

COMP4702/COMP7703 - Machine Learning

Prac 8 – Single and Multilayer Perceptrons and Trajan

Aims:

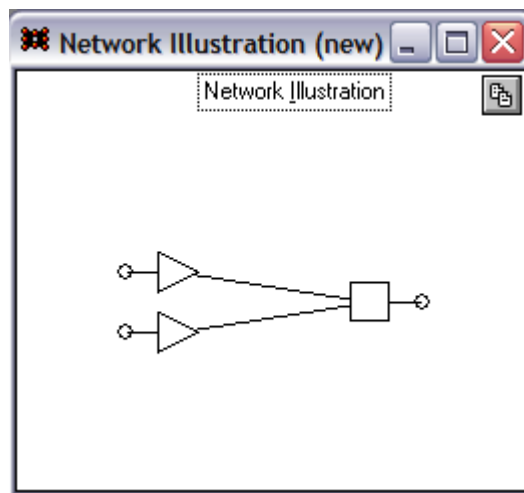
- To gain some experience in constructing single and multilayer perceptron networks and solving problems with them.
- To gain familiarity with the Trajan software package.
- To produce some assessable work for this subject.

Procedure:


Trajan is a fairly powerful neural networks software simulation package with many different features. In this prac you will begin to explore some of the basic functionality of Trajan. Try not to be distracted by all the features we won't be using!

Introduction to Trajan: the Boolean AND problem

- Log into windows and start up Trajan. You are automatically presented with *Open Data Set* dialog. Select *Cancel*. From the *File* menu, create a new dataset with 2 inputs and 1 output. You can then enter the AND training set manually (use the down arrow key to move to the next line!).
- From the File menu, create a new network. In the “create network” window, select a linear network. Make sure you set the number of inputs and outputs correctly. Click create and you should see a network illustration window appear.

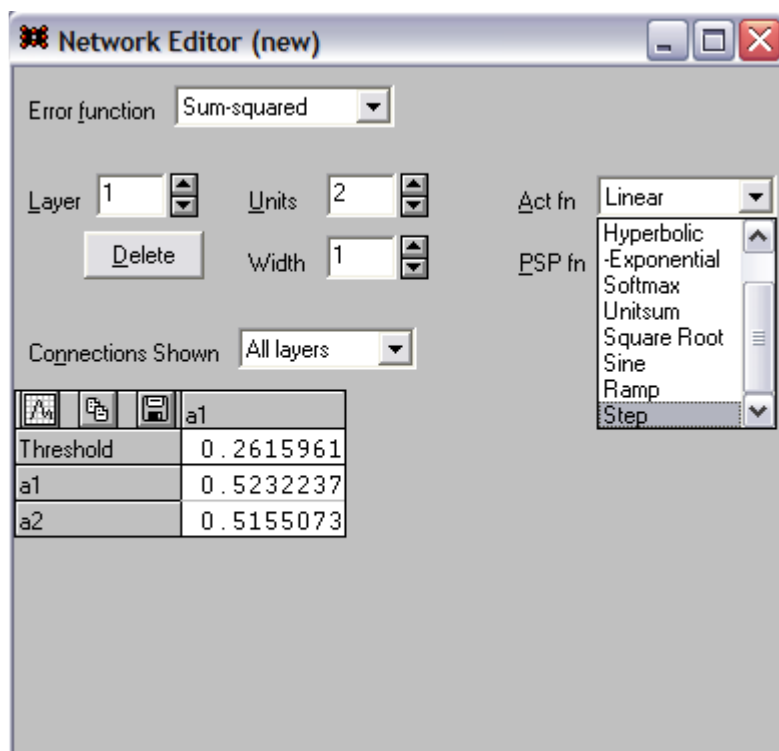


The basic mechanism for creating networks in Trajan manually is to create (adjust parameters, etc.) and then to commit networks that you create to the “network set editor”. It can be confusing unless you do things in the right order.

→ Trajan refers to a single-layer perceptron as a “Linear” network. For now we will train a single-layer perceptron in Trajan via the Delta rule¹. Select “Network...” from the Edit menu to open up the network editor window. This shows you the details of the network. Change from Layer “1” to “2” and you will see the initial (random) weight values for the connections from the two inputs to the neuron, together with the bias/threshold weight. To train via the delta rule, select (menu) Train -> Multi-layer perceptrons -> Backpropagation (NB: this is actually the delta rule when you have a single-layer net, as will be explained in lectures!). A Backpropagation window will appear. Click on the “graph” icon, , on the toolbar to create another window. Now click train in the Backpropagation window and observe what happens in the graph window.

- **Q1:** What value does the sum-squared error function level-off around? Try clicking on the Train button a few more times to make this clearer.

The reason error did not approach zero in the above is because we currently have a linear activation function for our neuron. Change “Act fn” to “Step”.



Now click “Reinitialise” in the Backpropagation window (to set the weights back to random values), clear the graph, and train again.

- **Q2:** Repeat this “Reinitialise” and “Train” process a few times and with the Step activation function and describe what you see. Does this agree with the lectures?

¹ The Delta rule can be thought of as backpropagation for a single-layer network (i.e. it does gradient descent on the weights wrt an appropriate error function. An earlier learning rule, the perceptron convergence rule, while it is very similar, does not do exactly this. If you are interested in reading more about this, see the “Neural Computing notes 2” document on the course web.

The Exclusive-OR Problem

- Enter the XOR dataset manually into Trajan. Create a multi-layer perceptron network with 2 inputs, 2 hidden units and 1 output. Now go to the Network Editor and check that you are using logistic (i.e sigmoidal between 0 and 1) activation functions on the hidden and output units.
- Train the network, experimenting with different learning rate (and momentum) parameters until you find a setting that approximately solves XOR. There are at least 3 ways to assess the network performance: by monitoring the error graph; by examining error on each data point (menu Statistics -> Case errors...) or by “running” the data inputs through the trained network and comparing the actual output with desired output (menu Run -> Dataset...).
 - **Q3:** What learning rate and momentum parameter values worked well for you? How many epochs did it take to solve XOR (and to what error value?).
 - **Q4:** Reinitialize the weights in your network and retrain with your parameter settings from Q3. Repeat this 10 times. Did your network always achieve the same error value? Suggest an explanation for this.
 - **Q5:** Try training network with 1 hidden unit to solve XOR, and then with 3 hidden units. Comment briefly on their performance.

Sonar data

Q6: In this question you just need to follow the experimental procedure below, and write a paragraph or two describing your observations. Also mention any difficulties you encountered, how well the network performed, etc.

One well-known early application of multilayer perceptrons involves distinguishing between rocks and mines on the ocean floor using sonar.

- Read about this application on p. 8-11 of the “Neural Computing notes 3” document on the course webpage (these notes have been previously used in this course).
- The sonar data set is the one used by Gorman and Sejnowski for distinguishing between rocks and metal cylinders described in Lecture 3. You can download this dataset (already in Trajan’s bdf format) from the course webpage. To open the file under the Trajan system click Open... in the drop down list under File. This brings up the Open Data Set dialog. Once you have opened the dataset you will be asked to select the basic or advanced version of the Intelligent Problem Solver. Just press Cancel.

You will see at the top of the Data Set Editor that there are 61 variables and 208 cases. Variables 1 – 60 have been set as inputs (black column headings) and Var 61 is the output as a nominal feature (i.e. Rock or Mine).

- One extra thing you will need to do is to split the data in the same way as was done by Gorman and Sejnowski, i.e. using half the data (104 cases) as the training set and the other 104 cases as a test set. This you can do by clicking on the microscroll button to reduce the 208 cases to 104 and you will find that the next window automatically increases from zero to 104. Click the microscroll button in the second window to reduce the 104 cases to 0. You will find that the third window automatically increases from zero to 104 (The contents of the first window indicate the number of training cases and the third window indicates the number of test cases.) If you now scroll down the data, you’ll find that those data cases from 105 onward are now coloured in blue. These are your test set.

- What kind of network should you use? Well, the aim here is to try and obtain a set of results similar to those obtained by Gorman and Sejnowski. So, you should look back at the Neural Computing notes 3/4 where you'll see that they tried various networks with differing numbers of hidden units. You should commence with a network having zero hidden units and then, depending on time available, try networks with the same numbers of hidden units as those used by Gorman and Sejnowski. One thing you need to note is that all their curves and tabulated results were based on averaging over 10 training runs from different initial sets of weight values. You won't have time to do this, but you should be able to get results that correspond in some measure to what was achieved by Gorman and Sejnowski.
- Now you can set up your network. Remember that you are trying to obtain results like those of Gorman and Sejnowski who used the mean-squared error function. Click on Network from the File-New pull down menu to display the Create Network dialog. Set the network type, number of inputs, number of outputs and the number of layers.

In setting up the problem, the output has been assigned the nominal values "Rock" and "Mine" but the network actually produces numerical values. (From which the decision on Rock and Mine are made). Output values close to 1 are given the value "Rock" and those to 0 are given the value "Mine". Values further away from 0 and 1 are treated as unclassifiable and given the value "unknown". The Pre-post-processing editor sets the threshold between unclassifiable values and those that can be classified available from the Edit pull-down menu. There are two threshold fields to set, Accept and Reject. If the activation is above the accept threshold, the input example is classified as a Rock and if it is below the reject threshold, it is classified as a mine.

- You are now ready to train a network and to do this, will need to follow a procedure similar to the one you used at the very beginning. On this occasion, however, the output variables have already been defined, so you don't need to worry about that. Click on Classification in the Statistic pull down menu after training. You can then see how well the network performed on the test set after the training. The blue columns show the test cases while the black is the training cases.
- Try different values for both the threshold and try to obtain results similar to what Gorman and Sejnowski got. Close the Pre/Post Processing after setting the threshold values.

Over-training a network

Q7: In this question you just need to follow the experimental procedure below, and write a paragraph or two describing your observations. Also mention any difficulties you encountered, how well the network performed, etc.

During training, the network is trained to minimize the error on the training set. But how do we know how much training to give the network in order for the network to generalise well and when should we stop training to prevent overfitting? The most effective method to ensure the network is able to generalize well is to hold back some of the data and not use it for training but instead, to test the generalization performance of the network.

To demonstrate this we are going to use the Pima Indians Diabetes dataset. This dataset

- indicates whether the patient shows signs of diabetes according to World Health Organization criteria
- has 768 samples for training & verifying
- has 8 input attributes

- has 2 output classes (either 0 or 1)
- Go to the course web page and download the “Pima Indians Diabetes” dataset (in plain text format). To open the file under Trajan system click Open... in the drop down list under File. This brings up the Open Data Set dialog. Change the file type to Delimited Text Files. The Trajan text importer will import this file for you, but make sure that the “First row gives...” and “First column gives...” boxes are unchecked. Trajan will ask if you want to save the file in bdf – just hit cancel. Then Trajan will want to use it’s Intelligent Problem Solver – again just say no!
- You should have a total of 768 cases in this dataset with 8 inputs and 1 output (9th variable) in each case. 400 cases will be used for training and the remaining 368 will be used for verifying the performance of the network. To arrange for this click the first microscroll button (first the right of the cases field) to 400 and click the second microscroll button (second the right of the cases field) to 368. The cases in black are the training data and the red ones are the verification data. Set the network type to be multilayer perceptron, error function to entropy (single) and the number of hidden units to 5. Use back propagation to train the network and set the number of epochs to 100, learning rate to 0.1, momentum to 0.1 and unselect the shuffle cases option. Use the training graph to observe the results. There will be two curves appearing in the graph, the blue one shows the error on the training data and the red one shows the verification data. Observe the two curves carefully. Click Train once. You should observe that the initial performance of the training and verification sets are approximately the same.
- Click Train a few more times. You should see that the verification error starts to rise (the training and verification curves start to split apart). This is an indication that the network is starting to overfit and that this is the point where training should cease.