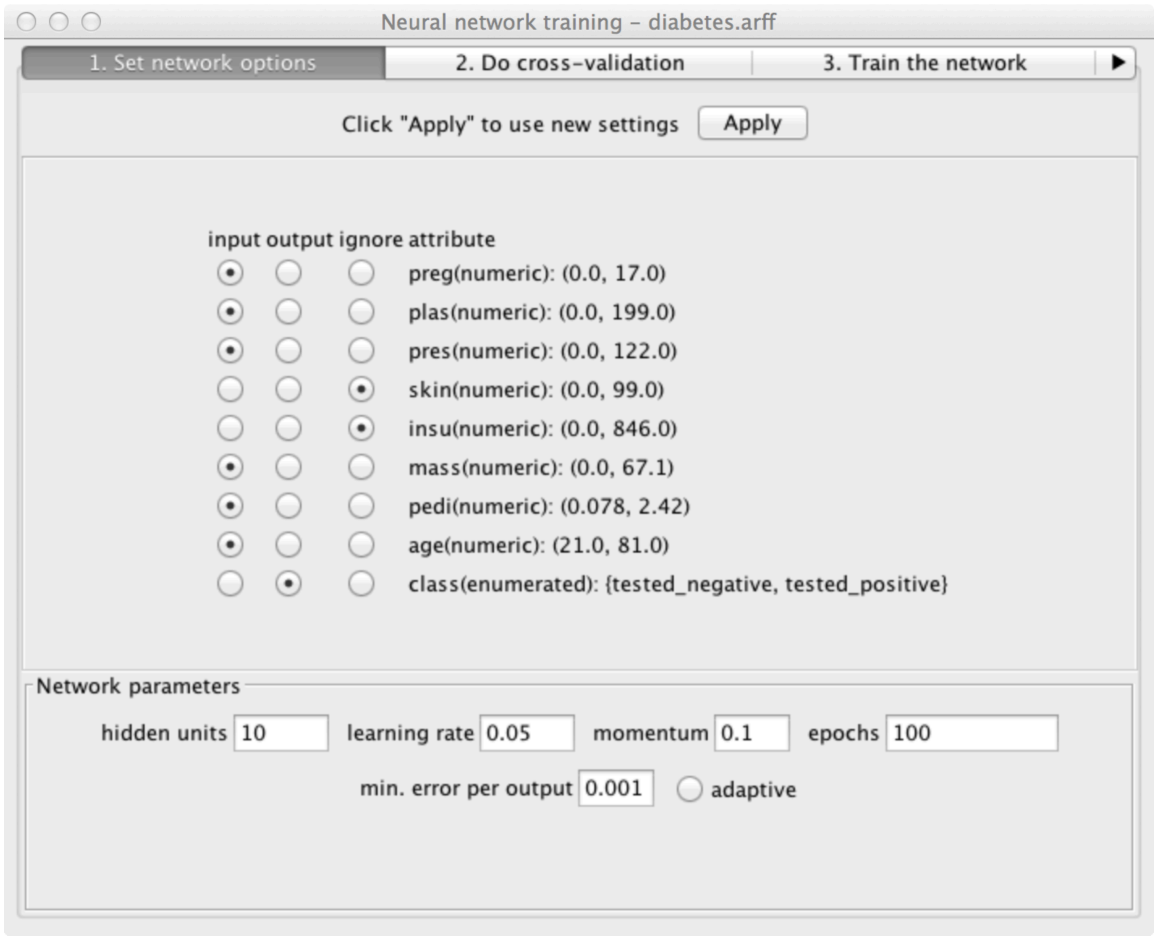
Each row is a training pattern. Each column is an attribute, except the last column, which is the predicted class. You can see that the predicted values are reasonably similar to the correct values (in the second last column).

This data can be copied to the clipboard (click on “copy output to clipboard”!), and pasted into, for example, and Excel spreadsheet, for graphing or further analysis.

Step 10

OK. Now go back to “View the trained network”. You will see that some of the input neurons have only weak connections. This indicates that these input attributes are not very important to the network. Note down the 2 least important input attributes.

Now go back to the “Set network options” tab, and click on “ignore” for the 2 selected attributes. Depending on your network, the panel might now look like this:



Important: *You now need to click the "Apply" button to confirm this change.*

Now go to the "Do cross-validation" panel and do another cross validation. With a bit of luck, you might get a better mean error rate (I got 77.725).

This is very good news: it means that by using only 6 of the 8 input values, we actually get a better prediction system. This makes the prediction system more accurate (or at least about the same accuracy), but we only have to collect 3/4 as much data to use the system.

Step 11

Now repeat the earlier process and fill in another table. With luck, you may find an even better combination.

When you have completed this workshop, submit your completed tables (Steps 4-6 and Step 11) to your tutor using the submission facility on Blackboard.