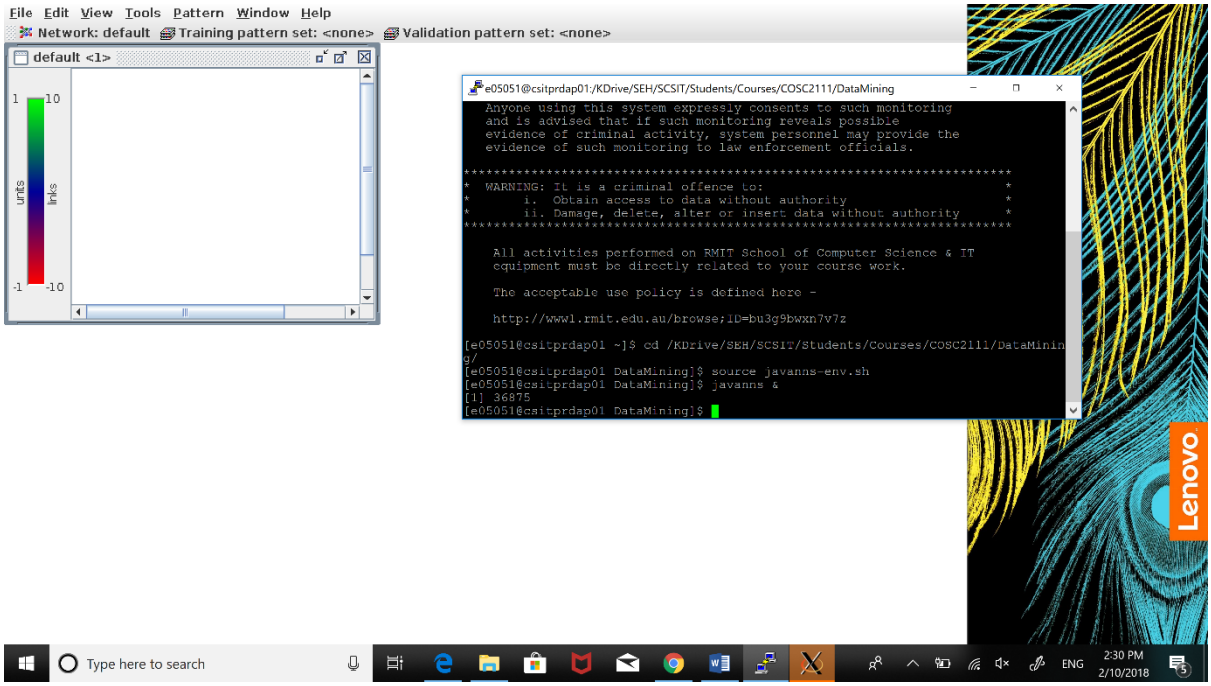
Solution to Laboratory Week 8.

1. Log into titan.csit.rmit.edu.au with an Xwindows connection.

2. cd /KDrive/SEH/SCSIT/Students/Courses/COSC2111/DataMining/

3. source javanns-env.sh

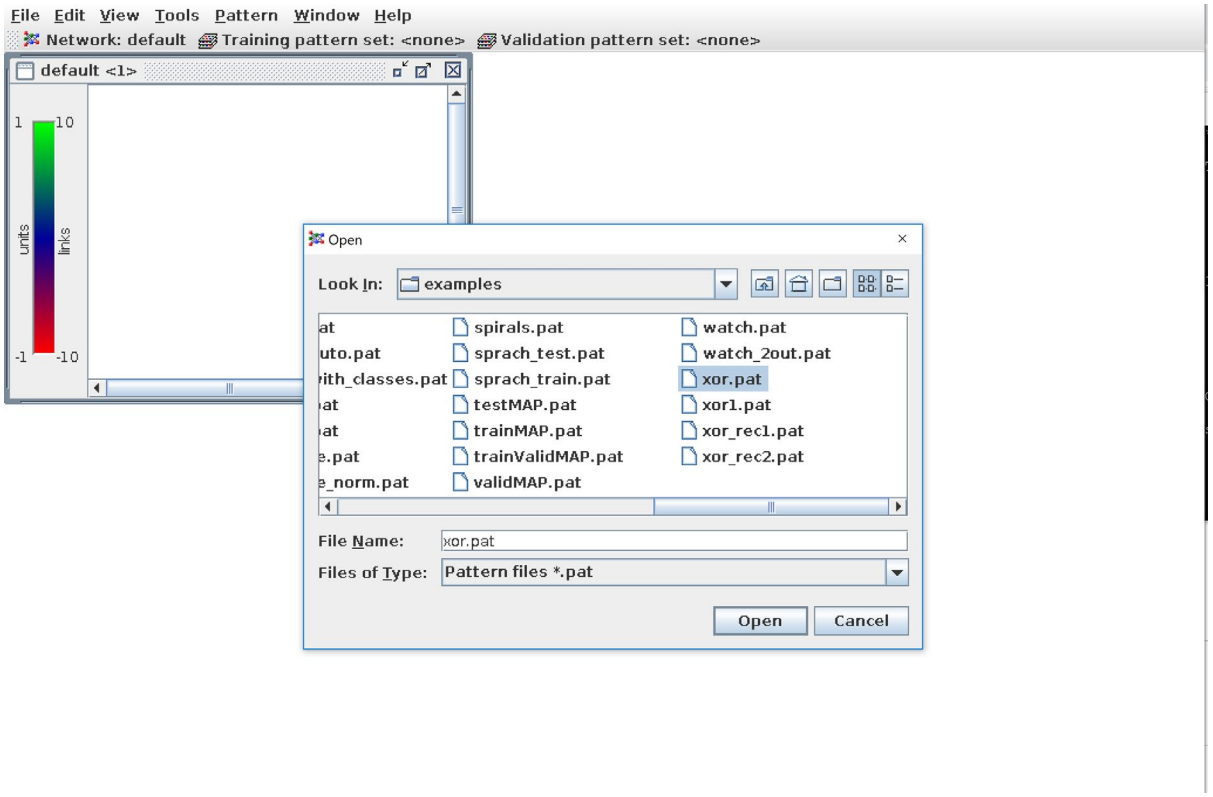
4. javanns &



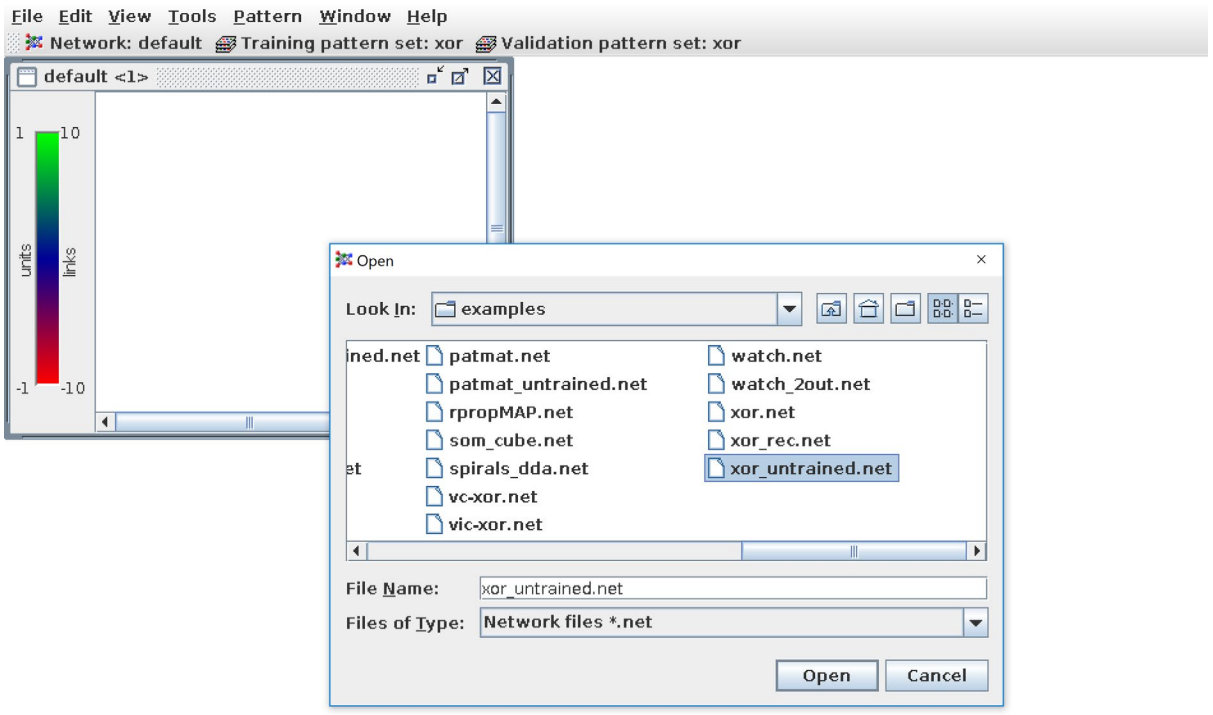
In this exercise we train a neural network to learn the XOR function.

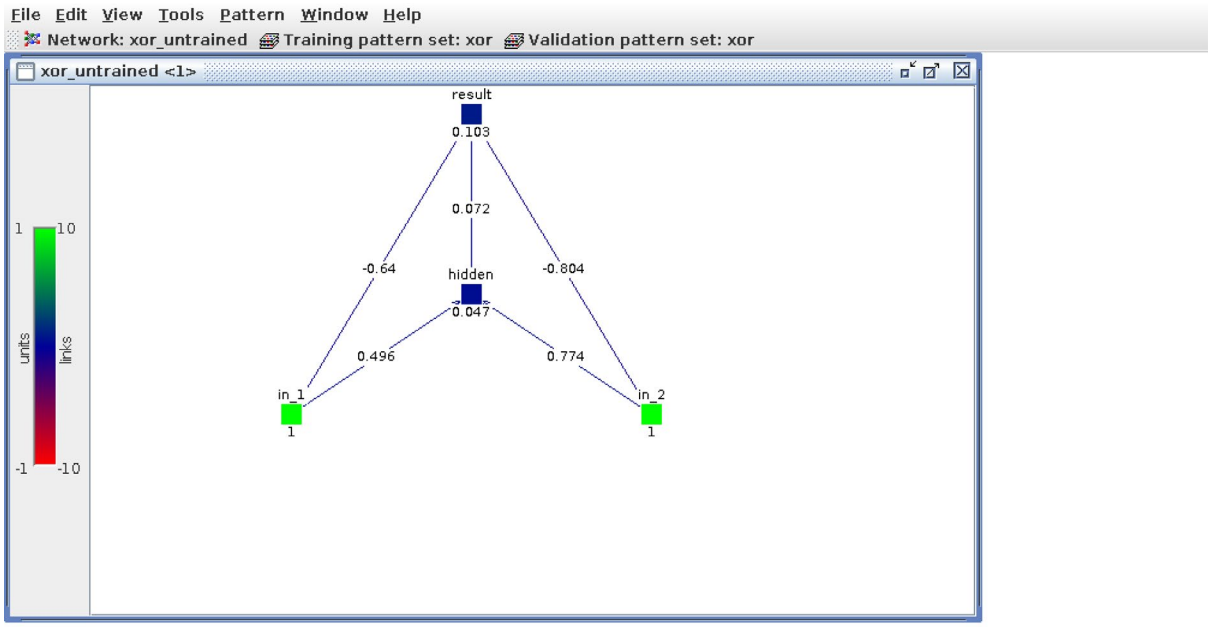
1. Start JavaNNS

2. Load the file containing the training data, xor.pat. This file can be found in subdirectory, named examples, of the JavaNNS home directory. File --> Open xor.pat --> Select the examples folder, and then open xor.pat.

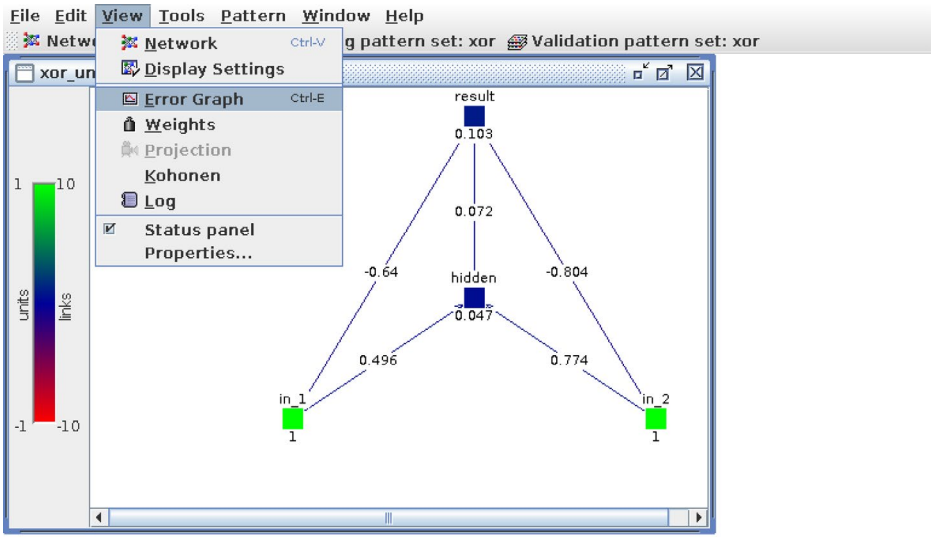


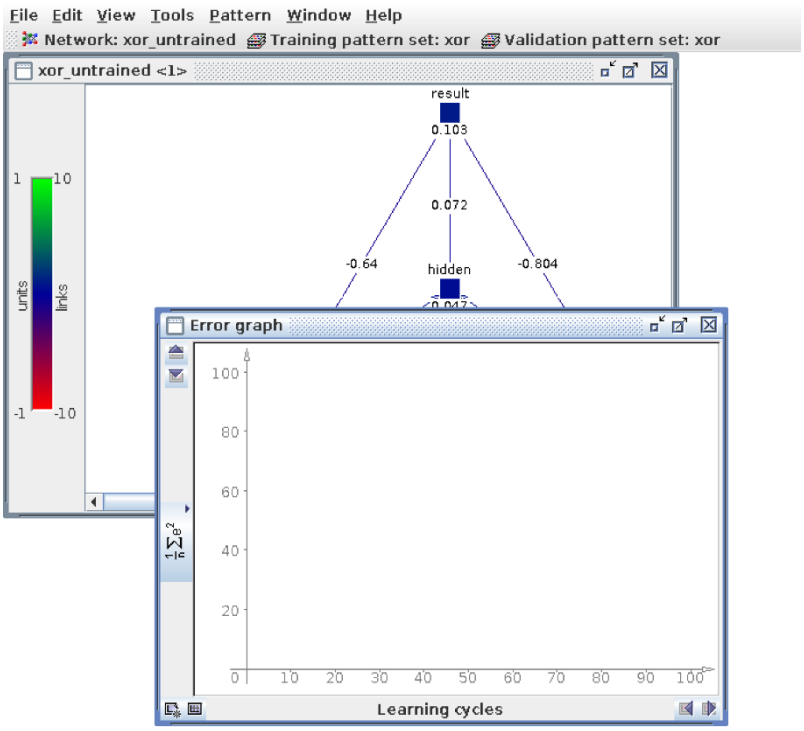
3. Now load a pre-designed network from the file xor untrained.net. You may need to resize the window to have a full view of the network. File --> Open --> Select the examples folder, and then open xor untrained.net.



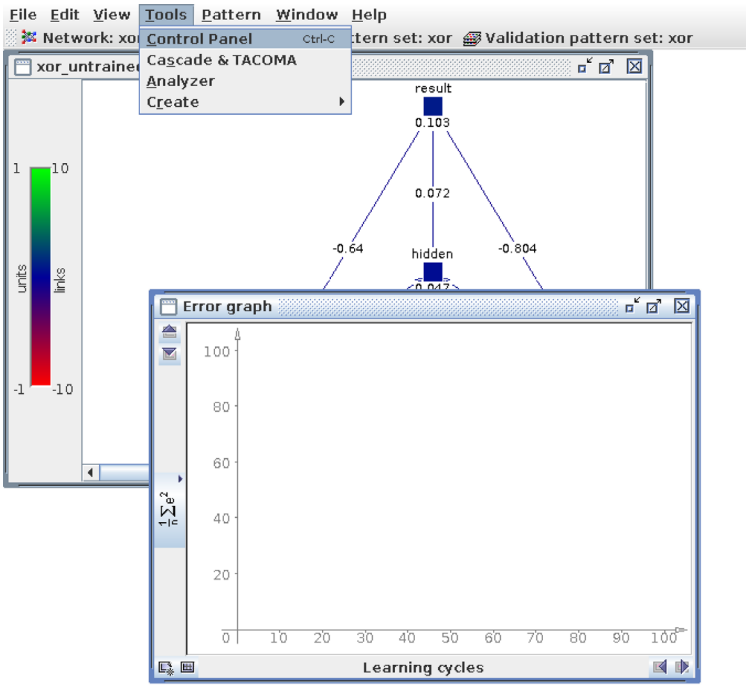


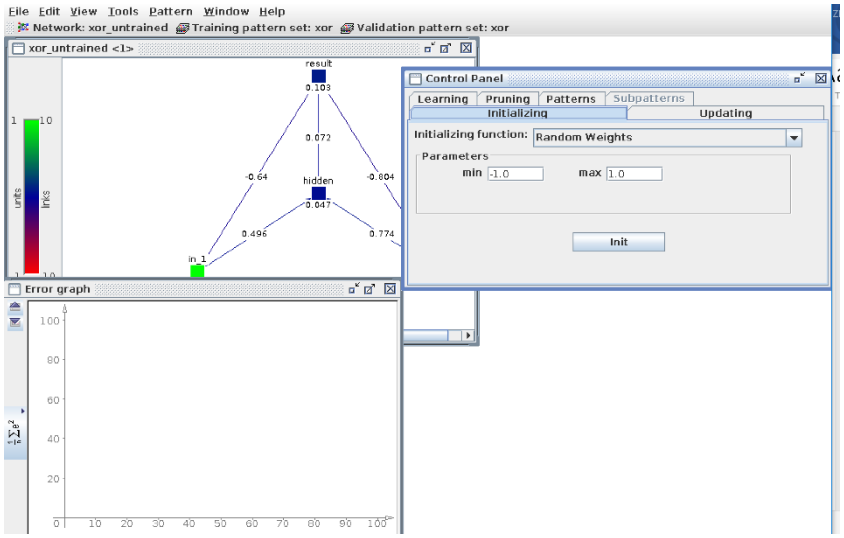
4. Now get the error graph. Go to View --> Error Graph (or press Ctrl+E). You can view the training errors in this window. You may want to adjust the range of the axes by clicking on the arrows near the axes.



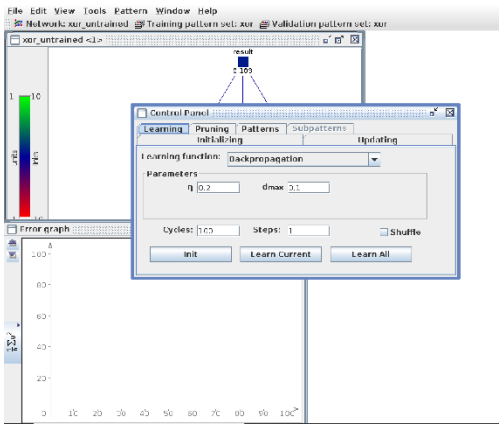


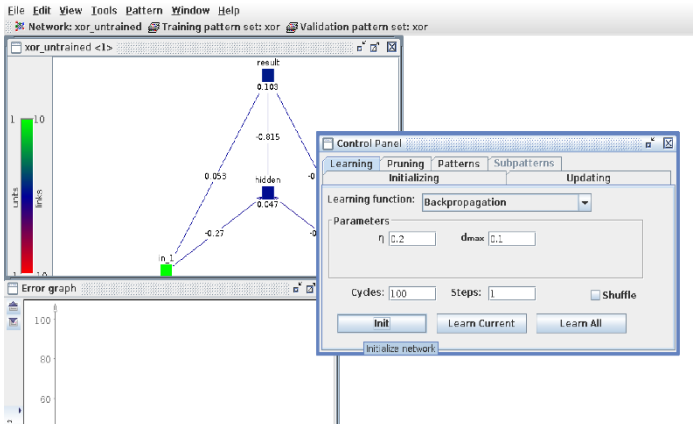
5. Go to Tools --> Control Panel. This is the main interface for training the network. You can also invoke this by pressing Ctrl+C whenever JavaNNS is in focus. We will need to use the settings in the Learning tab to do the training.

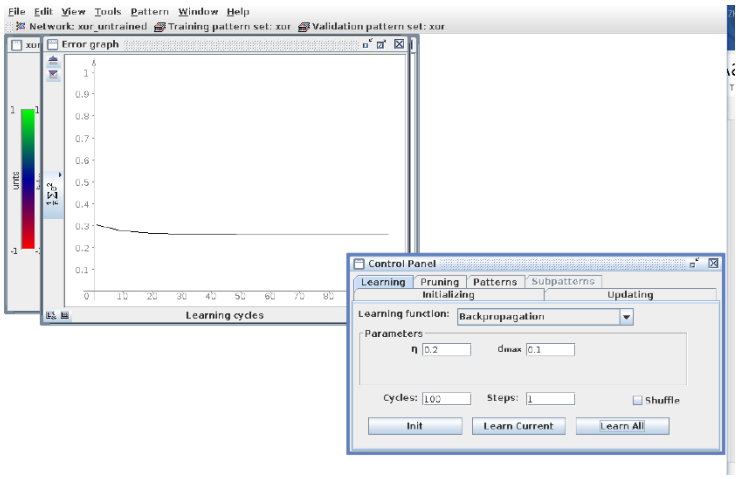


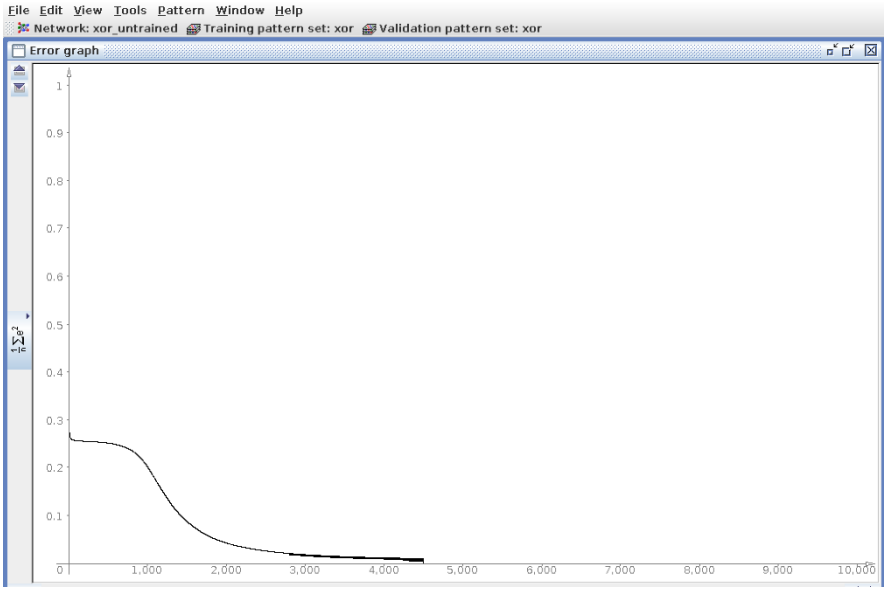


6. Use the default settings in the control panel. Click Learning, then init then Learn all. What can you see in the Error Graph? Keep clicking Learn All until the error gets close to zero. You may want to increase the number of cycles.

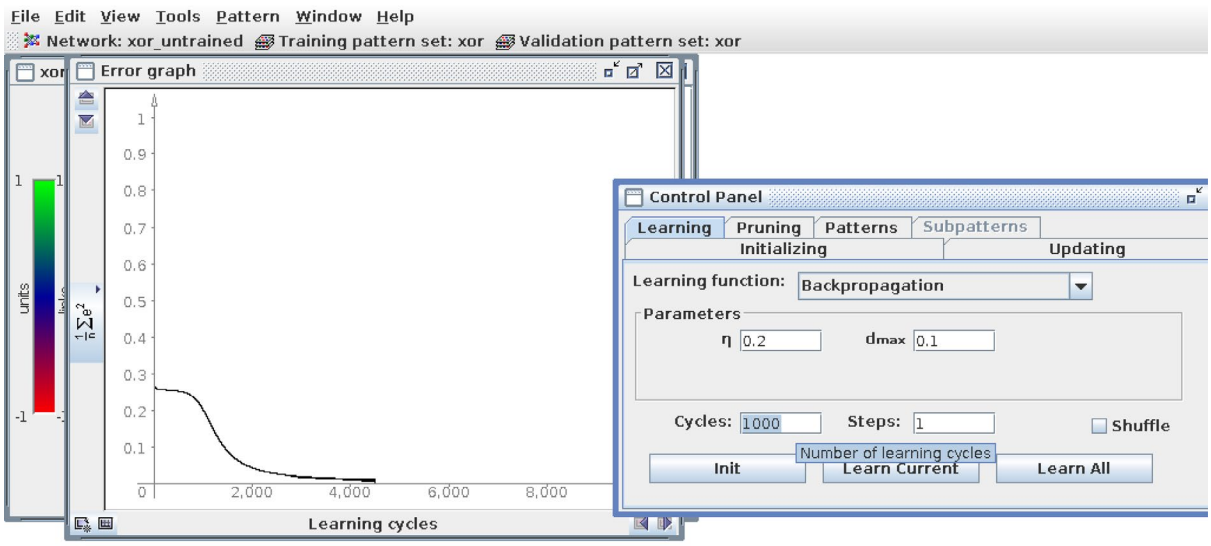






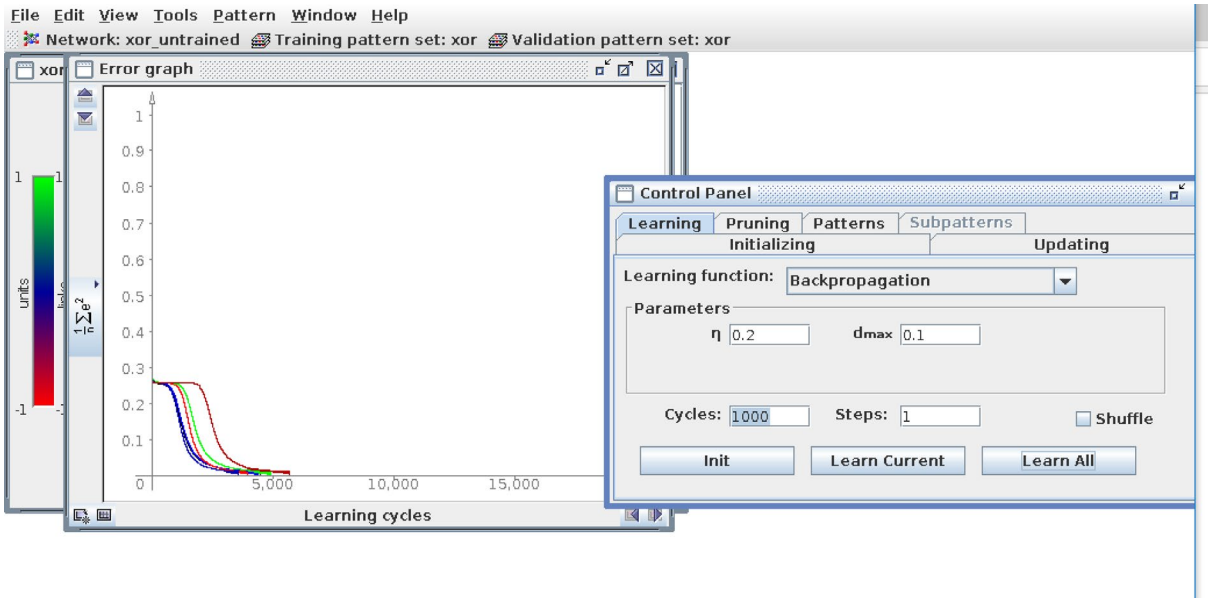


Error graph showing that by keep clicking Learn All the error gets close to zero.

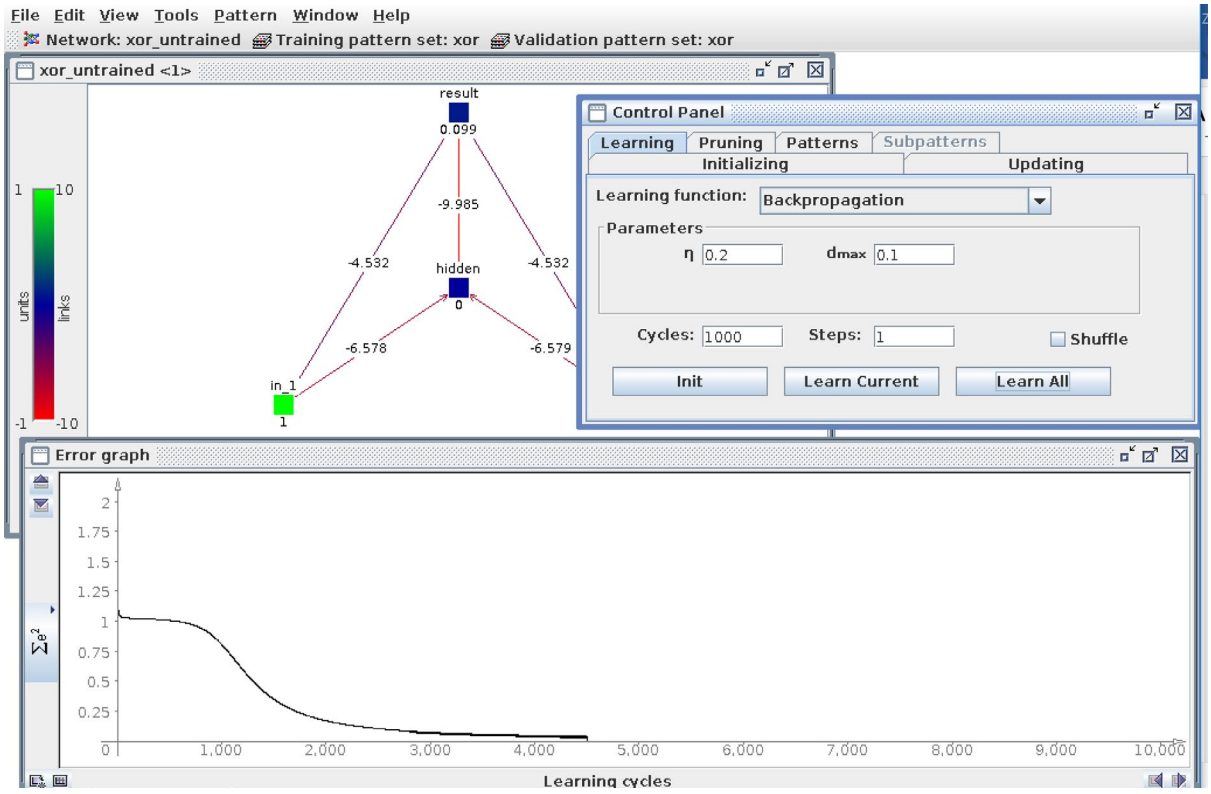


Snapshot of showing how to increase number of cycles from 100 to 1000.

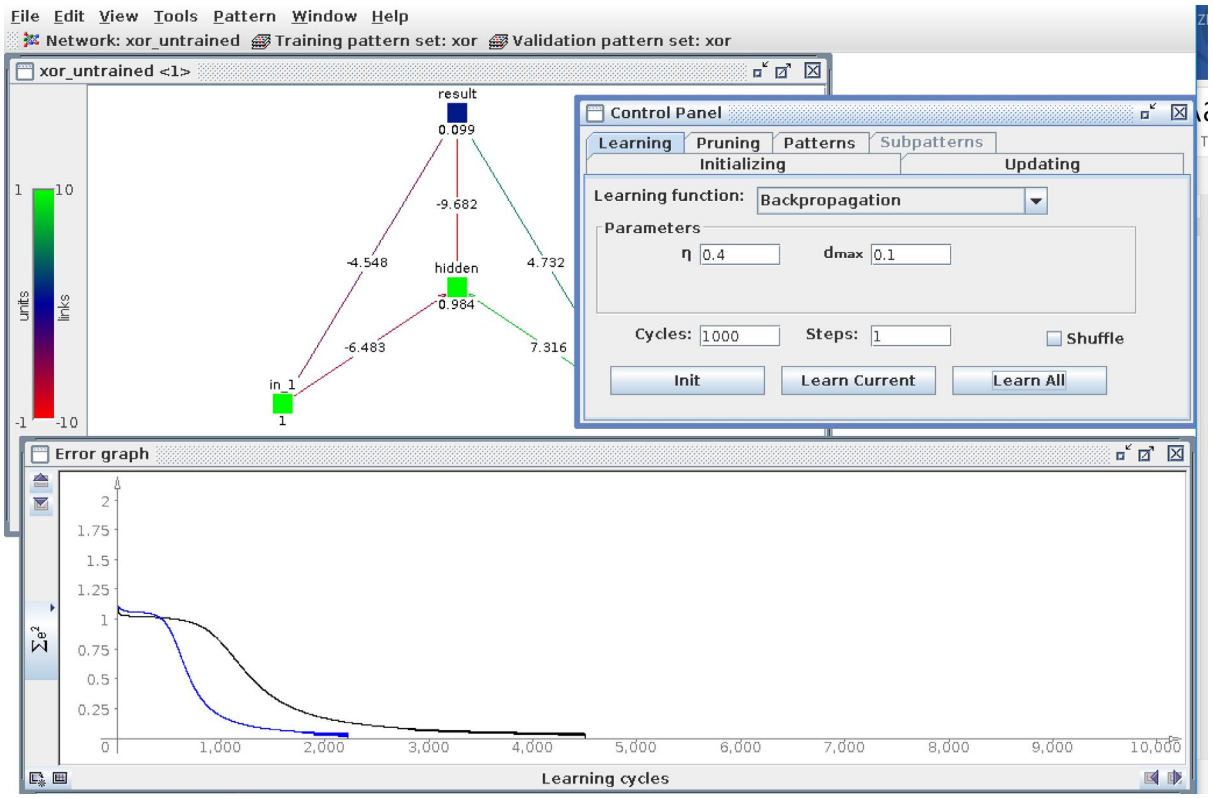
7. Repeat the init and train procedure several times. What patterns do you notice in the error graphs?



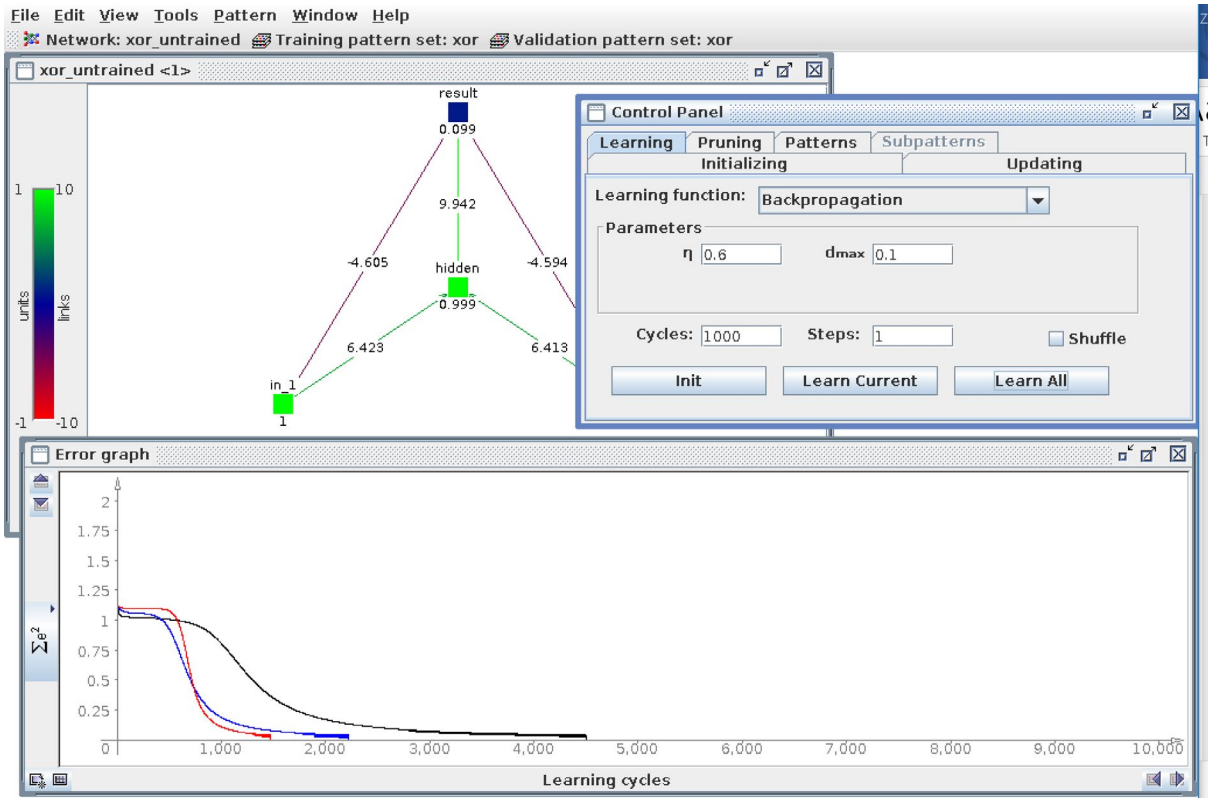
8. Try different different learning rates. Can you find any relationship between learning rate and error trajectory?



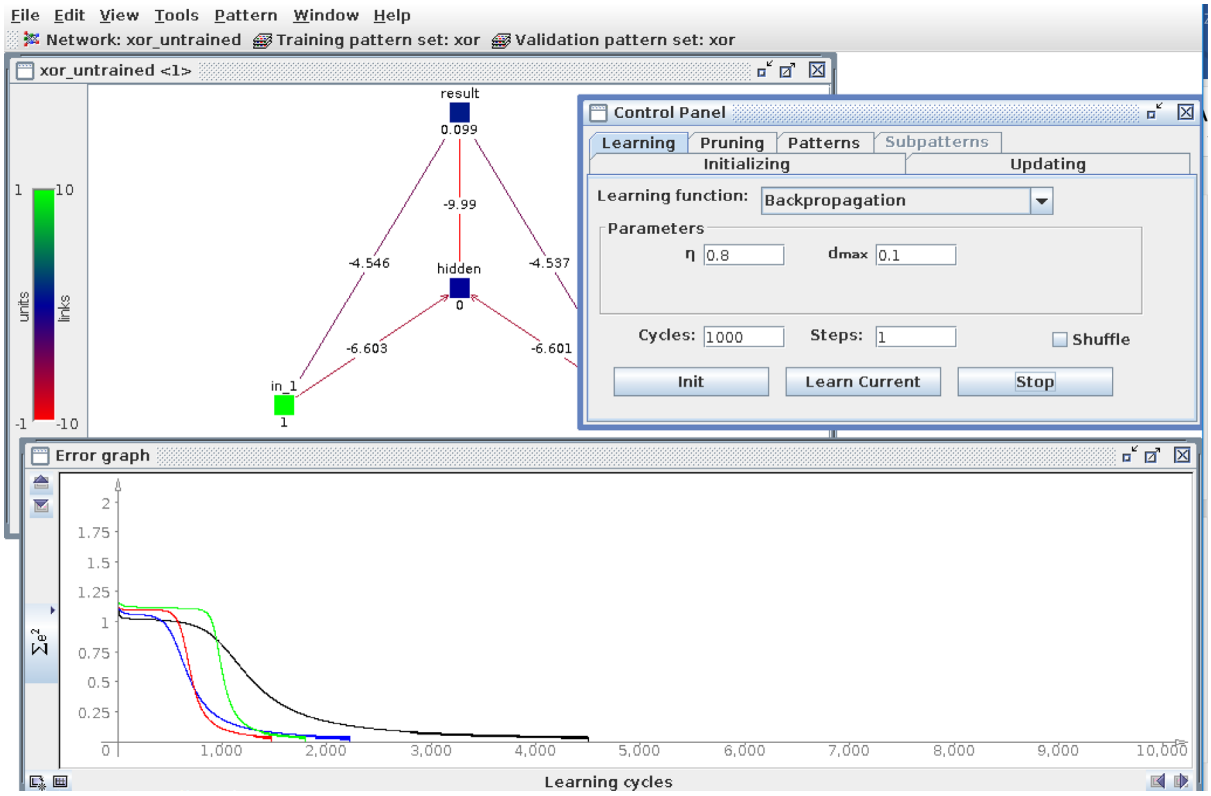
Error graph for learning rate of 0.2.



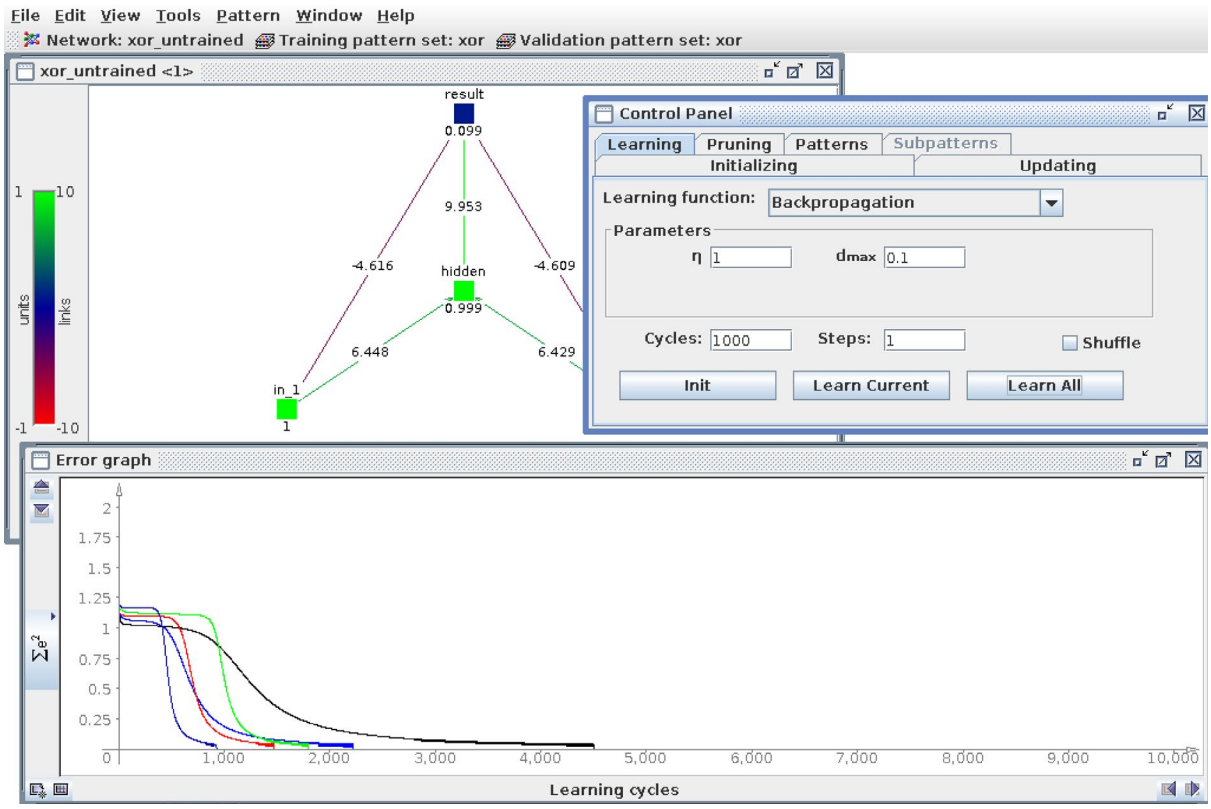
Blue line on error graph when learning rate is 0.4.



Red line on error graph for learning rate of 0.6 that only takes 2 steps to get to zero.

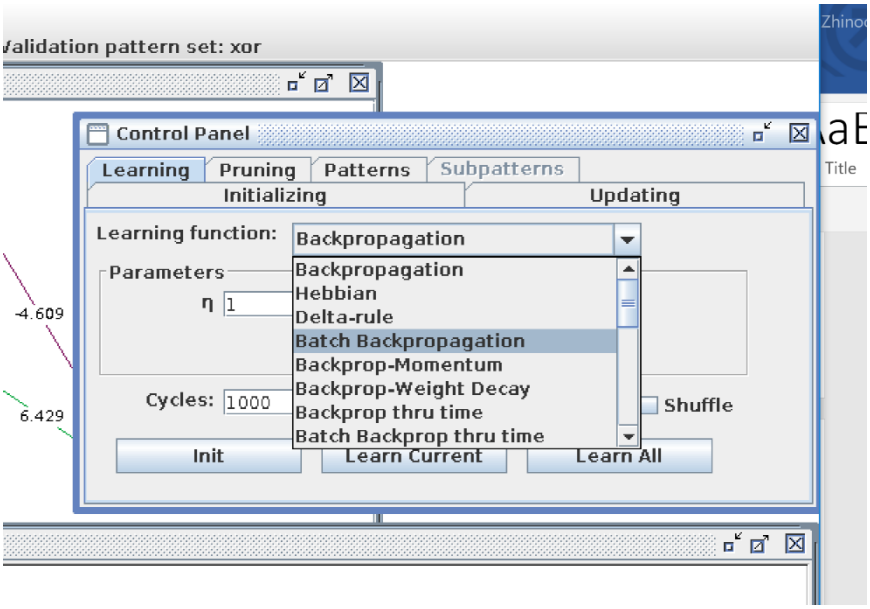


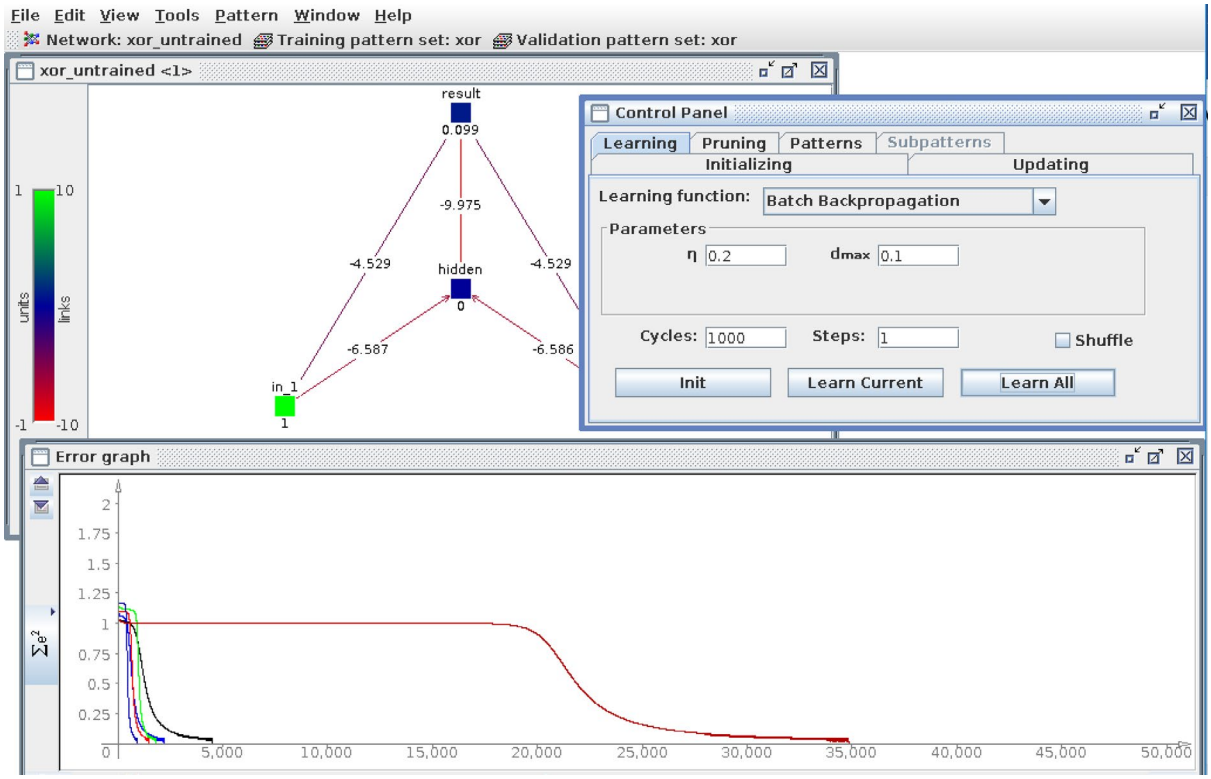
Green line where learning rate is 0.8 and only in 2 steps gets zero.



Dark blue line where learning rate is 1 and only takes 1 step to get to zero.

9. Try Batch Backpropagation as the learning function. How is it different?





The dark red line is the error of Batch Backpropagation.

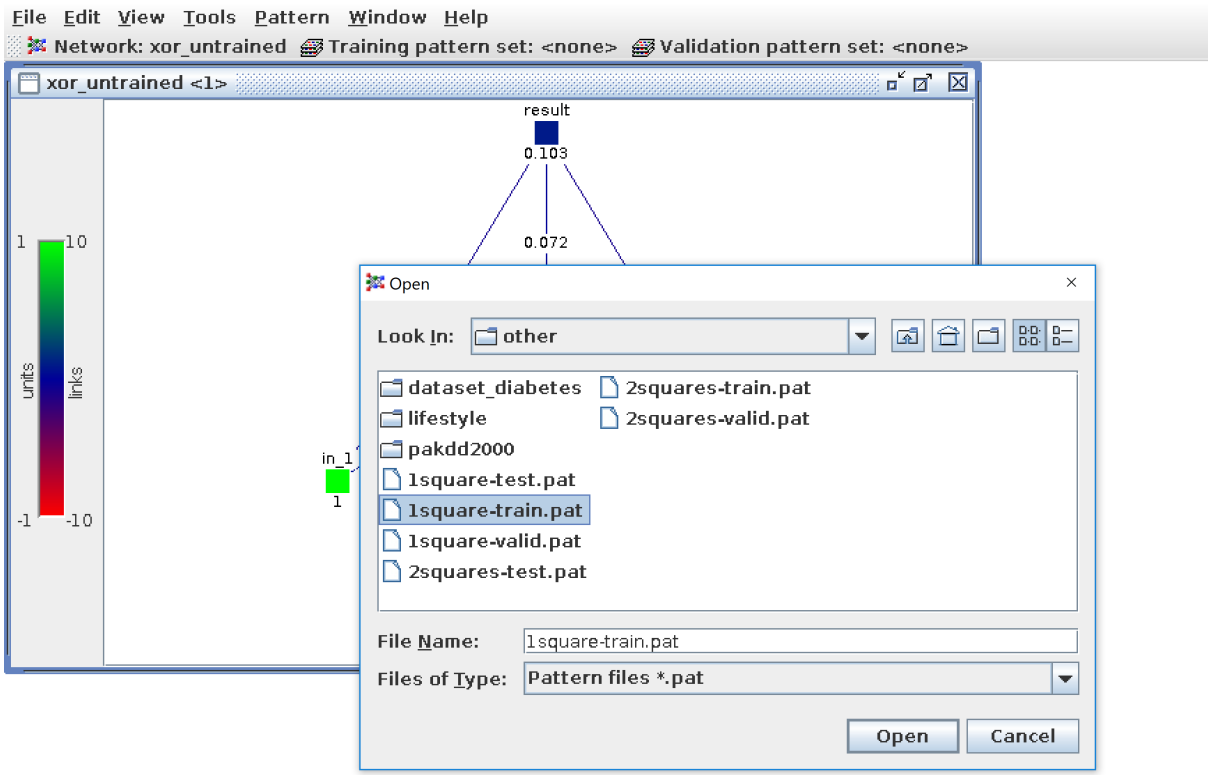
As it can be seen the error of Batch Backpropagation is getting close to zero after 35 cycles.

10. Try Backprop-Momentum as the learning function?

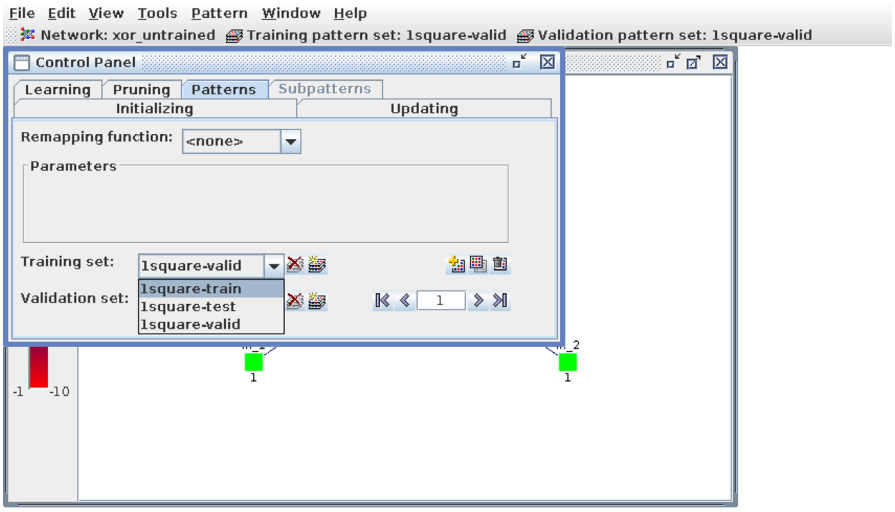
In this exercise we train a classifier with training, validation, and test sets.

1. Restart JavaNNS and reload xor untrained.net. This has 2 inputs and one output, which suits the 1square data.

2. Load the 3 files from /KDrive/SEH/SCSIT/Students/Courses/COSC2111/DataMining/data/other/: 1square-train.pat 1square-test.pat 1square-valid.pat

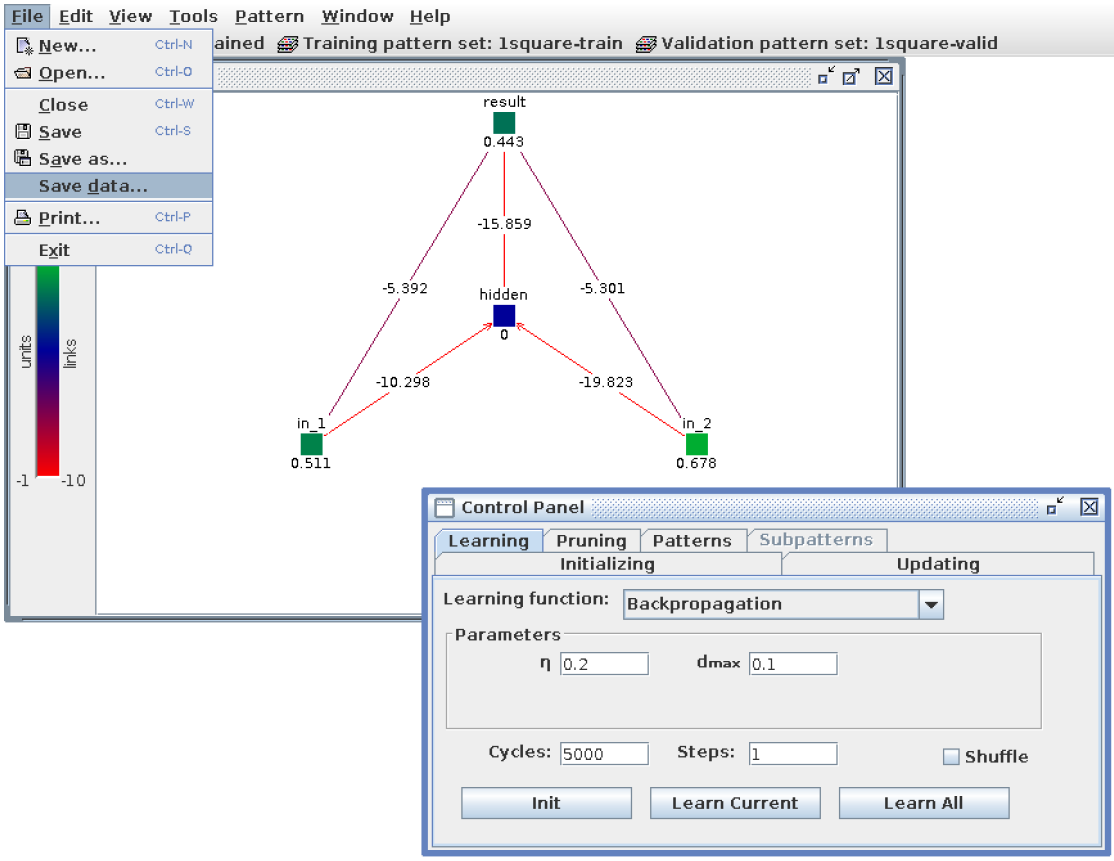


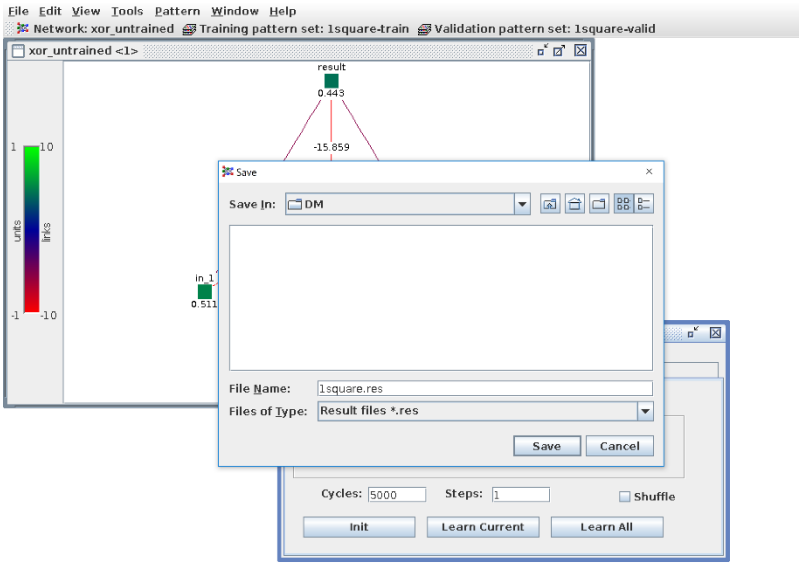
3. In Control Panel --> Patterns set the training file to 1square-train.pat and the validation file to 1square-valid.pat

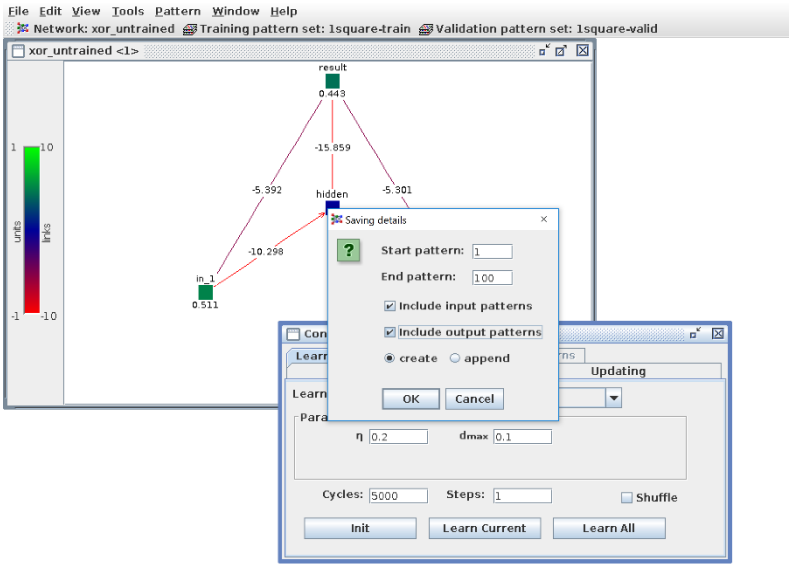


4. Train the network for 5000 cycles with Init and Learn All.

5. Generate a result file, 1square.res, with Save Data. Include the input and output patterns. [You will need to specify a folder for which you have write access.]

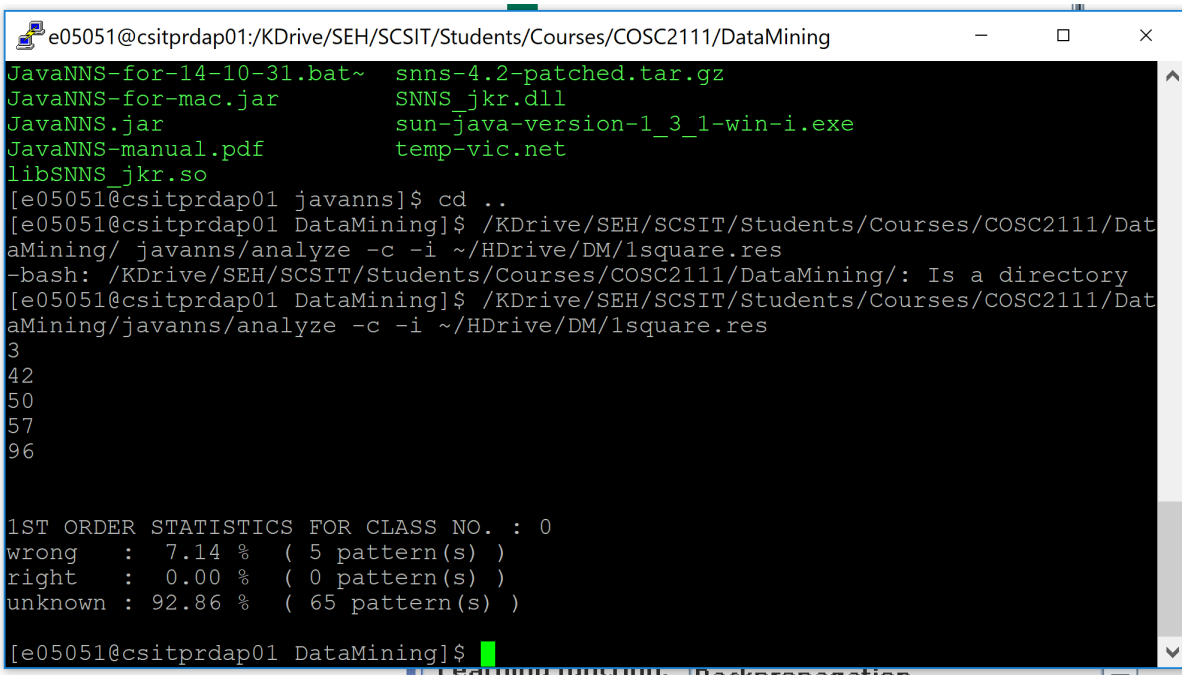






6. In a shell window, get the classiification rate with the default analyse strategy with (Put the following two lines on one command line)
/KDrive/SEH/SCSIT/Students/Courses/COSC2111/DataMining/ javanns/analyze -c -i 1square.res

Note: If you have used source javanns-env.sh you can use analyze -c -i 1square.res

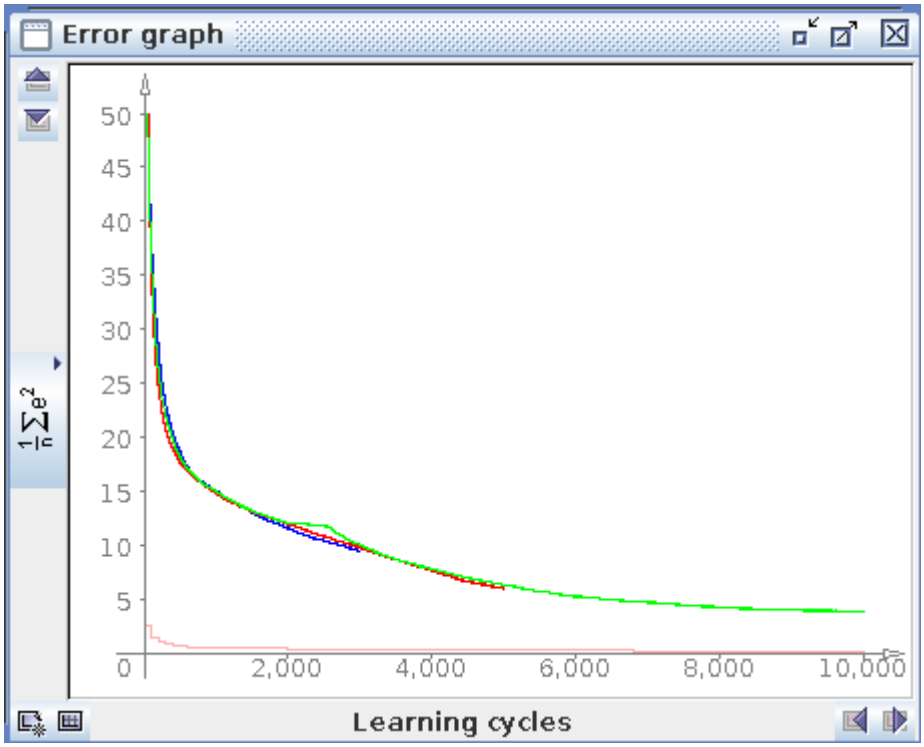


For the 1square task: Train=1000; Valid=100; Test=500; analyse 500050

Archtecture	Cycles	Train Classfn Accuracy	Test Classfn Accuracy
Xor 2-2-1	5000	60.6%	61.8%
2-10-1	2000	92.4	92.4
2-20-1	10000	98.8	99.2

7. What do you conclude? Is more training needed? More hidden nodes? A different interpretation strategy for analyze?

Two hidden nodes is not enough. More epochs doesn't help. Ten hidden nodes is a considerable improvement. Twenty hidden nodes is excellent, although I'm surprised that it took so many more epochs than ten hidden nodes. The following graph is for 3 runs with 20 hidden nodes.



8. Write a script to convert the iris.arff to suitable training, validation and test files for JavaNNS. How does the JavaNNS accuracy compare with the Weka classifiers?

Solution is on the KDrive