Simple 2-layer Networks

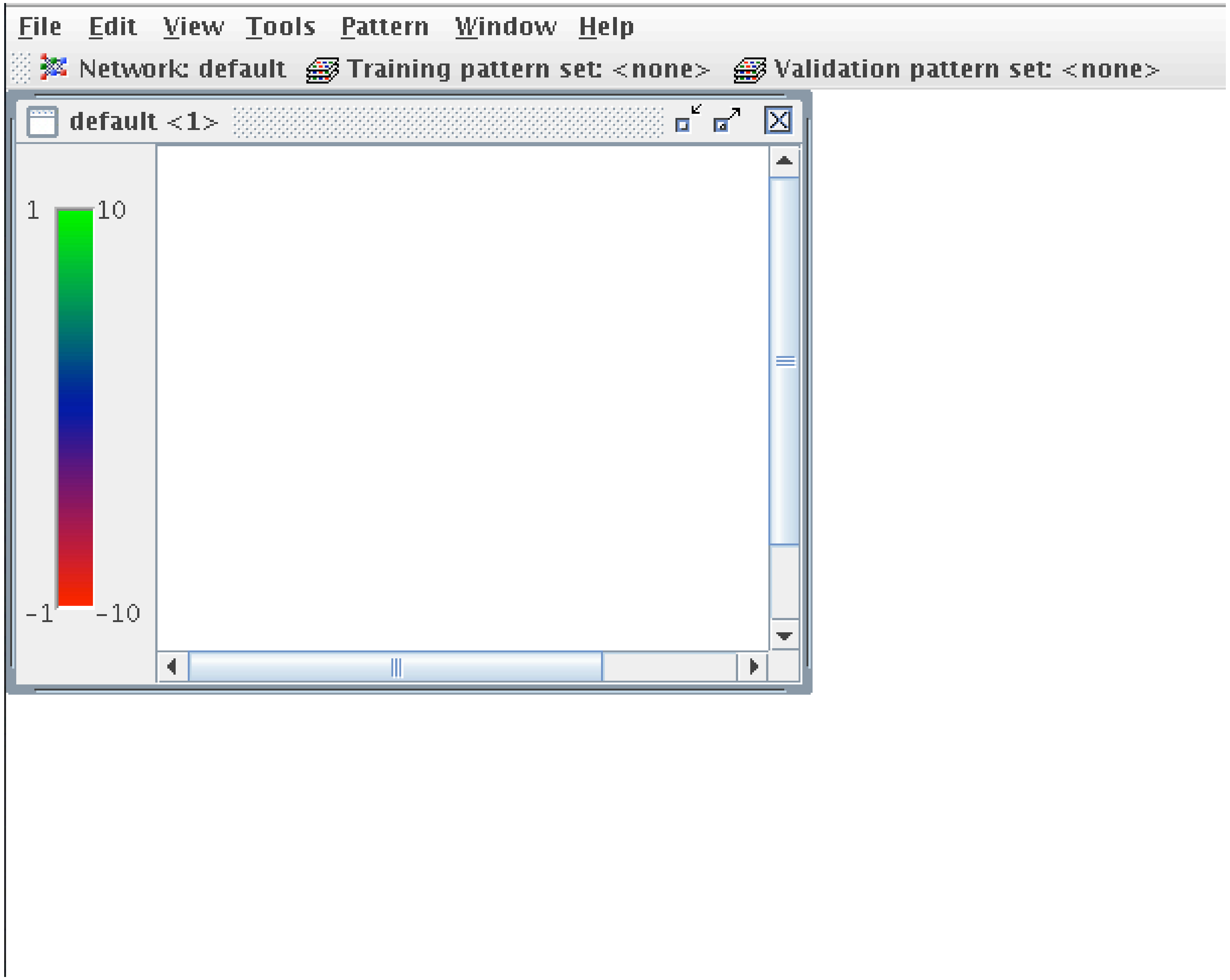
Connectionism Lab 1

- JavaNNS developed at Tübingen University, puts a Java front end on SNNS, the Stuttgarter Neural Network Simulator

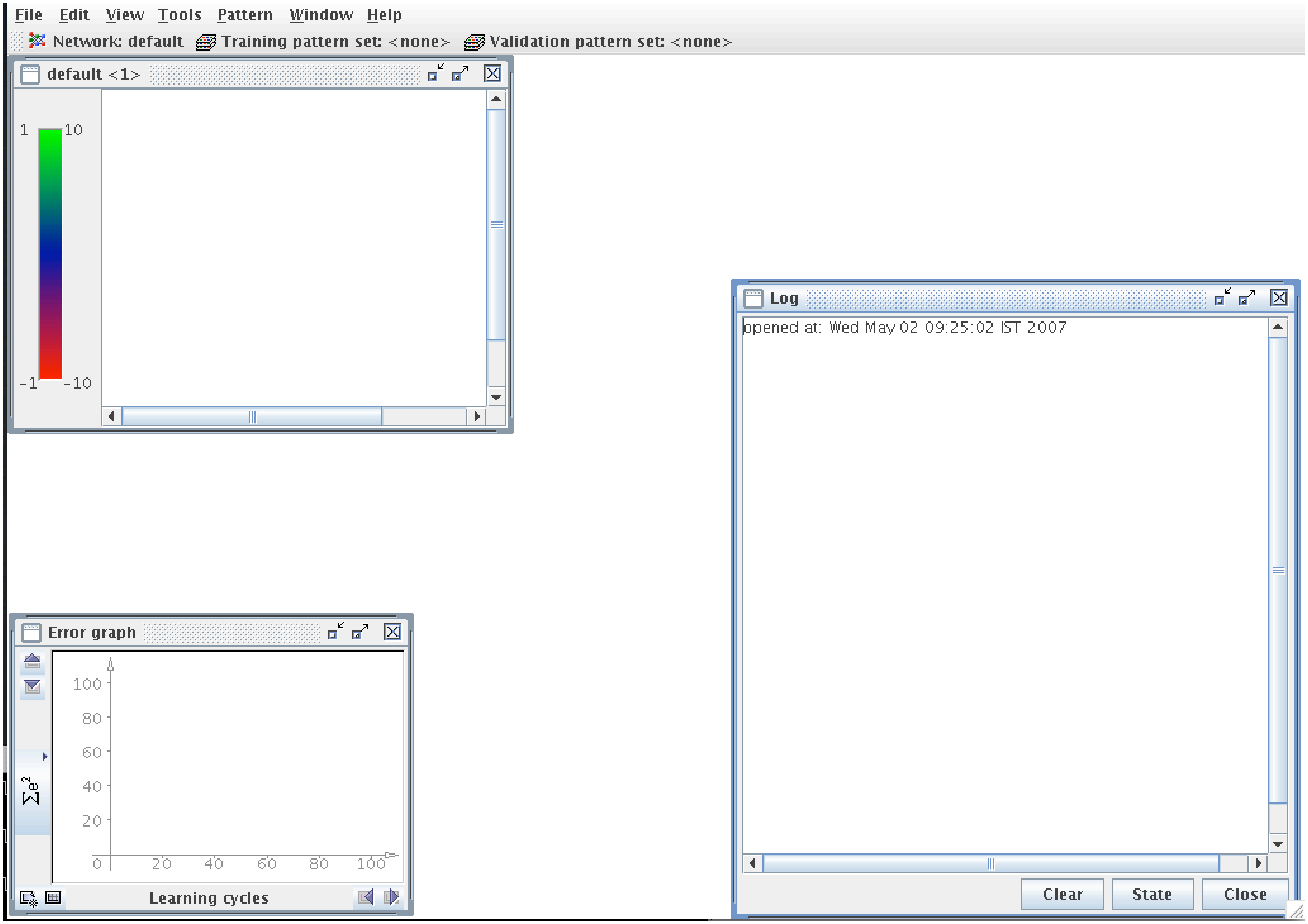
- Obtainable for free from http://www-ra.informatik.uni-tuebingen.de/software/JavaNNS/welcome_e.html

- From a commmand line, type ‘java -jar JavaNNS.jar’.

JavaNNS start screen

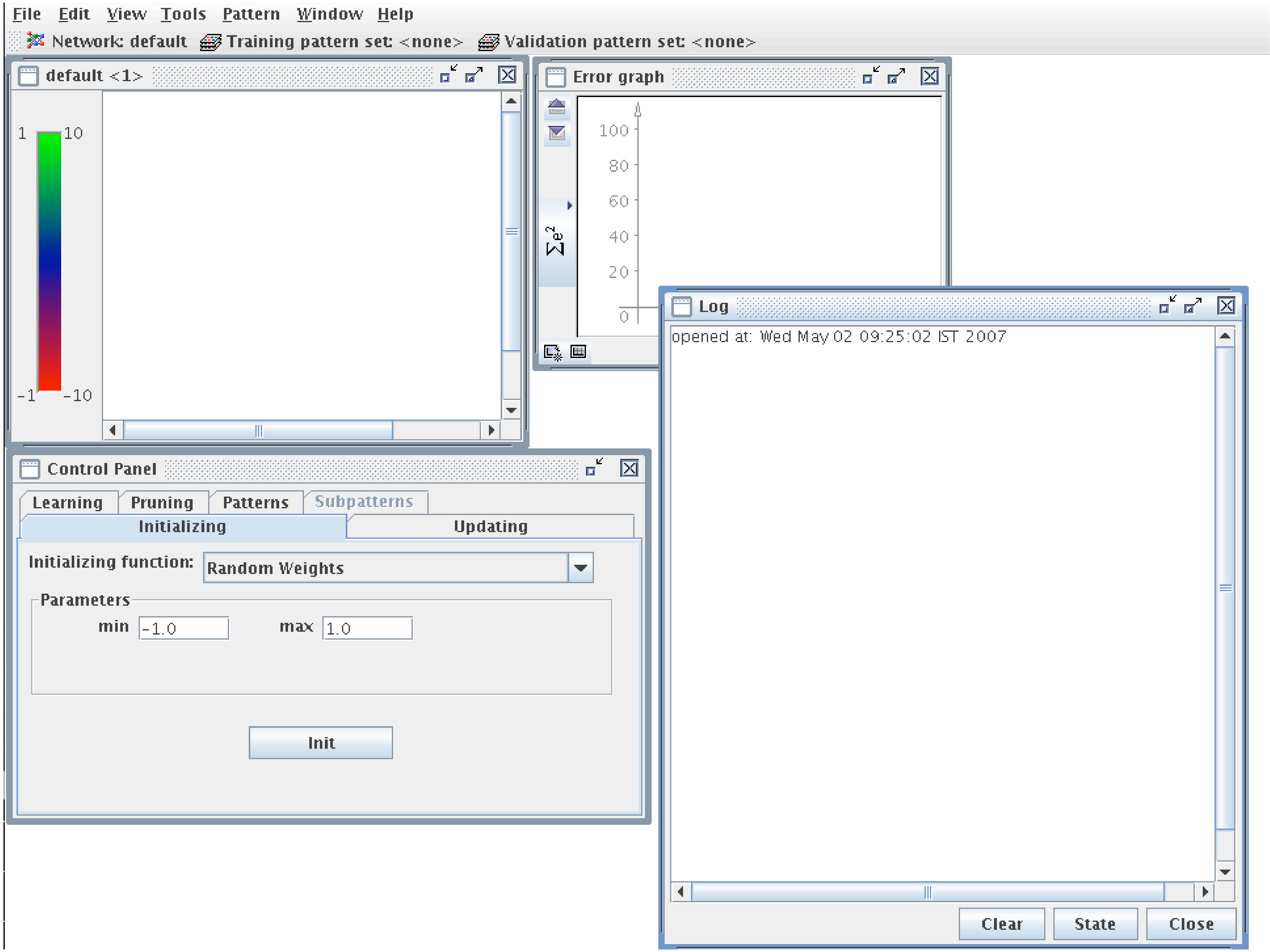

File Edit View Tools Pattern Window Help
Network: default
Training pattern set: <none>
Validation pattern set: <none>
default <1>
1
10
-1
-10

Add error panel and log panel from the ‘view’ menu



opened at: Wed May 02 09:25:02 IST 2007

Add control panel from the 'tools' menu. Rearrange to suit yourself



Network: default Training pattern set: <none> Validation pattern set: <none>

Initializing function: Random Weights

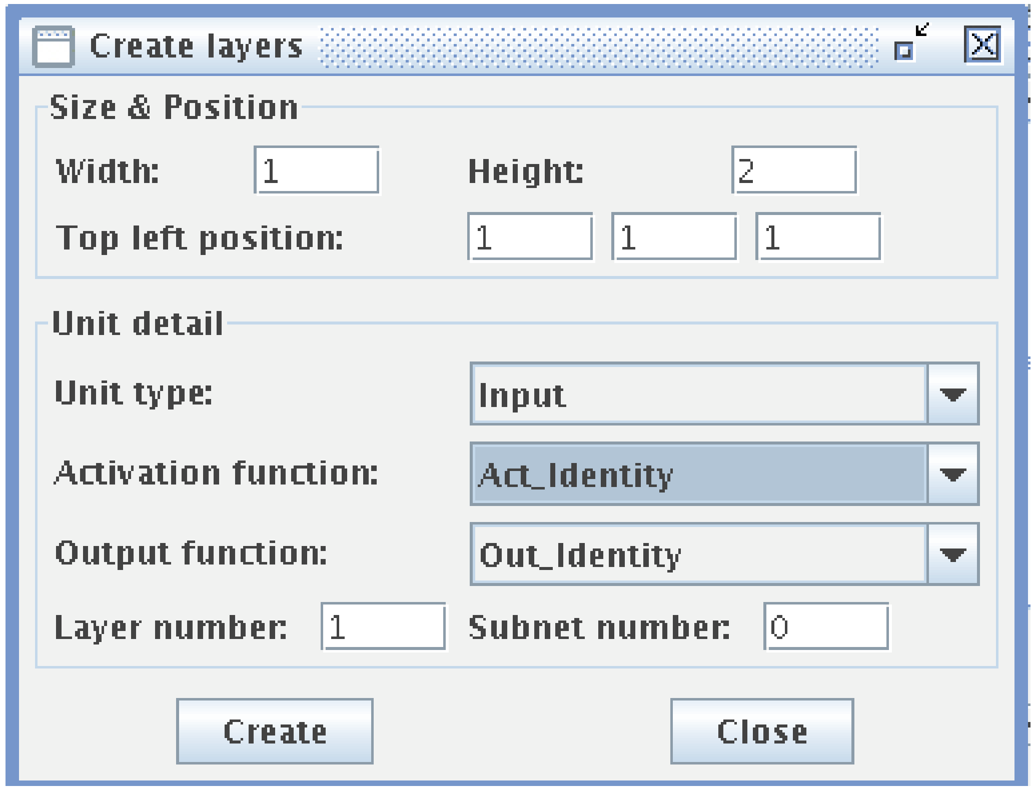
Parameters

min -1.0 max 1.0

opened at: Wed May 02 09:25:02 IST 2007

Goal: generate a 2 x 1 network, with linear units at both input and output

[Tools] > [Create] > [Layers]



Size & Position

Width: 1 Height: 2

Top left position: 1 1 1

Unit detail

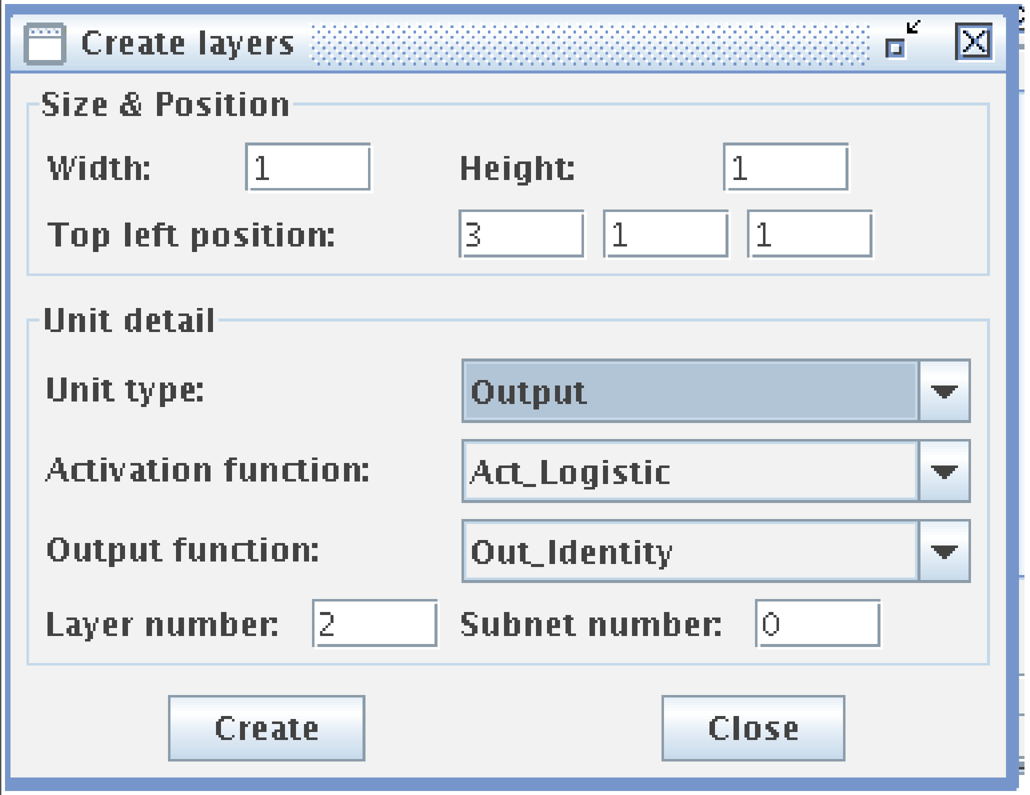
Unit type: Input

Activation function: Act_Identity

Output function: Out_Identity

Layer number: 1 Subnet number: 0

Input layer has 2 units with the identity function as the activation function. Unfortunately, JavaNNS insists that these units have biases. Formally, this is not strictly necessary.



Create layers

Size & Position

Width: 1 Height: 1

Top left position: 3 1 1

Unit detail

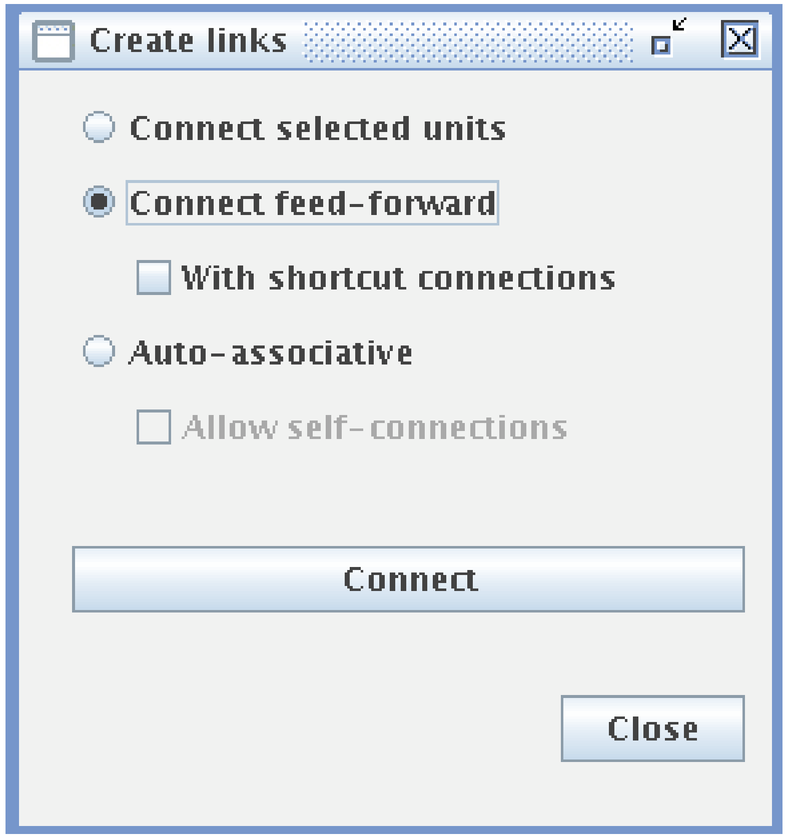
Unit type: Output

Activation function: Act_Logistic

Output function: Out_Identity

Layer number: 2 Subnet number: 0

We need one output unit, and we give it the nonlinear logistic activation function



Create links

○ Connect selected units

● Connect feed-forward

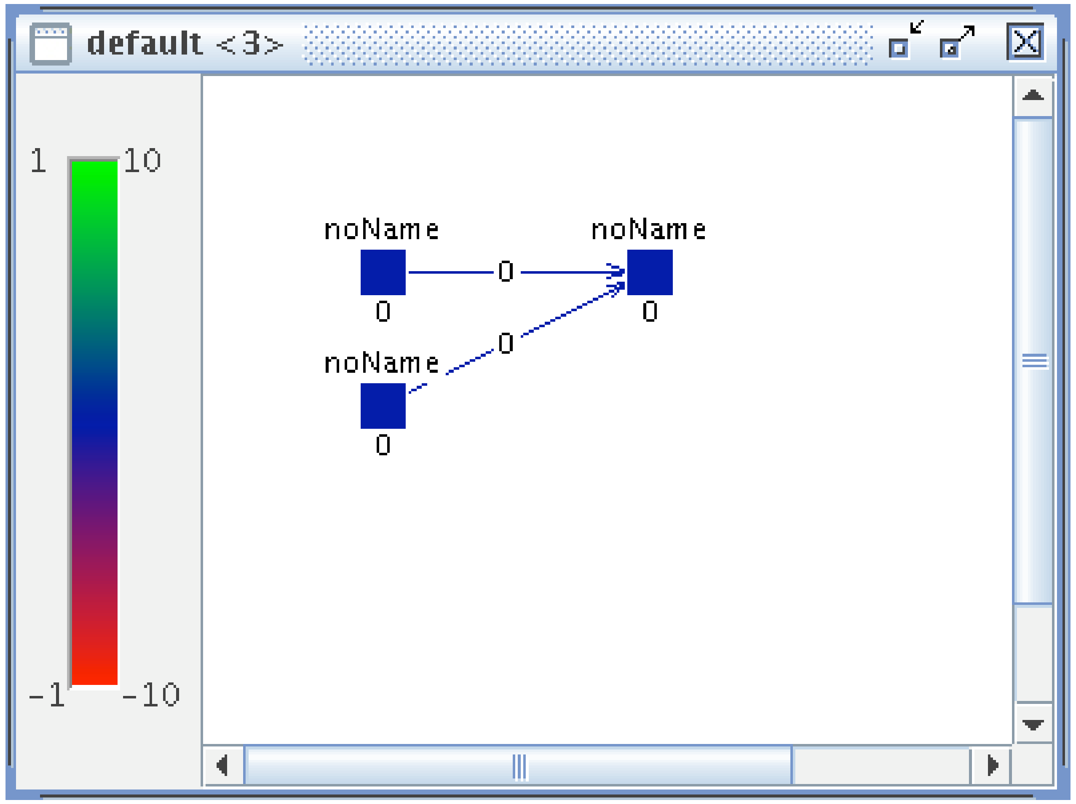
□ With shortcut connections

○ Auto-associative

□ Allow self-connections

[Tools] > [Create] > [Connections]

Select ‘Connect feed-forward’



The 2 x 1 network, fully connected.

In SNNS, the bias is not implemented as a separate, always ‘on’ unit, but as a parameter of each unit. They are trained along with the weights.

Now save the network in your project folder. Call the network myAnd.net.

You should create a separate project folder for each project you undertake with JavaNNS. This will allow you to identify those files which collectively relate to a single project.

Now we need to load some patterns. To do this, you will need to copy either of these two sample pattern files to a file called myAnd.pat in your project folder. The formats shown here differ only in the human readable comments. To the network, they are the same.

Verbose

```
SNNS pattern definition file V3.2
generated at Mon Apr 25 15:58:23 1994


No. of patterns : 4
No. of input units : 2
No. of output units : 1

# Input pattern 1:
0 0
# Output pattern 1:
0
# Input pattern 2:
0 1
# Output pattern 2:
0
# Input pattern 3:
1 0
# Output pattern 3:
0
# Input pattern 4:
1 1
# Output pattern 4:
1
```

Concise

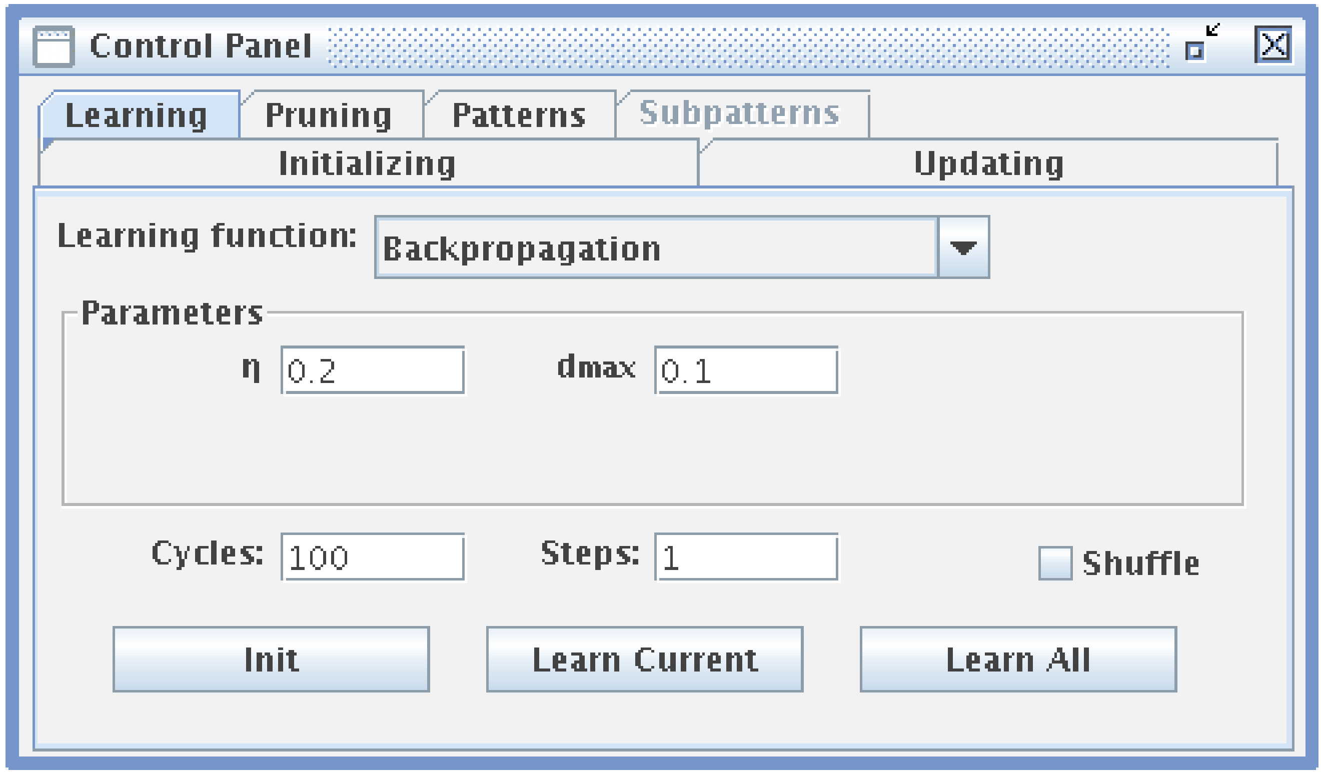
```
SNNS pattern definition file V3.2
generated at Mon Apr 25 15:58:23 1994


No. of patterns : 4
No. of input units : 2
No. of output units : 1

0 0 0
0 1 0
1 0 0
1 1 1
```

Use File -> Open to load your patterns

The control panel has 6 tabs. Select the ‘Learning’ tab.



Learning function: Backpropagation

Parameters

η 0.2

dmax 0.1

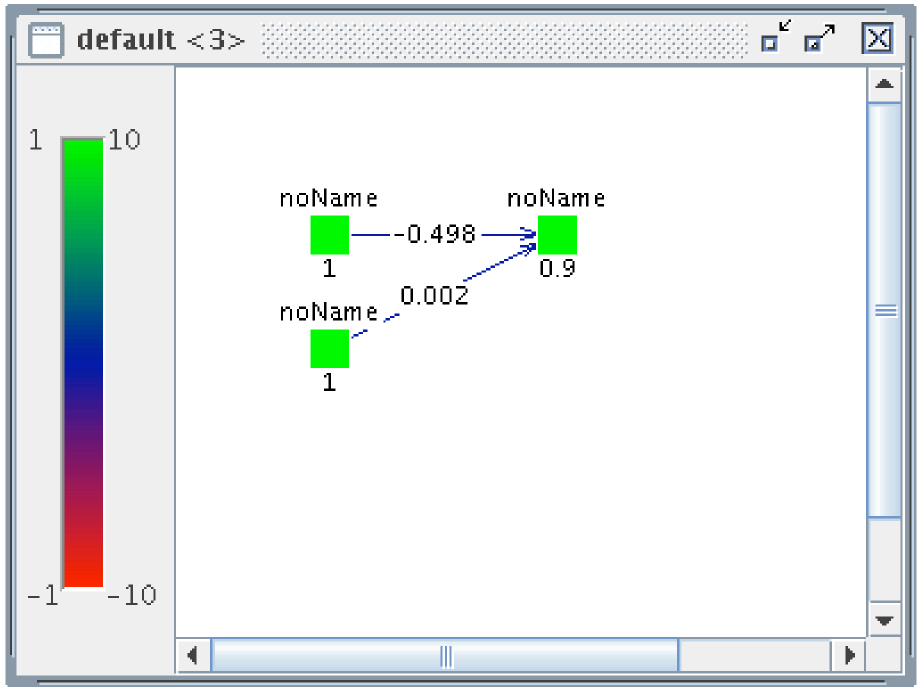
Cycles: 100

Steps: 1

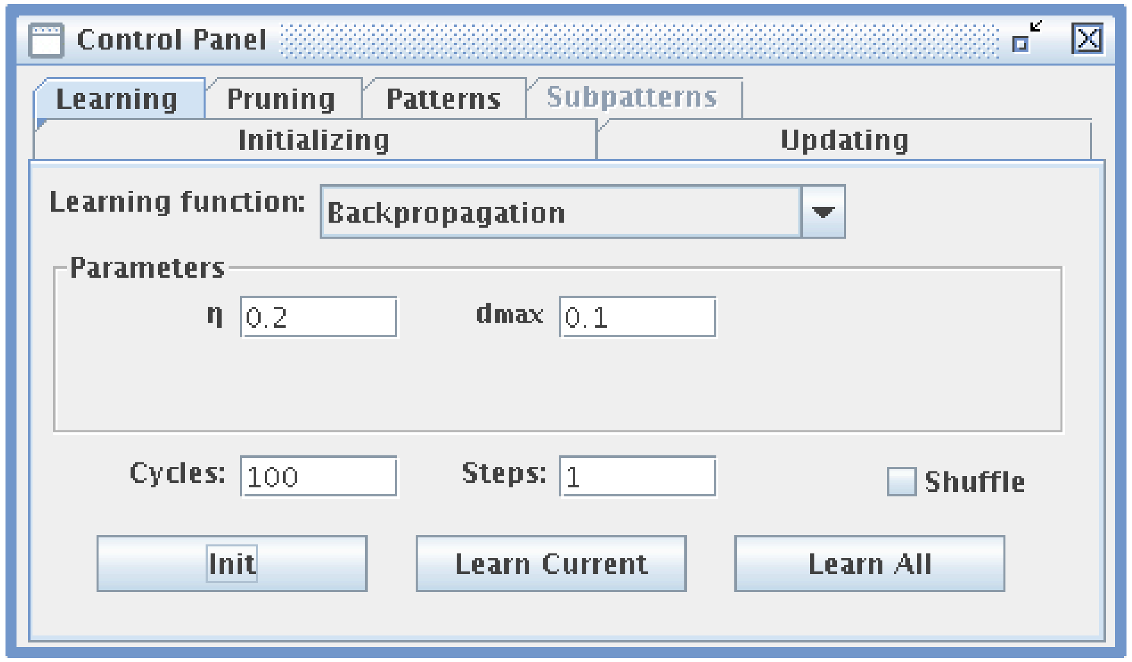
☐ Shuffle

Leave the learning rate (η) as is. d_max will not bother us here.

Next: initialization and training



Click ‘Init’ in the Learning tab. Note that the weights (and the biases, not shown) are provided with random values. The range of these values is set in the ‘Initialization’ tab, but we can be happy with the default range of [-1,1].

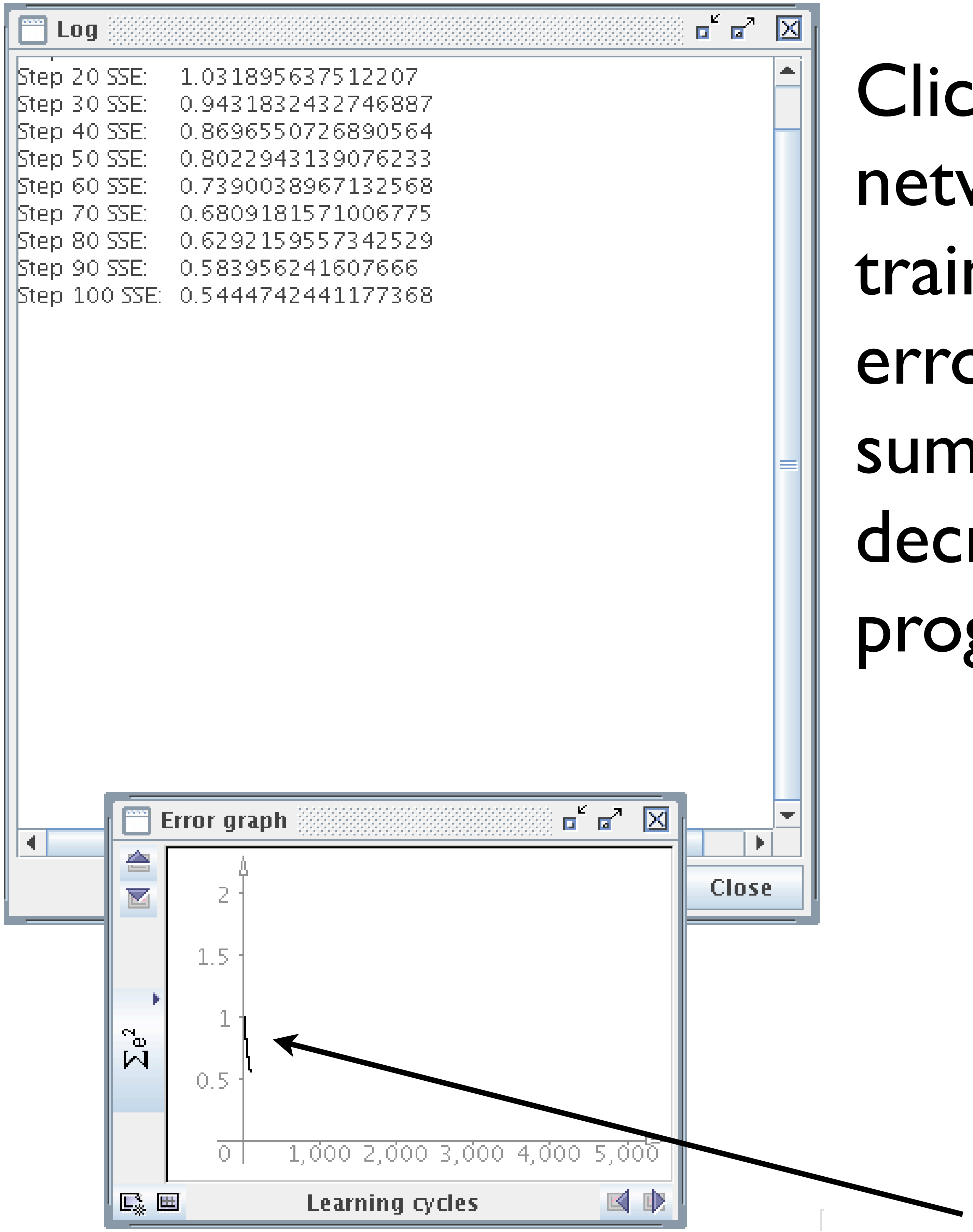


Learning function: Backpropagation

Parameters

η 0.2 dmax 0.1

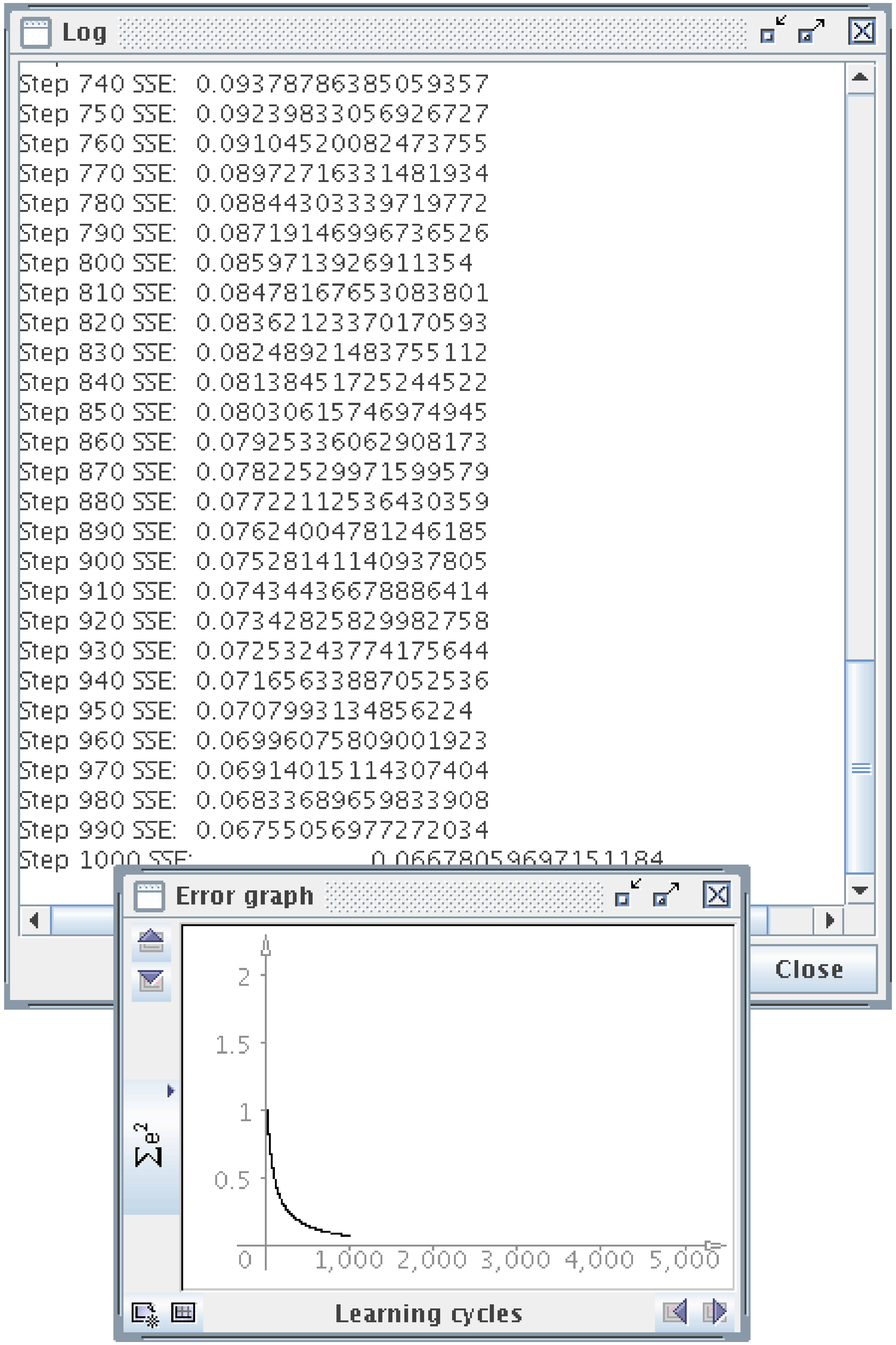
Cycles: 100 Steps: 1 □ Shuffle



```
Step 20 SSE:    1.031895637512207
Step 30 SSE:    0.9431832432746887
Step 40 SSE:    0.8696550726890564
Step 50 SSE:    0.8022943139076233
Step 60 SSE:    0.7390038967132568
Step 70 SSE:    0.6809181571006775
Step 80 SSE:    0.6292159557342529
Step 90 SSE:    0.583956241607666
Step 100 SSE:   0.5444742441177368
```

Click on 'Learn All'. The network will now perform 100 training cycles (epochs). The error plot shows how the summed squared error decreases as training progresses.

Plot of summed squared error

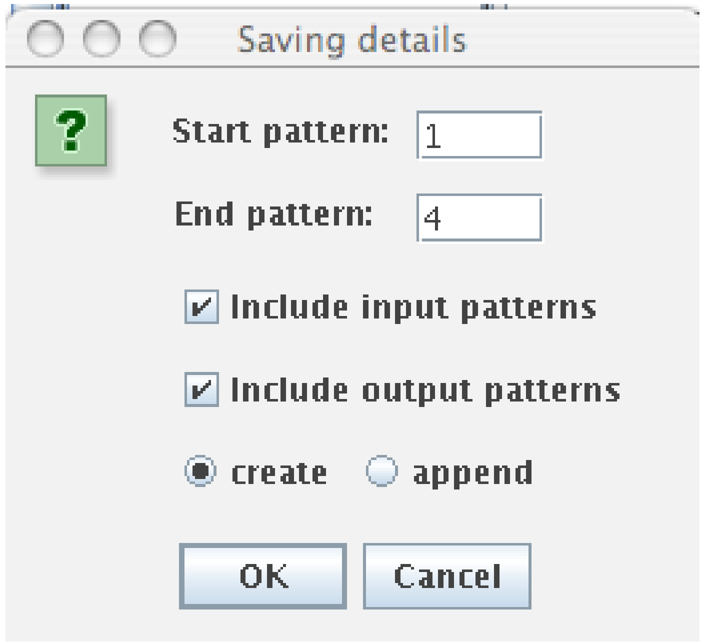


```
Step 740 SSE:  0.0937878638505935 7
Step 750 SSE:  0.09239833056926727
Step 760 SSE:  0.09104520082473755
Step 770 SSE:  0.08972716331481934
Step 780 SSE:  0.08844303339719772
Step 790 SSE:  0.08719146996736526
Step 800 SSE:  0.085971392691135 4
Step 810 SSE:  0.08478167653083801
Step 820 SSE:  0.08362123370170593
Step 830 SSE:  0.08248921483755112
Step 840 SSE:  0.08138451725244522
Step 850 SSE:  0.08030615746974945
Step 860 SSE:  0.07925336062908173
Step 870 SSE:  0.07822529971599579
Step 880 SSE:  0.07722112536430359
Step 890 SSE:  0.07624004781246185
Step 900 SSE:  0.07528141140937805
Step 910 SSE:  0.07434436678886414
Step 920 SSE:  0.07342825829982758
Step 930 SSE:  0.07253243774175644
Step 940 SSE:  0.07165633887052536
Step 950 SSE:  0.0707993134856224
Step 960 SSE:  0.06996075809001923
Step 970 SSE:  0.06914015114307404
Step 980 SSE:  0.06833689659833908
Step 990 SSE:  0.06755056977272034
Step 1000 SSE:              0.06678059697151184
```

Repeatedly clicking on the 'Learn All' button extends training. Note how error drops.

Congratulations. You just trained a neural network. But did it learn?

Choose ‘Save data’ from the file menu. Provide a suitable file name (and make sure it is saving to your project directory). myAnd.res might be appropriate.



Start pattern: 1

End pattern: 4

☑ Include input patterns

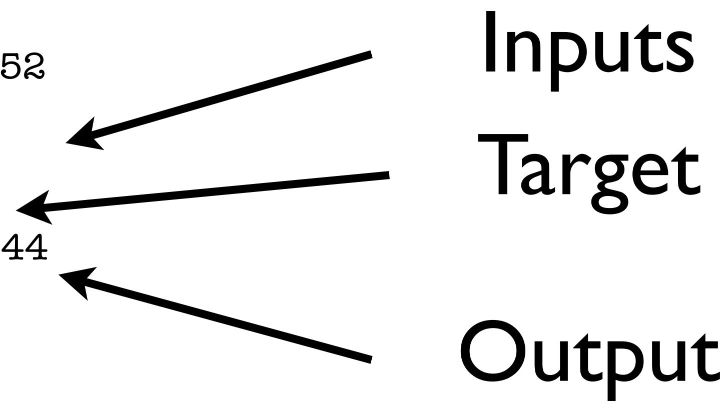
☑ Include output patterns

● create ○ append

For now, include both input and output patterns in the results file.

```
SNNS result file V1.4-3D
generated at Wed May  2 13:57:15 2007

No. of patterns     : 4
No. of input units  : 2
No. of output units : 1
startpattern        : 1
endpattern          : 4
input patterns included
teaching output included
#1.1
0 0
0
0.00517
#2.1
0 1
0
0.13902
#3.1
1 0
0
0.13952
#4.1
1 1
1
0.83444
```



Inputs

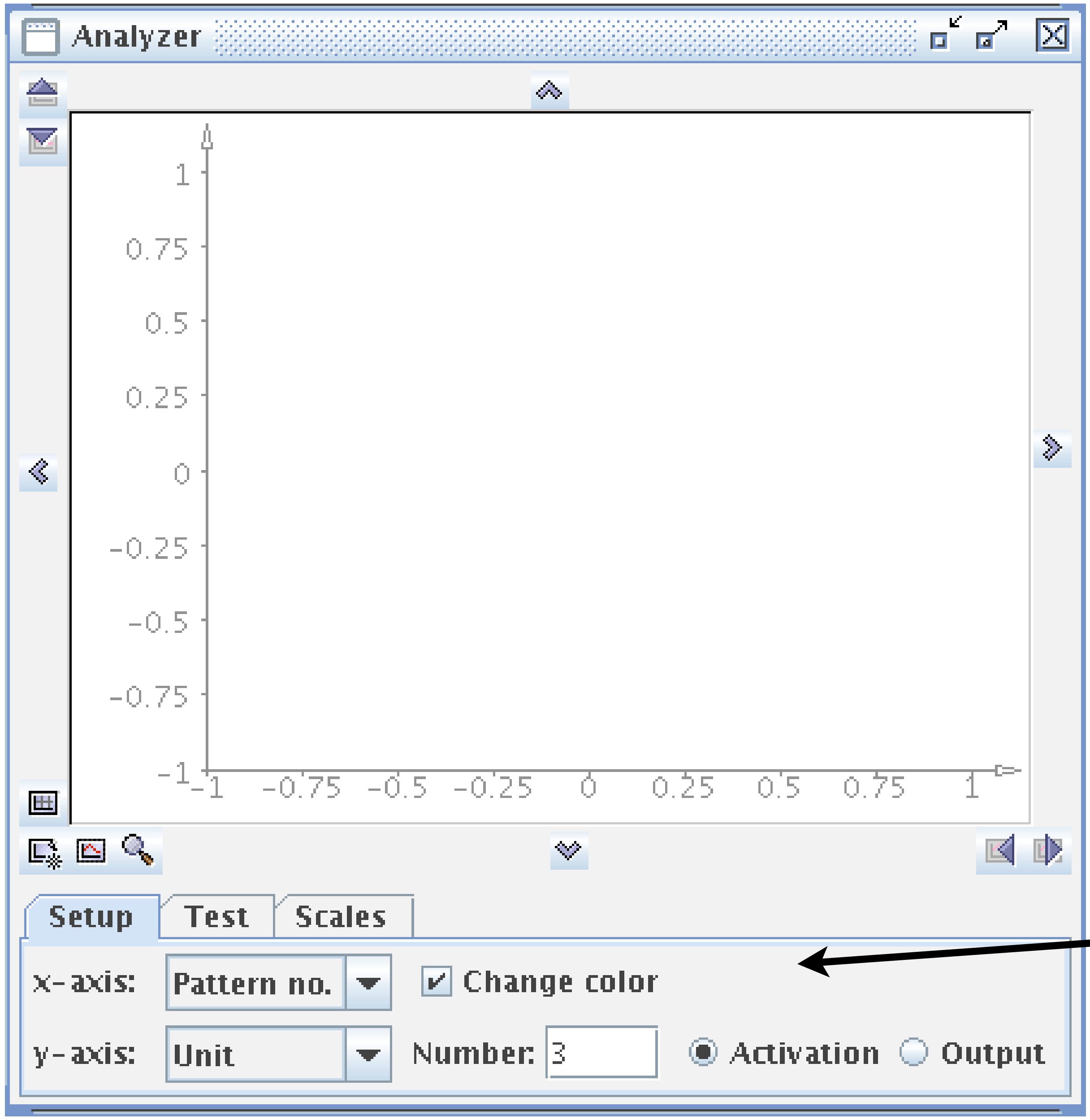
Target

Output

Here are my results. Yours will differ somewhat (why?).

All outputs are ‘close’ to targets. With additional training, they can be gotten closer.

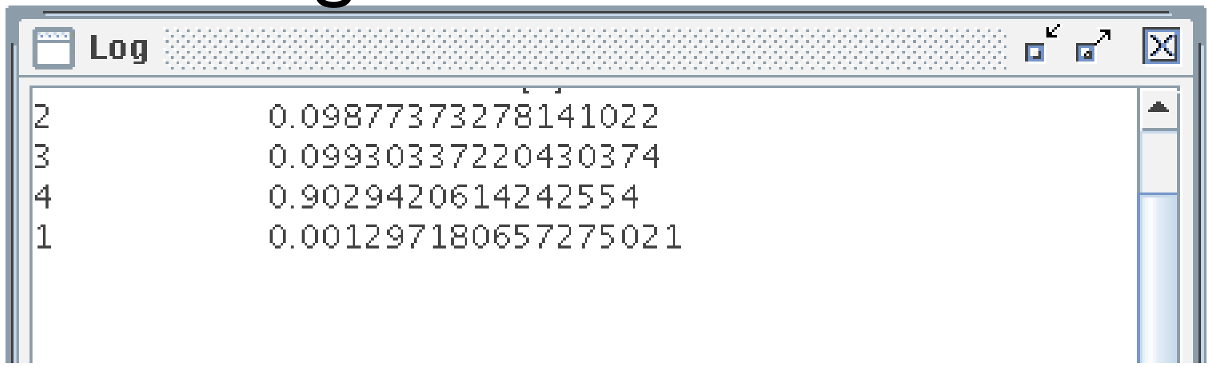
[Tools] > [Analyzer]



x-axis: Pattern no. ☑ Change color

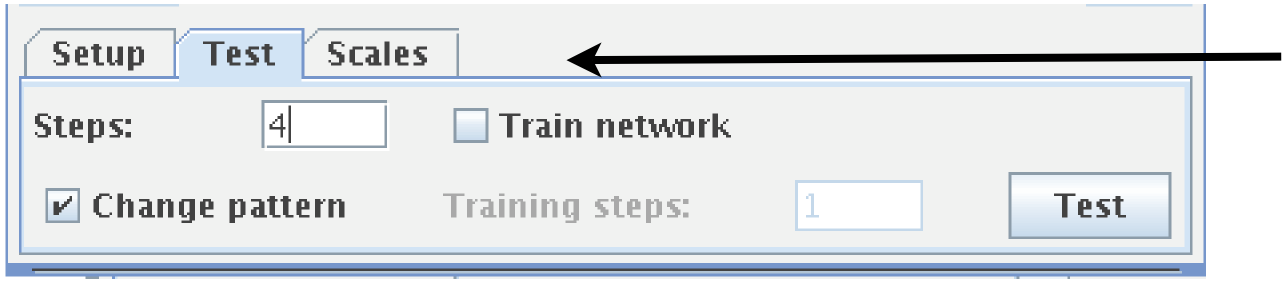
y-axis: Unit Number: 3 ● Activation ○ Output

Results appear in the Log window



```
2          0.09877373278141022
3          0.09930337220430374
4          0.9029420614242554
1          0.001297180657275021
```

Select ‘Pattern no. and Unit. Unit 3 is the output unit.



Steps: 4 □ Train network

☑ Change pattern Training steps: 1

Select 4 steps (for 4 patterns) and click ‘Test’.

Save your work! Weights are stored along with the network configuration in a single .net file.