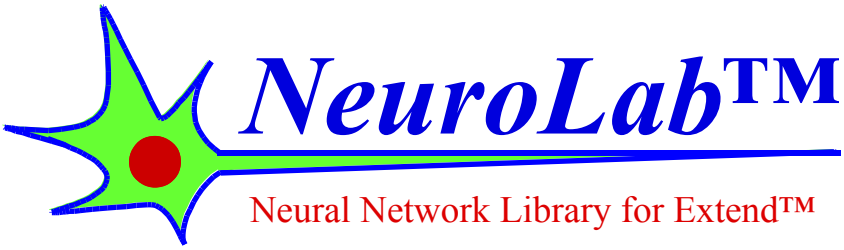
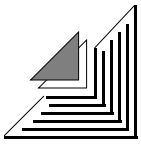


+



Extend™

NeuroLab DEMO Manual

MIKUNI **BERKELEY R&D CORP.**

4000 Lakeside Dr. Richmond, CA 94806

TEL (510) 222-9880 FAX (510) 222-9884

E-mail: neurolab-info@mikuni.com

WWW: http://www.mikuni.com

NeuroLab DEMO manual (Version 1.2, August 1995)

NeuroLab library requires the Extend simulation application developed by Imagine That, Inc. IMAGINE THAT, INC. SPECIFICALLY DISCLAIMS ANY AND ALL EXPRESS AND IMPLIED WARRANTIES INCLUDING, WITHOUT LIMITATION, THE WARRANTY OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE, AND THE STATUTORY WARRANTY OF NON-INFRINGEMENT. In addition, Imagine That, Inc. will not be liable for any special, incidental, or consequential damages of any kind, and whether arising out of breach of contract, warranty, tort (including negligence), strict product liability or otherwise, even if advised of the possibility of strict damage or if such damage could have reasonably been forseen.

The NeuroLab is an add-on library to the Extend simulation application. Just as for Extend, NeuroLab is licensed to be used on only one computer at a time that is owned or operated by the user. If you or your company want to use the NeuroLab on more than one machine, you must purchase a separate license for each machine.

Technical questions
For technical questions and customer support concerning NeuroLab, contact MIKUNI BERKELEY R&D CORP. For technical questions and customer support concerning Extend, contact Imagine That, Inc.

About NeuroLab
Mikuni Berkeley R&D Corp.
4000 Lakeside Dr.
Richmond, CA 94806
Tel: 510-222-9880
Fax: 510-222-9884
E-mail: 72450.3237@compuserve.com
neurolab-info@mikuni.com
WWW: http://www.mikuni.com/neurolab

About Extend
Imagine That, Inc.
6830 Via Del Oro, Suite 230
San Jose, CA 95119
Tel: 408-365-0305
Fax: 408-629-1251

Demo 3

Contents

Demo 4

Installation and requirements

This NeuroLab Demo v1.2 is executable only under the Extend 3.1 Demo program. If you already have Extend, please run the examples from application. In other words, to run the examples in this software package, double click application (Extend 3.1 Demo) first and open the examples from “Open” in the “File” menu.

The requirements of this **NeuroLab™ Demo v1.2** are

Macintosh® II or later (faster machines with FPU are recommended)
Extend (demo version 3.1 is included)
System software 6.07 or up
7 MB free space to install NeuroLab Demo and Extend Demo on the hard disk.

NeuroLab Demo v1.2 comes with two high density floppy disks. Files in the disks are in a compressed format (except NeuroLab ReadMe). To install,

- Insert disk labeled “NeuroLab Demo #1.”
- Double click file named “NeuroLab Demo Installer.” An installation window will show up on the screen.
- Select a destination drive for the installation by clicking “Drive” button.
- Click “Install” button to start the installation.
- Follow instruction to insert disk “NeuroLab Demo #2.”
- The installation is complete when you see dialog “The "NeuroLab™ Demo" has been installed successfully” on your screen.

The installer creates a new folder named “NeuroLab Demo v1.2” in the destination drive’s (you just assigned) root folder. All required files are put in the folder.

Chap. 1. Introduction

Before you try NeuroLab models

This manual is written under the assumption that you already have experience with the Extend program and Macintosh computers. Users need to have some knowledge about Extend such as distinction of input and output connectors, how to open a library and put blocks in the model, how to set simulation parameters, how to connect blocks, how to change the parameters of blocks and so on. If you are not familiar with these topics, you might want to take a look at the "Extend Demo Manual", try to run the tutorial models in the "Tutorial" folder, and run some example models.

❒ What are Artificial Neural Networks ?

The human brain may be the most sophisticated biological neural network in the universe and is often much more efficient, adaptable and tolerant than conventional computers in the fields of recognition, control and learning. For example, a baby can easily distinguish its parents' faces every time, even though the faces appear in different orientations or in different lighting. The number of neurons in the human brain is estimated to be the order of 10^{11} and the processing speed of a neuron is about 10^{-2} second. Although the processing speed of biological neurons is much slower than that of digital computers, the massive parallel processing power of these neurons overcomes their speed deficit. Most biological neurons have nonlinear responses to input stimuli to learn the nonlinearities of required functions. The redundant structure of biological neural networks with sufficient numbers of neurons helps to make a fault tolerant system because the contribution of one neuron/subnet is very small and can be covered by other neurons/subnets by adaptation or learning. The nonlinearity of neurons, the massive parallel processing power and the redundancy of networks are the most important reasons for the human brain's superiority.

The development of artificial neural networks was inspired by neuroscience, the study of the brain, biological neurons and synapses. They are intended to mimic the behavior of biological learning and decision-making processes, without trying to be biologically realistic in detail. Artificial neural networks are considered to be a simplified model of the human brain used to solve difficult problems in conventional computers by conventional methods. Desirable features and motivation for research in artificial neural networks are:

- highly parallel computation --- The developed algorithms can be implemented into special hardware/computers.
- system flexibility --- Networks can adapt to new environments by "learning." The network does not have to be programmed in FORTRAN or C.
- robustness and fault tolerance, as in the brain --- Nerve cells in the brain die every day without affecting its performance.
- capability of handling imperfect information --- Networks can deal with fuzzy, noisy, incomplete or probabilistic data.

Often "artificial neural networks" are referred to simply as "neural networks". In this manual, "neural networks" always refers to "artificial neural networks" unless they are specified as "biological neural networks".

❒ What is NeuroLab ?

Demo 6

NeuroLab is a neural network library for the block-based simulation program Extend. It is a powerful and effective tool for understanding, designing and simulating neural network systems. The blocks in this library can be used in conjunction with other Extend blocks to create complex application models embedding neural networks. NeuroLab consists of more than 70 functional blocks for artificial neural network implementation and many example models in several professional fields.

The purpose of this library is to help users understand the basics of neural networks and to offer a convenient tool for simulation of neural network applications. The field of neural networks has a history of only several decades and has found solid applications only in the past ten years. Neural networks are still a developing field and flexible/powerful tools are required for the development of algorithms, network structures and applications. With that tool, users can test, modify, simulate and evaluate their models/ideas before the actual implementation. We believe that NeuroLab is an ideal tool for educators, researchers and industrial engineers to explore the power of neural networks.

• Blocks

The functional blocks in NeuroLab are organized and structured to meet the requirements of many levels of end users. For example, the educational blocks will help beginners to understand neural systems since each of these blocks is designed to represent an exact mathematical calculation step in neural computing. NeuroLab also provides intermediate blocks which give users the flexibility to build various neural network structures since these structures are currently major research topics. Advanced/combined blocks serve in the speedy development and simulation of neural network applications since significant software overhead is removed. Several types of neural networks are included in NeuroLab to meet users' application requirements.

• Modeling

Thanks to the inherent features of the Extend program, NeuroLab also provides icon-based functional blocks for easy implementation of high quality simulation models. Just click, drag and connect blocks to see how neural networks learn and solve specific problems in real time. Users don't have to remember cumbersome commands or procedures to build models.

Users can construct neural networks from individual neurons, single-layer networks or multi-layer networks depending on which blocks are used. Network parameters such as the specific back-propagation methods, learning rates, initial weights and biases, and neuron number and type are easily changed in the dialog boxes of the functional blocks. In addition, familiarity with the Extend model simulation scripting language *ModL* allows you to modify the NeuroLab blocks for your own purposes with customized icons, dialog boxes, and of course with your codes. You can also include compiled program modules which are written in other languages using XCMD (external commands) and XFCN (external functions) interfaces for Macintosh and DLL (dynamic-link libraries) for Windows.

NeuroLab provides many kinds of output blocks to monitor neural network status using vivid color displays and animation in real time. The calculated results and final network status are easily saved to disk files for later simulation and examination. Data is imported to the models by text files.

• Applicable fields

Demo 7

NeuroLab is applicable in any field where nonlinear mapping and adaptive capability are integral parts of problem-solving. Data processing is perhaps the most prominent area for their utilization, especially in image processing (character recognition, object recognition and image restoration), since neural networks can learn or memorize nonlinear mappings between inputs and outputs. Another application of neural networks is adaptive nonlinear control because neural networks learn and adapt to nonlinear dynamics. Medical diagnoses using large amounts of patient data are one of the data classification examples which require nonlinear classification capability. Other fields include optimization, approximation and future prediction of nonlinear problems. NeuroLab provides easy-to-modify high quality examples in these areas. NeuroLab adds new dimensions for enhancing many applications.

❒ NeuroLab™ feature summary

- Interactive network parameters in the dialog box
  - *Neuron numbers*
  - *Neuron types*
  - *Initial weights & biases*
  - *Learning method*
- Various types of neural networks
  - *Hopfield network*
  - *Perceptron*
  - *Competitive networks*
  - *Feature maps*
  - *Recurrent networks*
  - *Context networks*
  - *Boltzmann machine*
  - *Counterpropagation network*
  - *Single-layer feed-forward networks*
  - *Multi-layer feed-forward networks*
- Many back-propagation options
  - *Momentum method*
  - *Adaptive learning rate*
  - *Normalized method*
  - *Accumulated learning*
- High quality application examples in many fields
- Network status monitored in real time
- Educational blocks for beginners
- Intermediate blocks for flexible-structure neural networks
- Advanced, combined blocks for speedy applications
- More than 70 functional blocks

❒ General rules (Color of block & Variable type) of notations

In this manual, the notations of certain items are distinguished by typeface for clarity. The typefaces for each item are as follows:

Command menu items are enclosed in quotation marks	----> "Menu item"
Items in the dialog box are printed in italics	----> • *Dialog item*
Emphasized words are printed in italics	----> *Concept word*
The name of each block, model and library is printed in bold	----> **Block name**
The name of each connector is printed in italics	----> *in*, *out*, *array*

Mathematical notations

For clarity of mathematical understanding, the notation of variables are described here. A one dimensional variable and a two or more dimensional variable are called *vector* and *matrix*, respectively.

Variable	Example	Explanation
Scalar	*w*	lower case
Vector	$\underline{W}$	upper case with underline
Matrix	$\underline{\boldsymbol{W}}$	upper case with underline, bold type face

Thus the variables with underlines may have more than one element. The *i*-th element of a vector $\underline{V}$ and the element of

a matrix in the *j*-th row, *i*-th column are expressed by v_i and w_{ji} respectively.

Demo 9

Scalar vs. Array

In the modeling with NeuroLab, the word *array* is used to refer to both vectors and matrices in order to clearly distinguish these from *scalars*. Thus there are only two data types in NeuroLab block connectors: the scalar and the array. The term array refers to all variables from one dimensional to five dimensional.

Block colors

All blocks in NeuroLab are grouped and color-coded according to their functions. The color of each block is defined by the perimeter color of that block . Color coding is as follows:

Color	Functions
Green	Input from file or function generator as model input
Red	Output to file or display results; animation blocks
Light green	Dynamics and control
Light blue	Basic function; data handling, data conversion
Brown	Neural network blocks
Orange	Educational blocks

This is convenient for making and interpreting models since it is possible to recognize the basic functions of the blocks from their colors.

Connector names and colors

All blocks in NeuroLab have *value* input and output connectors (not *item* input and output connectors for discrete event simulation; refer to the Extend manual for details). There are two types of value input and output signals: scalar and array. If variables are combined into an array, model connections are simplified and easy to trace because there are fewer connections between blocks. Most connectors of NeuroLab blocks are assigned to array signals (except in the case of educational blocks) to take advantage of this. On the other hand, the user has to be careful not to mix up array connectors and scalar connectors since a connection between an array connector and a scalar connector is possible and might give the user garbage simulation results. To reduce user confusion between scalar signals and array signals, each connector in the NeuroLab library is clearly specified by data types.

The array connectors are visually distinguishable by outer covers attached to the original *value connectors* (refer to Extend's manual). In addition, the connector name colors are assigned. The connectors which have underlined names are assigned to array (light blue for input, dark blue for output). The scalar connectors' (with names not underlined) colors are green for input and red for output.

Variables	Connector	Color	Sample
Scalar In		Green	*in*
Scalar Out		Red	*out*
Array In		Light blue with underline	*netIn*
Array Out		Dark blue with underline	*netOut*
Changeable		Violet	*flex*

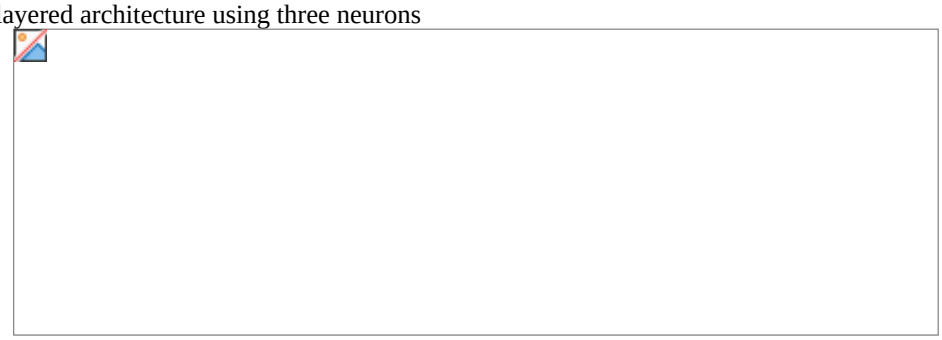
All blocks for continuous simulation which come with the Extend program use scalar value input and output. Even though data type checks are implemented in NeuroLab blocks, physical connections between scalar and array connectors are possible. In this case, an error message will be displayed during execution and the simulation will be stopped. The NeuroLab libraries provide three blocks to convert data types, **Scalar to Array**, **Array to Scalar**, and **Shift register.**

Demo 10

Chap. 2 Examples

	Xor.Edu.Layered	ÿ	**EDU_XOR2.MOX**
	Xor.Edu.Mixed	ÿ	**EDU_XOR.MOX**
Exclusive or (XOR) Example with Educational Blocks			

I. Model A — layered architecture using three neurons



II. Model B — flexible architecture using two neurons

Construction of the above two neural networks requires **Adaptive Weighted Sum**, **Sigmoid**, **Linear**, **I/O Control** and **BP Control** blocks in NeuroLab. In addition, **Input Data** and **Plotter I/O** blocks in Extend generic and plotter libraries are used to setup the desired input/output relationship to train the networks and to display simulation results.

Description

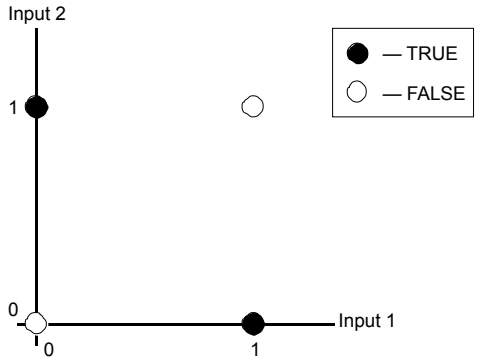
The XOR function can be considered as an input/output mapping,

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

The function output is TRUE when one and only one of the two inputs is on and is FALSE when the two inputs are both on or off. The exclusive or function can be interpreted geometrically as next figure.

Demo 11

It is easy to see from the figure that the function is not linearly separable. Thus, a simple perceptron that can handle only linearly separable problems, doesn't have enough freedom to achieve the input/output mapping of the XOR function. Algebraically, it can be shown that a two-layer feed-forward neural network is capable of learning the XOR problem. Intuitively, a two-layer perceptron that is able to handle mapping between two inputs and one output should have at least three neurons, as shown in model A. Model B is an alternative network architecture that can achieve the mapping of XOR function but uses only two neurons. Note the direct connections between input signals and the output neuron.



A basic neuron has two basic functions — adaptive linear summation and nonlinear transformation. NeuroLab provides **Adaptive Weighted Sum** and several nonlinear transformation blocks to simulate the two fundamental functions. In the first model, we use three **Input Data** blocks for implementing of the exclusive Or logic. For building the network, we use three **Adaptive Weighted Sum** blocks associated with two **Sigmoid** blocks and one **Linear** block to simulate the hidden and output neurons. The **I/O Control** and **BP Control** blocks are introduced to handle the forward and backward calculation flow of the network. Finally, a **Plotter I/O** block is used to monitor the accumulated square error of the network's computation. With these blocks, the construction of the Extend model is straightforward,

- Add proper **Adaptive Weighted Sum** and **Sigmoid** blocks to the newly created model window by selecting block names from Library menu or by dragging block icons from the opened library window to the model window.
- Add an output neuron to the model. The output neuron consists of a **Linear** block and a third **Adaptive Weighted Sum** block.
- Add an **I/O Control** and a **BP Control** blocks to the model to manage simulation orders of the network.
- Add three **Input Data** blocks to the model. The three blocks will be used to define the XOR logic.
- Add a **Plotter I/O** block to the model for monitoring the simulation results.
- Arrange the blocks in the model window accordingly and make the proper connections.

In the network, dialog boxes of the three **Input Data** blocks define the XOR logic which are used to train the network,

Demo 12

Row	Time	Y Output
0	0	0
1	1.5	0
2	1.6	1
3	3.5	1
4	3.6	0
5		
6		
7		
8		
9		
10		

Input 1

Row	Time	Y Output
0	0	0
1	0.5	0
2	0.6	1
3	1.5	1
4	1.6	0
5	2.5	0
6	2.6	1
7	3.5	1
8	3.6	0
9		
10		

Input 2

Row	Time	Y Output
0	0	0
1	0.5	0
2	0.6	1
3	2.5	1
4	2.6	0
5		
6		
7		
8		
9		
10		

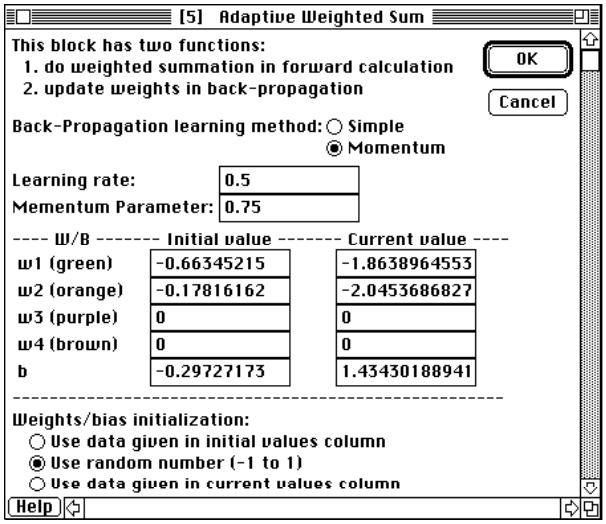
Desired output

The data sequences are repeated every 4 seconds. Once the desired input/output relationships are defined, the user can run the model and witness the decreasing of output error because all other blocks have their meaningful default values and setups.

Which parameters you should change

During simulation, current neuron weights and biases are displayed in **Adaptive Weighted Sum** dialog boxes, and square errors are plotted in the **Plotter I/O** window.

To further explore the power and flexibility of the model, we suggest that users examine the **Adaptive Weighted Sum** block in detail, since parameters determining the learning processes, e.g., learning methods, learning rates, and initial conditions, are defined in this block.



There are two back-propagation methods, simple and momentum, to be chosen in the **Adaptive Weighted Sum** block dialog. Simple learning is

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

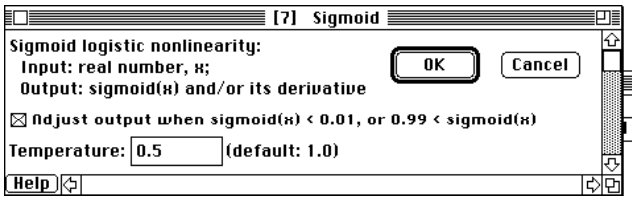
and for momentum learning is

$$\Delta \omega_{i\varphi}(\tau+1) = -\eta(1-\alpha)\frac{\partial E}{\partial \omega_{i\varphi}} + \alpha \Delta \omega_{i\varphi}(\tau)$$

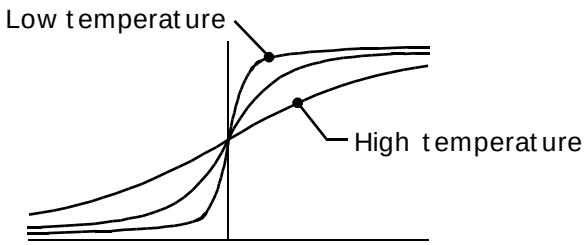
where E is output error of the network, and η and α are the learning rate and momentum parameter, respectively. Furthermore, we can select one of the three provided weight/bias

initialization schemes for different situations. For example, by using same initial values of weights and biases, it is easy to compare the convergence speeds of different learning methods. On the other hand, by using the most current weights and biases, we can accumulate learnings from different simulations, therefore speed up the learning process and shorten the total learning time. Additionally, if a good estimation of initial conditions doesn't exist, we can assign random small numbers as initial weights and biases. However, the network might sometimes converge to a local equivalent point. In that case, we need to try other sets of initial conditions.

Among the educational blocks, there are six nonlinear transfer functions that can be applied for constructing of neurons. In this model, the sigmoid function is used in the two hidden neurons. We can assign a positive number, called temperature, to the sigmoid function,

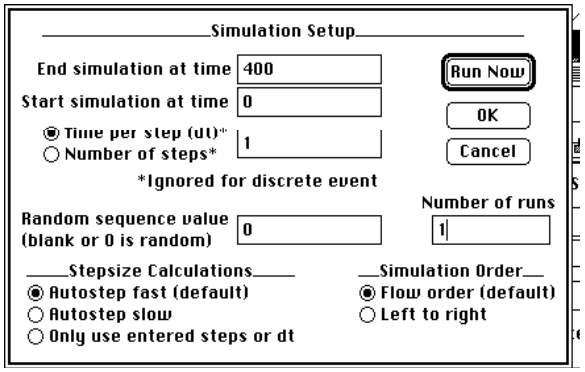


In addition to the learning rate and momentum parameter defined in **Adaptive Weighted Sum** dialog, the temperature also plays an important role in neural network convergence. Schematically, the temperature parameter affects sigmoid functions in the following way,



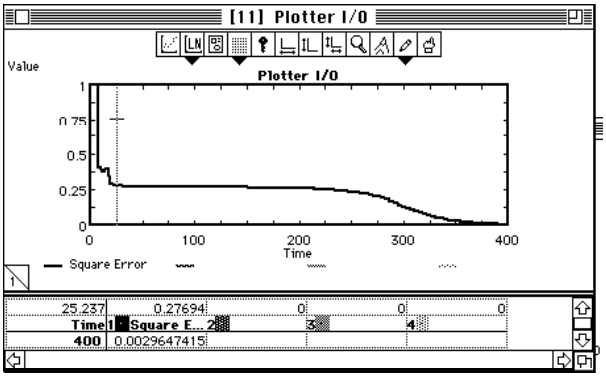
Therefore, nonlinearity of sigmoid function at low temperature is stronger than that at high temperature. In general, we can increase convergence speed by reducing the temperature. However, if the temperature is too low, the computation can be trapped by some local minimum or even become unstable. Try to find an optimal temperature for a certain set of initial conditions.

At this stage, the network is ready for simulation. However, before running the simulation, we should have proper simulation setups. In this problem, we chose



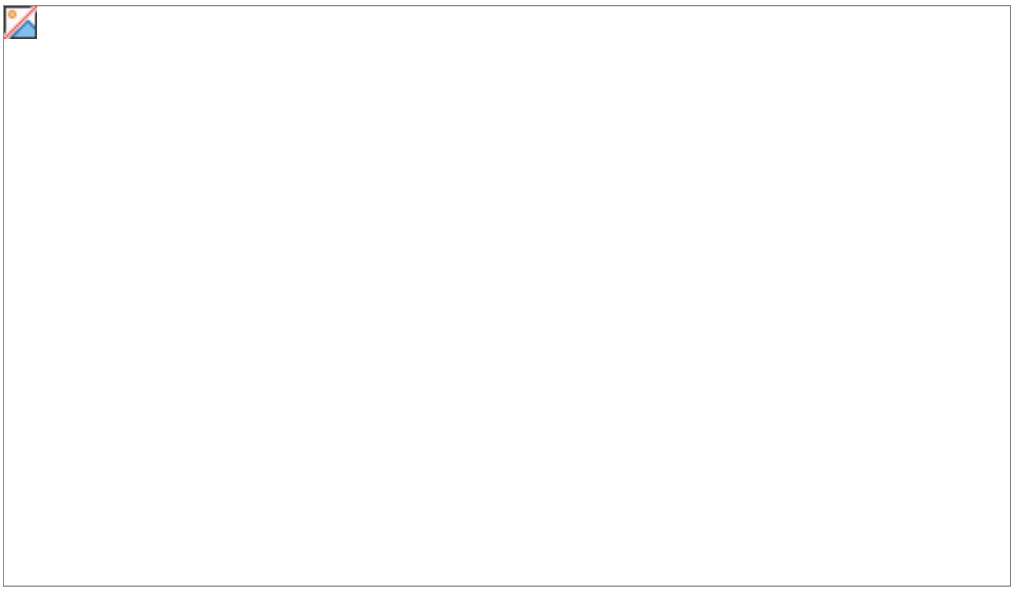
Demo 14

During simulation, the plotter block monitors square error of the network output. A typical simulation result is



which shows a gradually decreasing of the error. In the plot, the square errors are accumulated four times (specified in **BP Control** block dialog) during simulation, because we have four sets of input/output relationships.

	Decoder	ÿ	**DECODER.MOX**
Two-bit decoder with single-layer network blocks			



Description

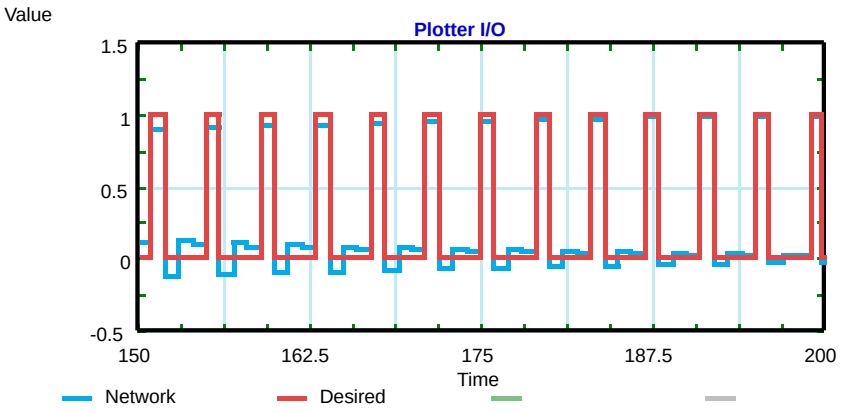
This is an example of a two-bit binary decoder using a feed-forward network. The desired input-output relationship is specified in a data file (the file name is “Decode.txt”). The format of this data file is simple and can be easily created using a word processing program (don’t forget to save in “text only” format). The first row of this file is a comment line. The second line specifies three parameters: the number of patterns, the number of inputs, and the number of outputs. The data given below contains four patterns with two bit inputs and four bit outputs. In the next four rows

of data, the first two columns are the inputs and the last four columns are the corresponding desired outputs.

```
// Desired input-output relationships for the Decoder example
4       2       4
0       0       1       0       0       0
0       1       0       1       0       0
1       0       0       0       1       0
1       1       0       0       0       1
```

In this example, the redundancy of the neural network is tested. The minimum network to achieve this decoding function is one hidden layer with two neurons and one output layer with four neurons.

The next graph shows the computed network outputs and desired outputs of one of the four output neurons. You can see that the network outputs gradually converges to the desired outputs as learning progresses. You can specify the view area of plotter by changing the minimum and maximum values of two coordinates.



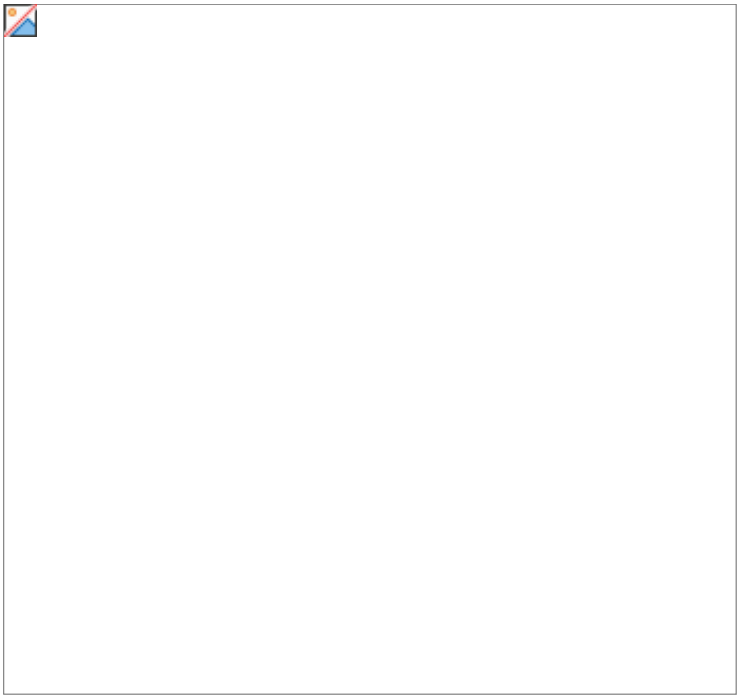
The weight and bias values can be monitored in real time using color animation in the **Matrix Color Display** block. The display's row corresponds to the neuron number in the current layer and the column corresponds to the neuron number in its previous layer. The values close to zero are displayed in green color. Using this information you can easily find out if one of the neurons is not necessary, because all connections to that neuron are displayed in green.

To see the color display of the weights and biases during simulation, make sure the "Show Animation" option in the "Run" menu is on. You can specify how often animation is updated in the dialog box of the **Matrix Color Display** block.

Which parameters you should change

To see different convergence speeds, change the learning rates or use other back-propagation options. To observe the effects of the network structure, change the number of neurons of each layer or add/delete the hidden layers.

	Character Recognition	ÿ	**CHARRECG.MOX**
Character recognition example with Hopfield network.			



Description

This example is a self-correlation memory test using a Hopfield network. The sample data and calculation method of the Hopfield network used here are exactly the same as in [Lippmann, 1987]. The memorization method used in a Hopfield network is different from the memorization methods of conventional computers. In the network, images are stored in the pattern of weight values instead of the actual pixel values. After base images are stored in the network, a probe image is presented to the network. If the probe image is close to one of the stored base images, this base image will be retrieved.

The upper four characters shown with the model(12*10=120 bits/image) are base images and the probe image is a character 3 with some added random noise. The weight values of the Hopfield network block are calculated from the base images at the beginning of the simulation. Then the probe image is fed into the network for the initial states of network neurons and, during simulation, the network neuron states are calculated until there are no further changes of neuron states.

After calculation, a clear image of the character 3 is retrieved. The character display blocks animate the image digitally according to the neuron states. The user can see the evolution of the output image (neuron's output) at each calculation step.

Which parameters you should change

To see another character image set, change the image file names. You can make image files easily using a word processing program (don't forget to save in "Text only" format). To see the effect of the stochastic neurons, change the **Hopfield** block to a **Boltzmann Machine** block.

	Nonlinear Dynamics	ÿ	**NL_DYN.MOX**
Nonlinear forward dynamics identification			

Demo 17

Description

This example is the identification of a nonlinear plant's forward dynamics. To identify a nonlinearity, a hidden layer with nonlinear neurons has to be included in the network. The nonlinear plant dynamics is described by a first order nonlinear differential equation in the dialog box of the **N/L Cont Plant** block as text "x1dot = -3.0*x1^3 + u1". Here x1 is the output of the plant and also the plant state.

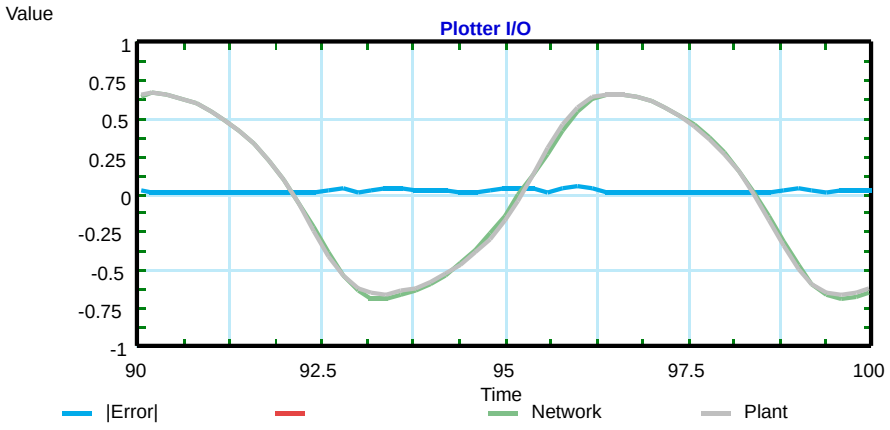
$$\frac{dx_1}{dt} = -3.0x_1^3 + u1$$

The **Shift Register** block provides 10 sampled plant input data for the neural network. The general mathematical expression for this identification is

$$x_1(k+1) = f(u(k),u(k-1),...,u(k-9))$$

This implies that the network tries to estimate the plant output from the history of only the input, since the input usually affects the internal state. In other words, the neural network tries to estimate internal state effects from the input of the plant.

If the function f() is linear, this is a FIR (finite impulse response) filter approximation which is often used for the communication channel approximation in the signal processing field.



Demo 18

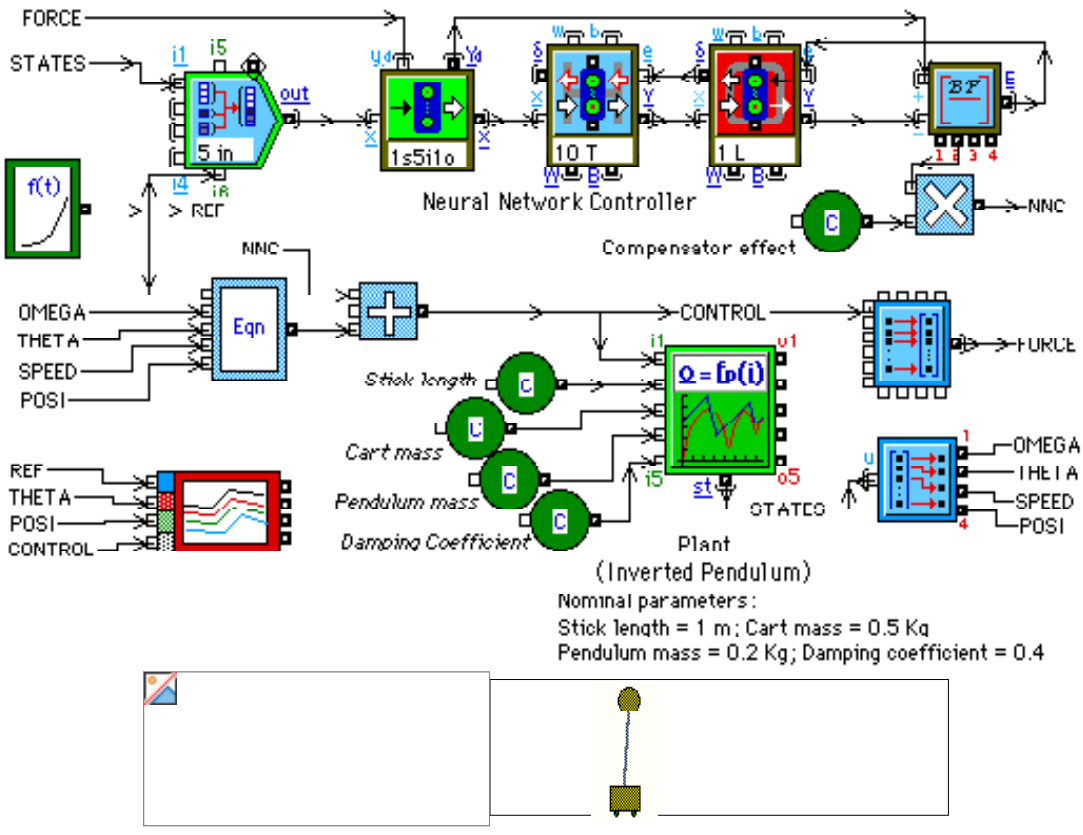
The plot below shows the plant response and neural network approximation to the sine wave after the neural network has been learning for 90 seconds. The user will notice that the plant has a nonlinear response to the sine wave input and the neural network identifier has an excellent estimate of plant output.

This model can be used for nonlinear function estimation (without dynamics) if the equations in the dialog box don't have the derivative keywords "DOT" or "dot".

Which parameters you should change

To investigate the capability for nonlinear identification, change the plant dynamics (order of plant, coefficients of equations, etc.) and change the configuration (neuron type, neuron number, hidden layer number. etc.) of the neural network. To increase the accuracy of the estimation, increase the amount of input data and the number of neurons in the hidden layer. You will notice a calculation speed difference. To increase the nonlinearity of the network, reduce the temperature of the neurons in the hidden layer.

	Inverted Pendulum	ÿ	**INVPEND.MOX**
Inverted pendulum control with feedback error learning.			



Description

This adaptive control example uses a two-layer feed-forward neural network. The plant consists of the cart and the inverted pendulum attached to that cart. The basic idea is to add the neural network to augment the state feedback controller which is tuned for nominal parameter values at the vertical equilibrium position. The control objectives are to balance the inverted pendulum and

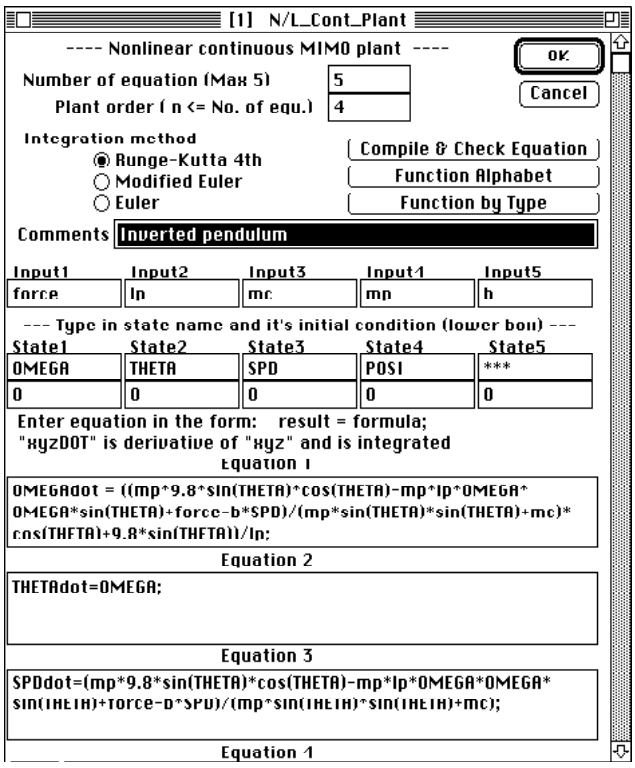
to control the cart position. The neural network is a feed-forward compensator that improves the performance of the state feedback controller by adapting to the nonlinear dynamics and parameter changes of the inverted pendulum.

This learning scheme is called “feedback error learning” [Kawato, 1987] because the output of the state feedback controller is considered the error to be minimized. If the neural network learns the inverse dynamics of the plant, there is no feedback error. Please compare the above model with the following block diagram to find out which parts are the plant, which parts are the compensator and so on.

In this model, the **Array Combiner** block provides the inputs for the neural network. The **Equation** block is the state feedback controller. Four plant parameters are set by the Constant block and the nominal values of the inverted pendulum are listed on this model.

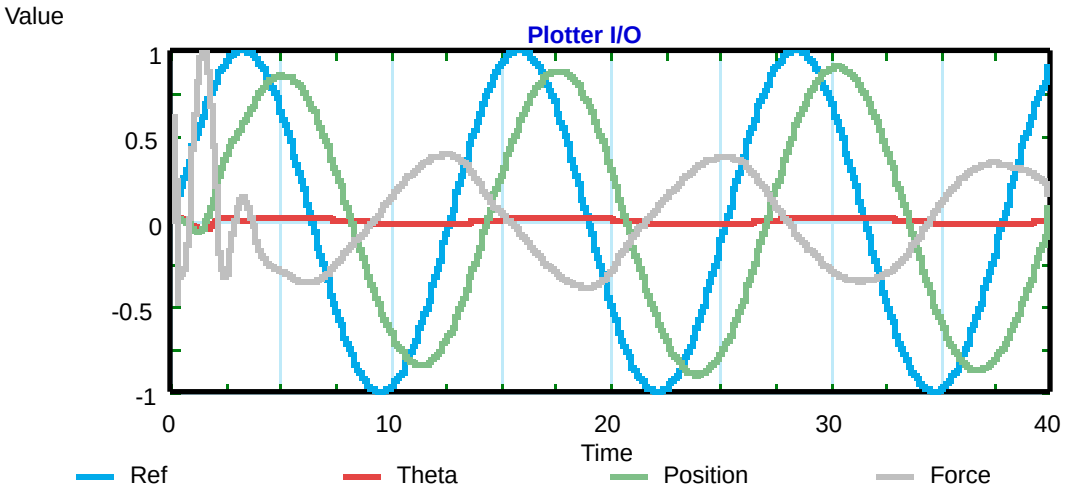
This compensator successfully increases the robustness of the control system to the pendulum's parameter variations; thus, the control system performs better than the state feedback control alone as learning progresses. The use of an advanced, combined two-layer feed-forward network greatly reduces model complexity and increases calculation speed. The animation block shows the real time motion of the inverted pendulum and is a vital tool for presentation.

Demo 20



Which parameters you should change

To see how the neural network compensator increases stability with respect to parameter changes, change the pendulum mass to 1.0 Kg from the nominal value 0.2 Kg. The performance of the controller gets better as the neural network learning progresses. You can see that the neural network successfully compensates for this parameter change in 5 seconds from the plot of force applied to the cart in the following graph.



Then set the value of multiplier for signal NNC to zero to omit the effect of the neural network compensator. The state feedback controller can't balance the inverted pendulum for this parameter value.

Demo 21

To see the capability of nonlinear identification, change the configuration (neuron type, neuron number, hidden layer number, neuron temperature. etc.) of the neural network. The user can try different plants or different control schemes in this model.

	TSP FM	ÿ	**TSP_FM.MOX**
Traveling Salesman Problem (TSP) with Feature Map			

Construction of the model requires **Feature Map (2D to 1D)**, **MNN Input**, and **XY Array Plotter** blocks.

Description

The task of this example is to identify an optimal (or near optimal) closed path for traveling through a group of cities with one and only one stop at each city. The first neural network approach to the TSP was by Hopfield and Tank in 1985, using Hopfield networks. This example demonstrates an alternative approach to the problem using a feature map.

A feature map is a competitive network with information being conveyed in the neurons' geometrical arrangement in the network. By defining a neighborhood function, the winning unit and its neighbor neurons are able to learn during simulation. We expect the network weights will finally be equal to the coordinates of the cities, if we repeatedly present all city coordinates to the network.

For the TSP , network inputs are a set of points (city locations) on a plane and outputs are a closed line passing all the given points. Therefore, the problem requires a feature map that maps data in the plane to a closed line (ring). During learning, the ring will be stretched accordingly so that the final contour passes all given cities.

The feature map approach has one drawback when applied to solve the TSP. Because the overall learning processes are unsupervised, there is no guarantee (or proof) that the achieved travel path is indeed the optimal one. However, intuitively, we are convinced that it is at least a good solution since the path doesn't intersect itself.

Functions of blocks shown in the model are summarized as follows.

• The **MNN Input** block provides data of city coordinates. In each simulation step, one set of city coordinates (i.e., a two-element array) is passed, sequentially or randomly, to the connected network. In the example, we generate 70 random numbers in the interval (-10, 10) to form coordinates for 35 cities located inside a 20 by 20 square region.

City	1	2	3	4	5	6	7	8	9
x	6.6868	4.5289	-3.4113	6.0417	2.6353	-5.0932	-9.1716	-6.9337	5.3413
y	-5.3584	0.7306	-1.3876	-1.9284	9.7465	3.3382	7.0106	7.3503	-3.7679

City	10	11	12	13	14	15	16	17	18
x	-8.7654	0.3861	8.583	-5.5897	-5.9087	-8.2201	3.9938	4.1292	-0.1074
y	-7.0322	9.952	2.7905	-0.6385	8.6383	3.8155	6.3081	0.8307	1.4561

City	19	20	21	22	23	24	25	26	27
x	-4.7644	4.2314	-2.4327	-5.9871	-5.3478	2.2781	-0.61	7.2237	9.5448
y	-6.8157	8.905	6.4267	-6.8917	-9.1785	-5.1277	-7.7996	-8.0692	0.3755

City	28	29	30	31	32	33	34	35
x	-1.1096	-9.3151	0.4754	-9.3672	5.0635	1.8318	6.4578	-3.4084
y	-9.5519	1.2297	7.1375	0.3382	3.4905	4.7435	3.9692	-9.544

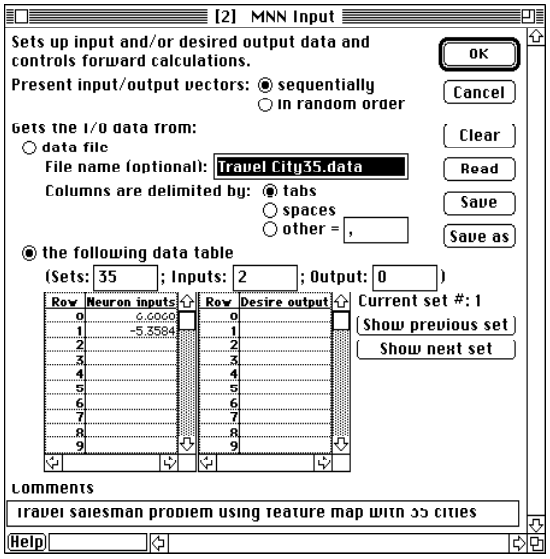
• The **Feature Map (2D to 1D)** block calculates and updates its weights for the winning neighborhood neurons.

• The **Cyclic Array Separator** block splits the weights into two arrays with x-coordinate and y-coordinate, respectively.

• The **XY Array Plotter** displays cities and the computed contour for result visualization.

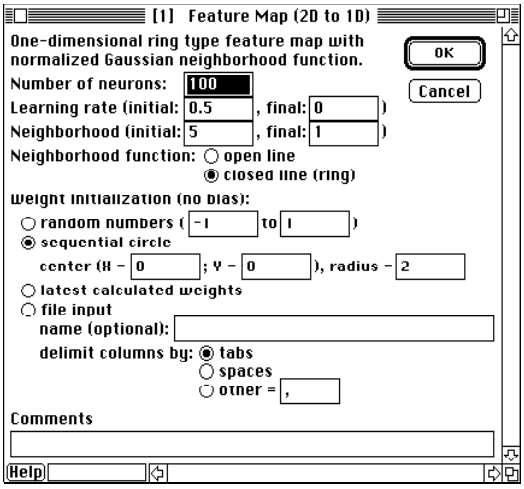
• The **Shift Register** block adjusts array length, since the **XY Array Plotter** block requires all inputs to have same dimension.

The city data stored in a text file can be read, viewed and modified in the **MNN Input** block dialog. The data will then be presented to connected network blocks one city at a time. In the dialog, we also decide that data will be outputted in random or sequential order in the simulation. In general, the order doesn't have significant effects on the feature map results, if the neighborhood function is adjusted properly as learning progresses.



In the **Feature Map (2D to 1D)** block dialog, users are allowed to freely choose the number of neurons, learning parameters and the weight initialization method. How many neurons is sufficient for handling the TSP with 35 cities is not clear. We might suggest that 35 neurons may be enough. However, depending on the city locations and initial conditions, it is possible that several neurons will converge to the same city. Therefore, it may be wise to choose a number of neurons that is

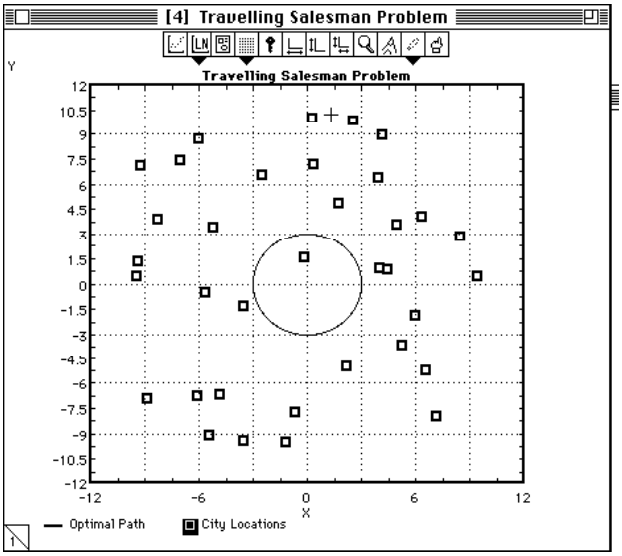
equal to or greater than twice the number of cities. Here is the **Feature Map (2D to 1D)** dialog. We will have 100 neurons in the feature map simulation.



One important aspect of implementing a feature map is that we would like the learning rate and neighborhood size of the network to be decreased gradually as simulation proceeds. In the dialog, we define both initial and final learning rates and neighborhood size for the feature map. Here, both initial and final learning rates must be numbers between 0 and 1, inclusively.

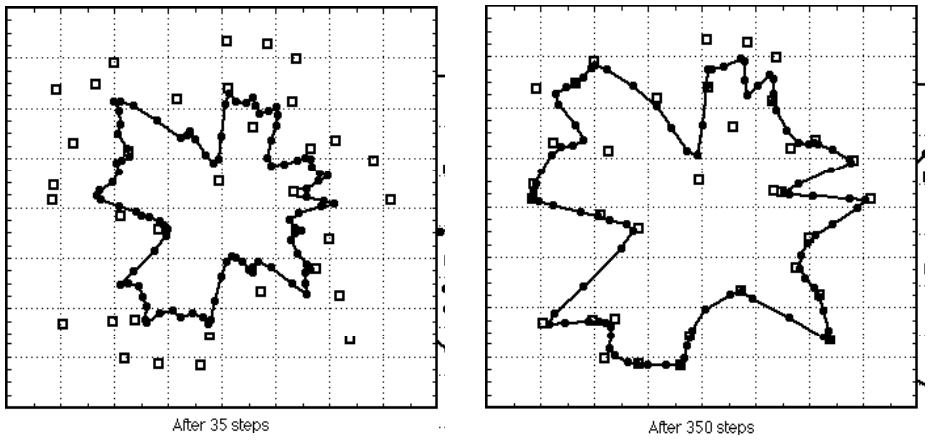
For this problem, we need a ring type neighborhood function, since we are looking for a shortest closed loop trip. Also, for the first run, we must set the initial weights to lie on a circle sequentially.

Here is a plot for initial network weights and city locations.

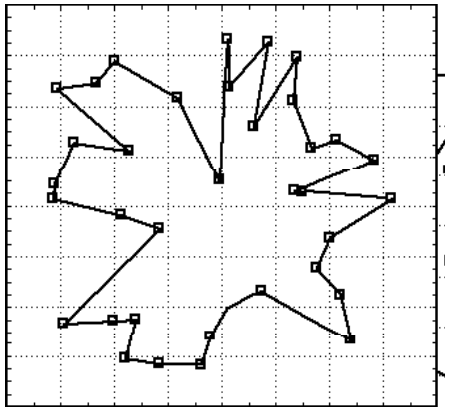


The initial weights are arranged sequentially around a circle centered at (0, 0) with radius 3.0. As an example, we choose the initial and final learning rates to be 0.5 and 0.01; and the initial and final neighborhoods to be 3 and 1, respectively. Typical results with these settings after 35 and 350 simulation steps are shown in the following figures.

Demo 24



As you see, the weight ring is gradually stretched out to fit the given cities. To fine tune the optimal traveling path, we then reset the initial neighborhood size to 1 and use the latest weights to continue the simulation. After another 350 steps, a near optimal traveling path is achieved.



Which parameters you should change

You may further explore this example by:

• Changing the number of neurons in the **Feature Map (2D to 1D)** block and trying to use a minimum number of neurons to achieve a near optimal traveling path.

• Using different initial and final learning rates and neighborhood sizes to see how they affect the speed of convergence.

• Placing the initial weight ring at different locations with different radii. Do the changes affect the performance of the feature map?

• Changing the settings in the **MNN Input** block to present the city coordinates in sequential order. Does this affect the convergence rate?

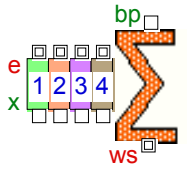
• Trying to solve problems with more cities.

Demo 25

Chap. 3 Block references

❒ Adaptive Weighted Sum [Edu]

Weighted summation computation and weight/bias updates.



Inputs	x_1, x_2, x_3, x_4: scalars, summation inputs and messages *bp* : scalar, error feedback from following block and message
Outputs	*ws:* scalar, weighted summation result and message e_1, e_2, e_3, e_4 : scalars, errors to connected blocks and messages

Description

This is the only block of NeuroLab's educational blocks that has a learning capacity. The **Adaptive Weighted Sum** block, together with activation function blocks, forms one neuron. We recommend users pay extra attention to this block while using the educational blocks to explore the structure and power of artificial neural networks.

This block performs weighted summation during forward calculation and updates weights/bias during backward calculation. The maximum number of inputs to this block is four. Each input to this block, *x*, has a corresponding error output, *e*, for back-propagation. The executions of the block are activated by messages instead of the simulation order number of the block. When the block gets messages from all connected *x* connectors, the forward calculation will be executed. In a forward calculation, the summation of bias and connected inputs multiplied by corresponding weights is calculated. On the other hand, the backward calculation is executed when the block receives a message from the *bp* connector. In the backward calculation, the block computes necessary corrections for the weights and bias using simple or momentum back-propagation, and updates the weights and bias accordingly. The *e* connectors, if connected, will send back-propagation errors (also computed in the backward calculation phase) to the connected blocks. Note that, for a meaningful calculation, paired connectors *x* and *e* attached to the same color should connect to the same block.

The two back-propagation methods are valid only when the *bp* connector is connected. If not *bp* is not connected, but a back-propagation method is selected, the block will give users a warning message and temporarily switch itself to the Hebbian rule with decay. If Hebbian with decay is selected, the information from the *bp* connector is ignored.

Dialog Choices

• *Learning method:* The choice of learning method to update weights and bias.

  • *momentum-delta:* The momentum back-propagation algorithm will be used when this radio button is selected.

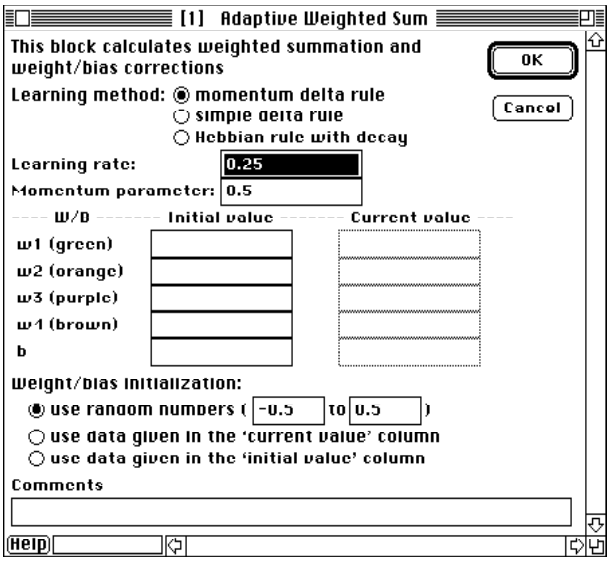
  • *simple-delta:* The block uses simple back-propagation.

  • *Hebbian with decay:* The Hebbian rule is used. This choice should be used when the block is connected to a non-differentiable activation function.

• *Learning rate*: The learning rate of the neuron. This parameter is required for all three learning methods.

• *Momentum parameter*: The momentum parameter of the neuron. This parameter is needed only when momentum delta is selected.

Demo 26



• *w1, w2, w3, w4, b (initial value):* The initial weights and bias of the neuron. The user can edit these values.

• *w1, w2, w3, w4, b (current value):* Displays updated weights and bias of the neuron during simulation.

• *Weight/bias initialization:* Choice of weight and bias initialization.

  • *Use random numbers:* The random numbers uniformly distributed in the range bounded by two given numbers.

  • *Use initial value column*: The numbers given in the initial value column. This initialization method can shorten the learning time if a set of good initial conditions is known.

  • *Use current value column*: The numbers given in the current value column. Users can choose this method to continue the learning process and hence gradually improve the performance of the network in different simulation runs.

Connectors

• *x (1-4):* Each value inputted from *x* is multiplied with its corresponding weight to get the weighted summation during forward calculation.

• *ws:* The weighted summation. A message is sent to all connected input connectors after forward calculation.

• *bp:* Receives error information and back-propagation message for backward calculation from a connected block.

• *e (1-4)*: Sends error information and back-propagation message to all connected input connectors to activate the back-propagation calculation of connected blocks.

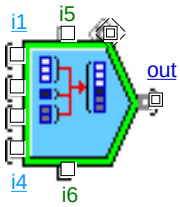
Example

The educational blocks simulate elementary components of a neural network. They provide an ideal tool for modeling small neural networks for illustration, presentation, and educational purposes. Here, we built a single neuron with two educational blocks: an **Adaptive Weighted Sum** and a **Sigmoid** blocks.

Demo 27

☐ Array Combiner [Mis]

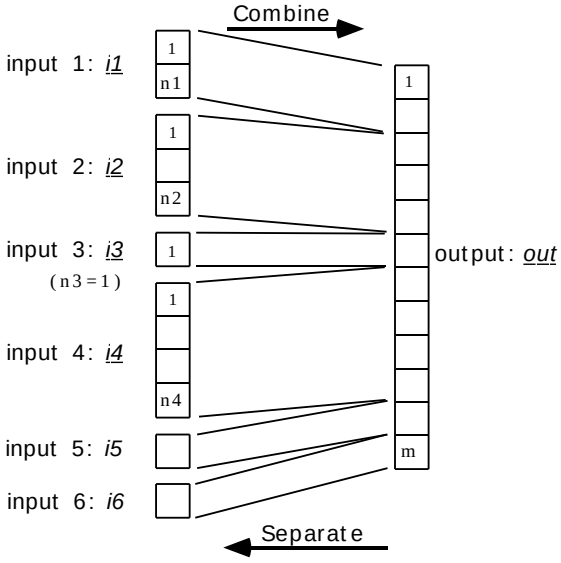
Combines arrays and scalars into one large array.

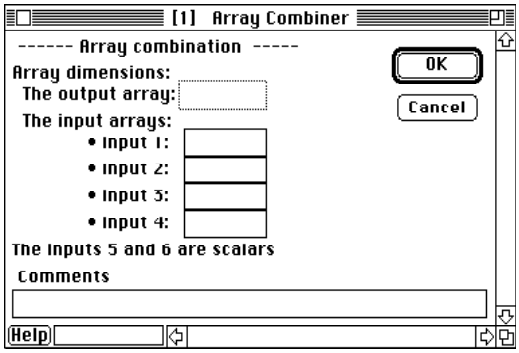


Inputs	*i1-i4:* arrays, to be combined into a large array *i5, i6:* scalars, to be combined into a large array
Outputs	*out:* array, combined large array *info:* array, dimension information

Description

The purpose of this block is to gain flexibility in data handling for scalars and arrays. This block combines four arrays and two scalar signals into one large array. Four array input dimensions are automatically detected if this block is connected to NeuroLab blocks. The total number of elements in the output array will be animated on this block during simulation. The order of elements in the combined array is array in *i1*, *i2*, *i3*, *i4* and scalar in *i5*, *i6*. The following diagram shows how input arrays and scalars are combined. The dimension of the output array is m = n1+n2+n3+n4+1+1 if all input connectors are connected. If some input connectors are not connected, their corresponding input dimensions are set at zero. The counterpart block which separates an array into several arrays is **Array Separator**.





Array combination and separation

Dialog choices

• *Output array dimension:* The dimension of the output array will be displayed.

• *Input array dimensions:* The dimension of each input array. If this block can not detect array dimensions, user have to enter values.

Connectors

• *i1* - *i4*: The input connectors correspond to the dialog choices *Input 1* through *Input 4*, with the top connector number 1 and the bottom connector number 4.

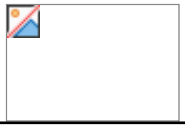
• *i5*, *i6*: The scalar inputs.

• *out:* The output is a combined array.

• *info:* The output diamond connector supplies dimension information to the **Array Separator** block for array separation.

❒ Array to Scalars (4) [Mis]

Gets up to four element values from an array and converts to scalars.

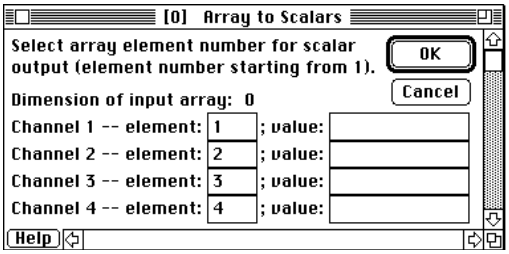


Inputs	*u* : array
Outputs	*n1, n2, n3, n4* : scalars, selected array elements

Description

Most block connectors in NeuroLab pass arrays during simulation. There are many cases where users will need to monitor a specific element of the passed arrays. For example, to plot variables in the neural networks, probably you need this block because the **Plotter I/O** block in Extend's ( )**Plotter Lib** or (ÿ) **PLOTTER.LIX** handles only scalars.

Dialog choices



• *Dimension of input array:* Shows the dimension of input array.

• *Channel 1*, *channel 2*, *channel 3*, and *channel 4*: Each channel corresponds to an output connector. For example *channel 1* corresponds to the *n1* connector.

- *Element*: The indices of array elements that will be passed to the output connectors.
- *Values*: Displays values of the selected elements.

Connectors

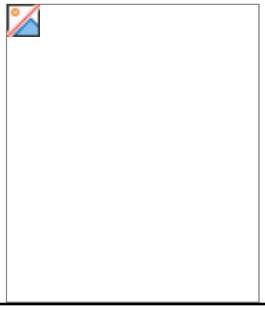
• *u*: Input array to this block.

• *n1*, *n2*, *n3*, and *n4*: Each connector, if connected, can output a chosen element of the input array.

❒ Binary Image Display [Mis]

Displays 1-bit (black & white) image in 2-D space.

Demo 29

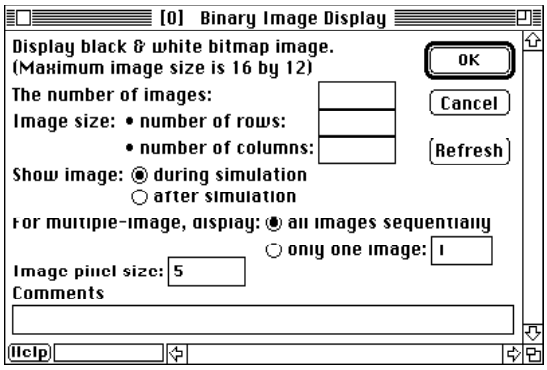
	Inputs	array, image matrix (maximum dimensions are 16 by 12)
	Outputs	array, image matrix

Demo 30

Description

This block displays black and white (1-bit) image with dimension of up to 16 rows by 12 columns. It can be used to display the input, output and base images of a Hopfield network. To see the image during simulation, users have to turn on “Show Animation” under Extend’s “Run” menu.

Dialog Choices



• *Number of images*: The number of images stored in the input array.

• *Image size*: Displays or sets the dimensions of the input images.

- *Rows*: Image height in pixels.
- *Columns*: Image width in pixels.

• *Show image:* Choose to update the image *during* or *after* simulation.

• *Multiple image display*: Specifies how multiple images will be displayed.

- *All*: Displays all input images sequentially.
- *Only one*: Displays only one selected images.

• *Image pixel size:* The number of pixels that will be used to display one image bit. For example, pixel size=5 means that the square area defined by 5 by 5 pixels in CRT is assigned for one bit of image.

Connectors

Both input and output connectors pass arrays which contain the same black and white images.

Binary Image File In		**[Mis]**
Reads a binary 2D image data from disk for Hopfield networks.		
	Inputs	None
	Outputs	*x:* array, image data

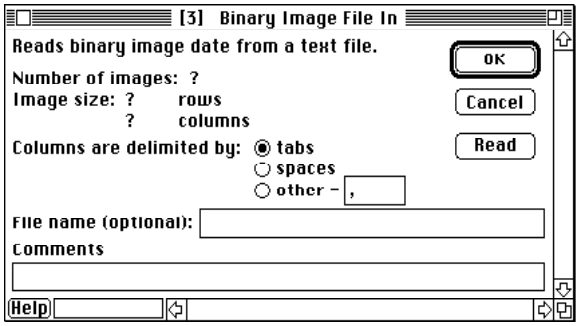
Description

This block reads a image data from disk files which stored information in text format. The **Hopfield** and **Boltzmann Machine** blocks use this block to input binary images. A binary image pixel (bit) can have only two states and their values +1 and –1 indicate the on and off of an image pixel, respectively. The text files that will be read by this block should have the following format:

Line 1: *Comments* (optional)
Line 2: *n* (integer)
Line 3: *r1* (integer) *c1* (integer)
Next r1 rows: Tabulated data (real number, *r1* rows, each row has *c1* column
Line r1+4: *r2* (integer) *c2* (integer)
Next r2 rows: Tabulated data (real number, *r2* rows, each row has *c2* columns)
etc.
Note: Data after non-numerical characters in each line are omitted

where *n* is the number of images stored in the file and, *r1*, *r2*, *c1*, and *c2* are the row and column dimensions of the input images, respectively.

Dialog Choices



• *Read:* Reads the specified data file. If the file name is unspecified before simulation or if the file cannot be found, a standard file-open dialog will be opened so that the user can select the proper data file.

• *Number of images:* Displays the number of data sets in the input file.

• *Image size:* Displays the row and column dimensions of the input image.

• *Column delimit*: A selection of the column delimit character.

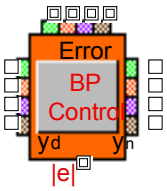
• *File name*: Name of the text file that stores the image data.

Connectors

• *x:* The array output connector. The block passes data containing binary images in array form at beginning of simulation.

❒ BP Control [Edu]

Computes output errors & activates back-propagation (for educational blocks).



Inputs	y_n: scalars, neural network outputs and messages y_d: scalars, desired outputs and messages
Outputs	*error*: scalars, network output errors and messages |e|: scalar, accumulated root-mean-square error.

Description

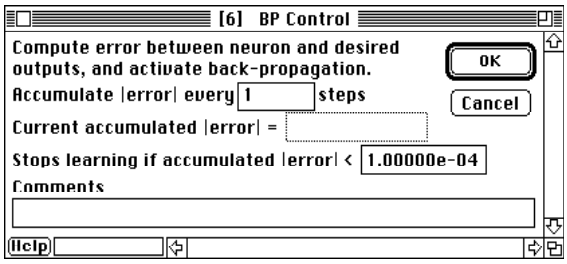
This block is designed to be used with educational blocks for controlling simulation orders. It will compute neural network output errors and activate backward calculation of the network when all corresponding y_n and y_d data are ready. The block can handle at most four outputs from output neurons of a network. For meaningful results, corresponding y_n, y_d and *error* connectors, which have the same color, should be connected accordingly.

Dialog Choices

• *Accumulate |error| every __ steps:* The number of simulation steps for accumulating root-mean-square error |*e*|. The plot of |*e*| will be smoother if the value is equal to the number of training I/O sets. If this number is one, current error is outputted.

• *Current accumulated |error|:* Displays the accumulated error.

• *Stop learning if accumulated |error| < :* You can specify the minimum value, at which it is considered that the network has learned the problem to a certain accuracy, to stop the simulation.



Connectors

All connectors of the block are scalar connectors.

• y_n*:* Four input connectors at the left of the block. These connectors receive computed results from the neurons.

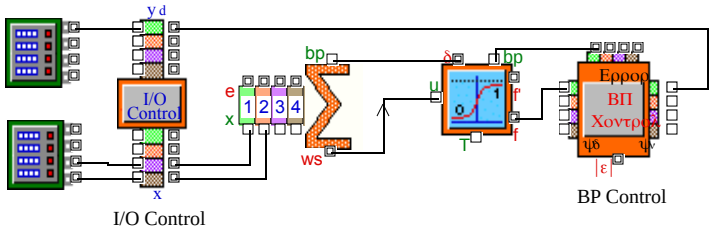
• y_d*:* Four input connectors at the right side of the block. These connectors receive the desired outputs of neurons. Usually, these connectors are connected to **I/O Control** block.

• *error:* The block calculates the difference between *yd* and *yn* (in the same color), and passes the result to the four corresponding *error* connectors. The error connectors then send messages to connected blocks to activate backward calculation.

• |*e*|: Accumulated root-mean-square error between the desired and network outputs. The steps for accumulation can be set in the block dialog.

Example

Demo 33



☐ BP Delta Sum [Edu]

Sums up the errors for back-propagation delta.

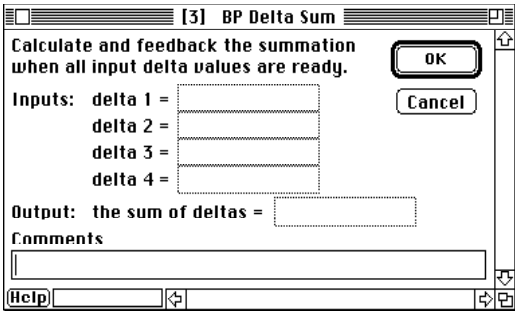
Inputs	δ: scalars, back-propagation errors and messages from connected function blocks
Outputs	*s:* scalar, summation of all input δ's and message

Description

This block is intended for use with educational blocks to sum up the errors for back-propagation when the model has multiple errors that will back-propagate to the same **Adaptive Weighted Sum** block (i.e. multiple neurons and multiple layers). This block is required for simulating multiple output networks with educational blocks.

Dialog Choices

The input value(s) and summation result are displayed in block dialog during simulation.



Connectors

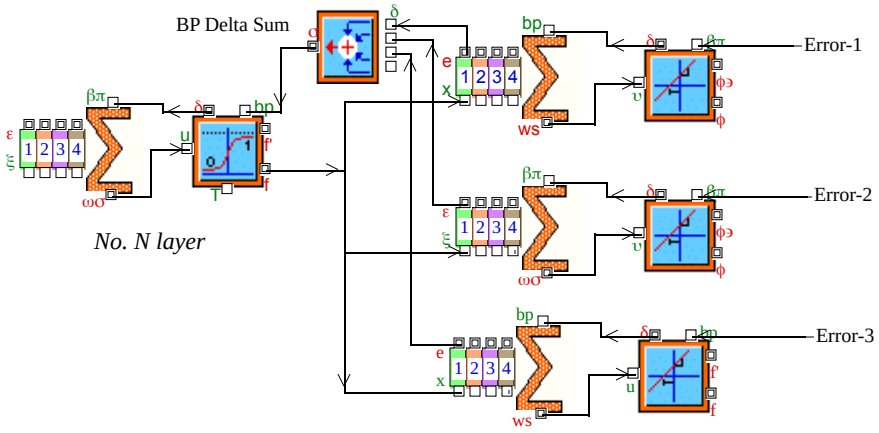
• δ: Back-propagation errors from the *e* connectors in the **Adaptive Weighted Sum** blocks.

• *s:* The summation of all input back-propagation errors. This connector should be connected to the *bp* connector in the transfer function blocks like **Sigmoid**, **Tanh**, **Linear**, etc**.**

Example

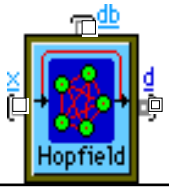
The following model is the part of muti-layer feed-forward network. The **BP Delta Sum** block accumulates back propagation errors and passes to the previous layer neuron.

Demo 34



❒ Hopfield [Net]

Hopfield network with discrete output neurons.



Inputs	*x:* array, probe data, initial states of neurons *db:* array, data base for setting of network weights
Outputs	*d:* array, network output, retrieved image

Description

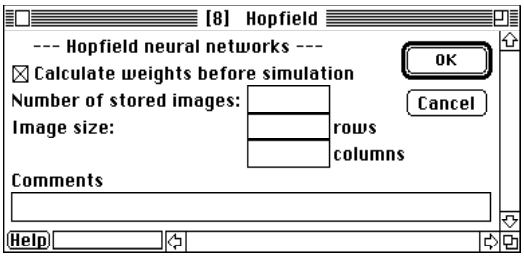
A Hopfield network contains McCulloch-Pitts neurons that are fully connected to each other without self-looping. The connections among neurons of a standard Hopfield network are symmetric. Operations of a Hopfield network can be divided into a storage phase and a retrieval phase. Strictly speaking, there is no training in a Hopfield network because its weights are prescribed and do not change during simulation.

A typical application of Hopfield networks would be to build a network that stores content-addressable data or information. For the image processing application, one neuron corresponds to one image pixel. Since this block uses a discrete activation function, the outputs of neurons are suitable for black and white images.

The weights between neurons are determined from the base data or images supplied from the *db* connector. The probe data are supplied from the *x* connector and set initial states (neuron states: fired or non-fired) of the network. During the simulation, the states of network evolve and gradually settle to some equilibrium point. However, the equilibrium point may or may not be exactly the pre-stored data or information.

Dialog Choices

Demo 35



• *Calculate weights before simulation*: If checked, the block uses the base images inputted from the *db* connector to calculate the network weights before simulation.

• Number of images: The number of images stored in the network for future retrieving.

• *Image size*: Displays or sets the row and column dimensions of the input data or images.

Connectors

All three connectors pass arrays.

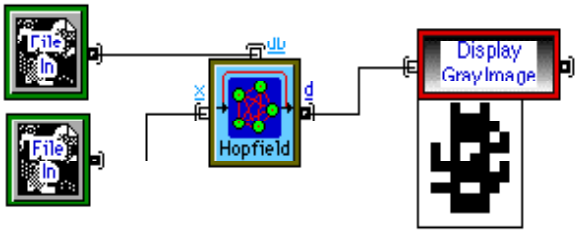
• *x*: The input image or pattern to be tested by the network. The data is also called “Probe image” which is the initial state of the network.

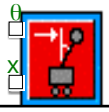
• *db*: The image database that will be used to set the network weights.

• *d*: The pattern or image corresponding to *x* that is retrieved by the network.

Example

This is the character recognition example discussed in Chapter 4.



☐ **Inverted Pendulum Animation**		**[Mis]**
Animates the motion of an inverted pendulum with position and angle.		
	Inputs	*x:* scalar, cart position (meter) θ: scalar, pendulum inclination angle (degree)
	Outputs	none

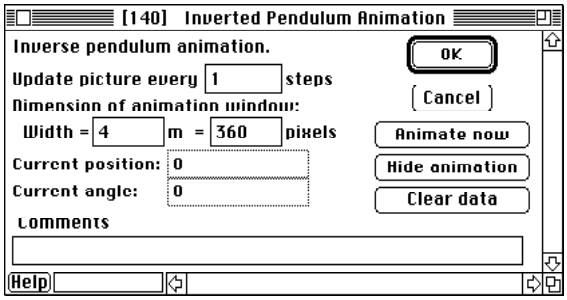
Description

The position and orientation of an inverted pendulum are supplied to the block for real time animation. The animation window is attached to the right side of the block. Pendulum inclination angle is limited to ±45°. Control of the inverted pendulum is considered to have failed when the inclination angle becomes greater than 45°. If control fails, the block generates a beep sound and then stops the simulation.

The pictures animated during simulation are stored in the resource file **NeuroLab.rsr** in the ( ) **Extension** folder of Extend or (ÿ) **EXTENSNS** directory of Extend. Users can replace this resource

file and modify the script in the block to make their own animation blocks using the same technique.

Dialog Choices



• *Update steps*: Number of simulation steps for which the animation picture will be updated periodically.

• *Animation window*: The size for the animation window is determined by the number of screen pixels. Then that window is assigned to the specified physical distance. If the width is assigned 4 meters, the animation window's horizontal limits correspond to -2 m and 2 m. The height of the window will be adjusted automatically to maintain the proper aspect ratio.

• *Current position*: Shows current cart position.

• *Current angle*: Shows current inclination angle of the pendulum.

• *Animate now*: Replays previous animation results without running the simulation.

• *Hide simulation*: Hides simulation window from current model window.

• *Clear data*: Clears all stored animation data.

Connectors

All input connectors pass scalars.

• *x*: Position of cart in meter.

• θ: Inclination angle of the inverted pendulum in degree.

Animation

To see the animation results during simulation, you have to turn on the "Show Animation" option under Extend's "Run" menu.

❒ I/O Control [Edu]

Controls and activates the forward calculation of networks (educational).

Inputs	*yd:* scalars, desired outputs *x:* scalars, network inputs
Outputs	*yd:* scalars, desired outputs and messages *x:* scalars, network inputs and messages

Demo 37

Description

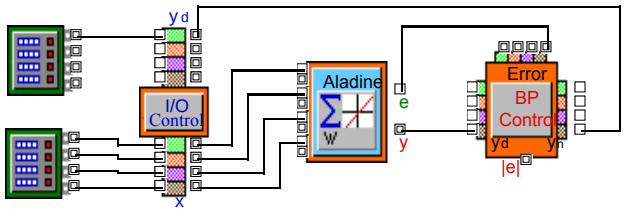
This is the block that should be used as an interface between networks built by educational blocks and their outside systems. The block collects all possible inputs and desired outputs, then passes the data to blocks that are connected to its output connectors and activates forward calculation in the network. The block can handle up to four inputs and four desired outputs. Usually, the y_d output connectors are connected to the **BP Control** blocks for computing output errors of the network.

Connectors

All connectors pass scalars. The four input connectors at the upper left portion of the block get desired output data from the outside system and pass it to the four output connectors at the upper right corner, accordingly. Similarly, the four input connectors at the lower left portion of the block get network input data from the outside system and pass it to the four output connectors at lower right corner.

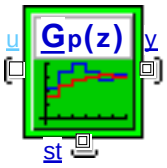
Example

Below is an adaline network with four inputs and one output.



❒ Lin Disc MIMO [Ctl]

Linear Discrete Multi-Input Multi-Output plant. Its dynamics are specified by a transfer function or first-order difference equations.



Inputs	*u:* array, plant inputs (dimension = plant inputs)
Outputs	*y:* array, plant outputs (dimension = plant outputs) *st:* array, internal states (dimension = system order)

Description

This is a linear multi-input multi-output (MIMO) discrete plant block (a discrete version of **Lin Cont MIMO** block). Plant dynamics can be specified by either a transfer function (TF, only applicable to single-input single-output (SISO) case when the input and output would be connected with a one element array) or a state space (SS) description. States are time-dependent internal variables whose dynamics are described by first-order difference equations. The plant order is the number of first-order difference equations. The maximum plant order is 6. All connectors require array type signals.

The general form of the TF description is expressed as a ratio of polynomials in the sampling operator z (compare with the Laplace operator s in the continuous case).

$$\frac{Y(z)}{U(z)} = \frac{b_n z^n + \ldots + b_1 z + b_0}{a_n z^n + \ldots + a_1 z + a_0}$$

Here b_j, a_j are coefficients of the j th power of z in the polynomials. The operator z means one time step advance (one sampling period ahead). If you divide both numerator and denominator by z^n, and express the reciprocal of z^{-j} (i.e. j steps delay), you can rewrite the transfer function into a difference equation.

State space representation is expressed in matrix form of difference equations.

$$\underline{\Xi}(\kappa+1) = \underline{\Phi}\,\underline{\Xi}(\kappa) + \underline{\Gamma}\,\underline{Y}(\kappa)$$
$$\underline{Y}(\kappa) = \underline{X}\,\underline{\Xi}(\kappa) + \underline{\Delta}\,\underline{Y}(\kappa)$$

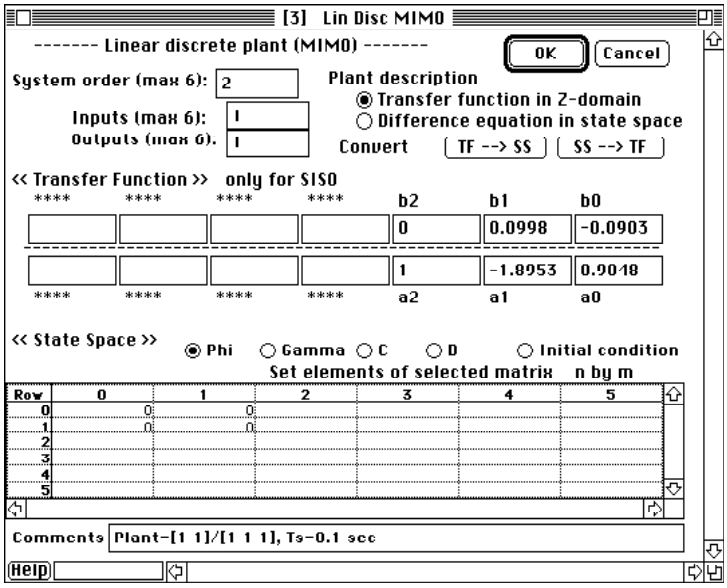
Where $\underline{\boldsymbol{\Phi}}$, $\underline{\boldsymbol{\Gamma}}$, $\underline{\boldsymbol{C}}$ and $\underline{\boldsymbol{D}}$ are system matrices whose dimensions depend on the numbers of inputs, outputs, and the system order of the plant. $\underline{X}$(k), $\underline{Y}$(k) and $\underline{U}$(k) are states, outputs and inputs at the k-th step, respectively.

Dialog choices

• *System order:* Plant order (maximum is 6).

• *Inputs:* Number of plant inputs (maximum is 6).

• *Outputs:* Number of plant outputs (maximum is limited to 6).



• *Plant description:* Choose the plant description from transfer function (TF) or state space (SS). Transfer function description is only applicable to the SISO case.

  • *Convert: TF-->SS:* Converts the plant dynamic description from TF to SS type. This conversion is achieved by using a control canonical form.

  • *SS-->TF:* Converts SS to TF description. A controllability matrix is used for this conversion and its singularity is checked for proper conversion.

• *Transfer Function:* The coefficients of the transfer function polynomial using the sampling operator z. Accessible coefficient boxes are determined by the plant order. If the plant is second order, the applicable coefficient boxes are b2, b1, b0, a2, a1 and a0. Boxes on the left side always correspond to higher power of z.

• *State Space:* Four system matrices and the initial conditions of the plant are typed into the table by selecting the radio buttons.

• *Phi, Gamma, C, D matrices:* System matrices correspond to first order difference equations. Each matrix's appropriate dimension is displayed above the upper right corner of the table. Typed-in values are accepted only inside of appropriate dimension. Note that the table index starts at 0 instead of 1.

• *Initial condition:* This column vector is used for the initial condition of the plant states.

Connectors

• *u:* The plant inputs. The array dimension should be equal to the input number. If this is not connected, input values of zero are assumed.

• *y:* The plant outputs. The array dimension is the output number.

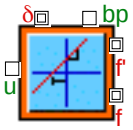
• *st:* The internal states of the plant. The array dimension is the plant order.

Example

This is a simulation set-up for the **Lin Disc MIMO** block. Notice that the connectors for the input and output of the plant require array type connections. Please use the **Array to Scalars** and the **Scalars to Array** blocks for array to scalar and scalar to array conversions.

❒ Linear [Edu]

Linear transfer function specified by slope and offset.



Inputs	*u* : scalar, function input and message *bp* : scalar, output error and message
Outputs	*f* : scalar, function output and message *f'* : scalar, function derivative δ : scalar, back-propagation delta and message

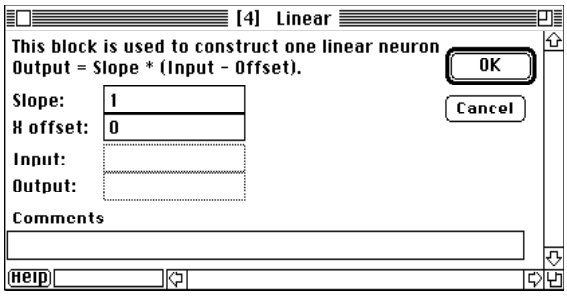
Description

A linear relationship, f(*u*) = slope*(*u*-offset), is implemented in the block. The block performs two types of operations, forward and backward calculations. The executions are activated by messages instead of the simulation order number of the block. In the forward calculation, the block outputs the function value f(*u*) and its derivative f'(*u*). The δ connector outputs the back-propagation delta to the **Adaptive Weighted Sum** block for back-propagation learning.

When the *u* connector is NOT connected to the **Adaptive Weighted Sum** block, the block follows the simulation order assigned by Extend and executes only forward calculation.

Dialog Choices

Demo 40



• *Slope*: The slope of the linear function.

• *Offset*: The function crosses the x-axis at this value.

• *Input*: Displays the input of the function.

• *Output*: Displays the output of the function.

Connectors

All connectors pass scalars.

• *u*: The function input. Also, the received message activates the forward calculation.

• *f*: The function output. The message is sent after the forward calculation.

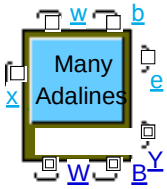
• *f'*: Calculated function derivative (same as the slope).

• *bp*: The output error from down-stream block. The received message activates the backward calculation which updates weights and biases.

• δ: The value to be passed to the **Adaptive Weighted Sum** block for back-propagation learning. The message is sent after the backward calculation.

❒ Madaline [Net]

Many adaptive linear elements.



Inputs	*x* : array, inputs of the network and message *e* : array, output errors and message *w* : array, initial weights in matrix form *b* : array, initial biases
Outputs	*Y* : array, outputs of the network and message *W* : array, current weights in matrix form *B* : array, current biases

Description

A Madaline block contains a layer of adaptive linear elements that are trained by the normalized Least Mean Square (LMS) learning rule.

Dialog Choices

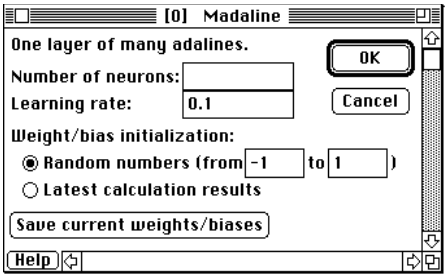
• *Number of neurons*: Specifies the number of adalines in the block.

• *Learning rate*: Specifies the network's learning rate.

• Weight/bias initialization: Select a method for the initialization.

• *Random numbers*: Initializes initial weights and biases of the network as random numbers bounded by two values that can be specified by users.

• *Latest results*: Uses current network weights and biases as initial condition for the simulation.



Connectors

• *x*: Inputs array to the network. All inputs are fully connected to neurons in the block.

• *e*: Errors between the desired and computed outputs. The network will be trained by the LMS rule if the connector is connected. Otherwise, decayed Hebbian learning will be executed.

• *Y*: Network output array that is equal to weighted summation over the network inputs.

• *w*: Inputs initial weights to the block if connected.

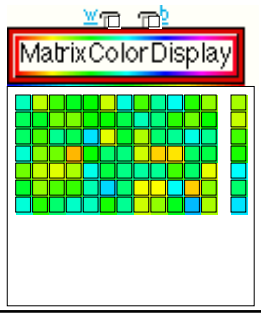
• *b*: Inputs initial biases to the block if connected.

• *W*: Outputs current weights for display.

• *B*: Outputs current biases for display.

Matrix Color Display **[Mis]**

Displays matrix element values in continuous colors. One square box represents a matrix element and its normalized value.



Inputs	*w* : array, network weight matrix *b* : array, network bias array
Outputs	none

Description

The weights and biases of neurons are essential parameters in understanding the relative significance of each neuron in the network. The library provides several ways for the user to examine the weights and biases of each neural layer. This block uses Extend's color animation feature to display the relative magnitude of weights and biases in color. Each square box represents one weight or one bias. The blue and red colors represent the specified lower and upper limit values, respectively. The intermediate color is generated by interpolation if the value is between these two specified values.

In general, this block can be used to display any matrices and arrays. However, it is used to display current weights and biases of a neural layer most of the time. When the block is connected to **MNN**

Hidden Layer and **MNN Output Layer** blocks, it has the ability to set the input matrix dimensions automatically (in this case, the row number corresponds connected layer's neuron and the column number corresponds previous layer or input).

Users who build their own blocks and would like to use the auto-dimension function should have their blocks send out the special array containing the dimensional information (see Appendix D).

Dialog Choices

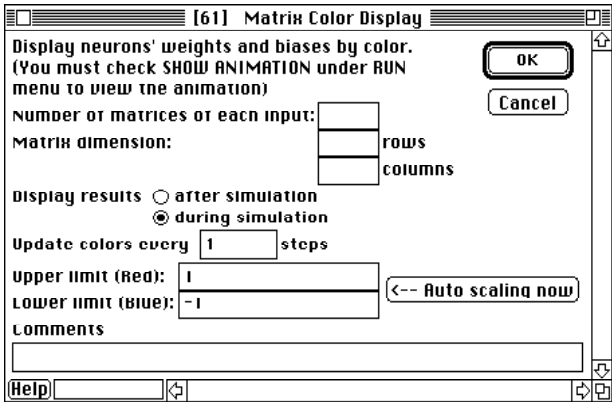
• *Number of matrices*: The number of matrices in the input array.

• *Matrix dimension*: Displays or sets the dimensions of the input matrix.

• *Display after simulation*: Displays the result only at the end of simulation.

• *Display during simulation*: Updates the color display during simulation.

• *Update colors every _ Steps:* When "display during simulation" is selected, the block updates the color display at every specified number of simulation cycles.



• *Upper limit (red)*: If the element value is greater or equal to this value, a red color square is displayed at corresponding location.

• *Lower limit (blue)*: If the element value is smaller than or equal to this value, a blue color square is displayed at the corresponding location. If the element value is between the above two values, the color is generated by interpolation.

• *Auto scale*: Sets the upper and lower limit of the display according to the maximum and minimum elements of the input matrix when this button is clicked.

Connectors

Both connectors pass arrays.

• *w*: In general, this array connector inputs a two-dimensional matrix. Most of the time, it is connected to the current weight matrix of a feed-forward neural layer.

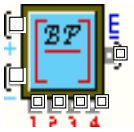
• *b*: In general, it can be connected to any vector, but in NeuroLab applications it is connected to the current bias array of a feed-forward neural layer.

Animation

Colors will change according to values of the weight matrix and bias array. Maximum values are displayed as pure red and minimum values are pure blue.

☐ MNN BP Control [Net]

Calculates output errors and activates back-propagation.



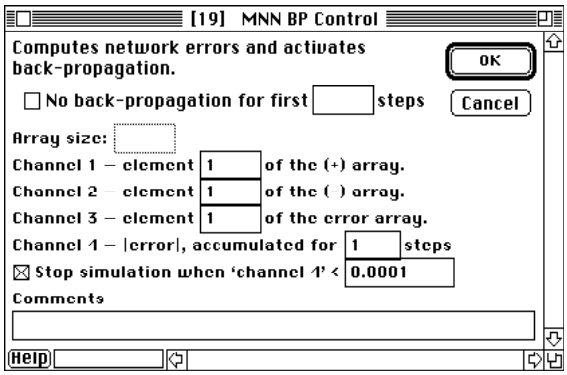
Inputs	+ : array, desired network outputs and message – : array, calculated network outputs and message
Outputs	*E* : array, output errors and message *1* : scalar, specified element of the (+) array *2* : scalar, specified element of the (–) array *3*: scalar, specified element of the error array *4* : scalar, the accumulated RMS error

Description

The **MNN BP Control** block takes in two arrays from the + and – input connectors, computes the difference between the two arrays, and passes the result to the *E* connector. In addition, users can choose to output specified element in the +, – and error arrays, as well as the accumulated error, by using the four value connectors at the bottom of the block.

When the block is applied in a multi-layer feed-forward neural network, its main function is to compute the error between the network and desired outputs. In this case, the + connector is connected to the desired outputs and the – connector is connected to the network outputs. The output error array is then computed and directed back to the *e* connector in the **MNN Output Layer** block or other feed-forward network blocks which will activate the network's back-propagation learning process.

Dialog Choices



• *No back-propagation for ? step:* This option forces the back-propagation errors to be zeros during the first specified simulation steps.

• *Array size:* Displays the dimension of the input arrays.

• Channel 1, 2, 3 - element _ of: Indices for the elements of the +, – and error arrays to be passed through the *1*, *2* and *3* output connectors.

• *Steps for |error| accumulation:* The simulation cycle number for accumulation of RMS error (root mean square of error array).

• *Stop simulation when 'channel 4' is less than:* If checked, the simulation will be stopped when the accumulated RMS error is less than that specified value.

Connectors

The connectors on the left and right hand sides of the block pass arrays. The four connectors at the bottom of the block pass scalars.

• + and –: Two arrays are passed into this block through these two connectors. The two arrays must have the same dimension. If both connectors receive the messages, then the difference is calculated.

• *E*: The array subtraction result is passed through this connector. The message is sent to connected blocks after the calculation.

• *1*: Selected element of the + array.

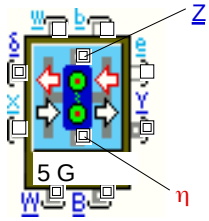
• *2*: Selected element of the – array.

• *3*: Element of error array.

• *4*: The accumulated RMS error.

❒ MNN Hidden Layer [Net]

A hidden layer in a multi-layer feed-forward network.



Inputs	*x*: array, layer input and message *e* : array, back-propagation error and message *w* : array, initial layer weights *b* : array, initial layer biases
Outputs	*Y* : array, layer outputs and message δ: array, back-propagation error and message *W* : array, current layer weights *B* : array, current layer biases η: scalar, current learning rate *Z:* array, activities of neurons

Description

A typical multi-layer feed-forward neural network contains an input layer, several hidden layers, and an output layer. The network is connected to the outside world through the input and output layers. This block represents one layer of hidden neurons in a multi-layer feed-forward neural network. The number of neurons must be specified before simulation is started. The back-propagation function will work when it is turned on in the **MNN Output Layer** block and the *e* connector is connected so that the block can get the back propagation error signal from its following layer. Note that the forward and back-propagation calculations for the entire network are activated by messages and are controlled by the **MNN Input**, the **MNN Output Layer** and the **MNN BP Control** blocks. The simulation order, which is assigned by Extend, of blocks inside the neural network is insignificant for the simulation result.

Dialog Choices

• *Neuron number*: The number of neurons in this hidden layer.

• *Learning rate*: The learning rate (η) for the neurons in this layer (default value: 0.25).

• *Momentum parameter*: The momentum parameter (α) for neurons in this layer (default value: 0.5). This parameter works only when momentum method is selected.

• *Activation function:* A choice of transfer functions for the neurons from sigmoid, hyperbolic-tangent and Gauss functions.

  • *T/D*: The temperature of sigmoid and hyperbolic-tangent function, or standard deviation for Gauss function.

• *Back-propagation methods*: Select a back-propagation method applied in this hidden layer. Note that the block will perform back-propagation only when the *back-propagation switch* in the **MNN Output Layer** block is turned on.

For a simple learning algorithm,

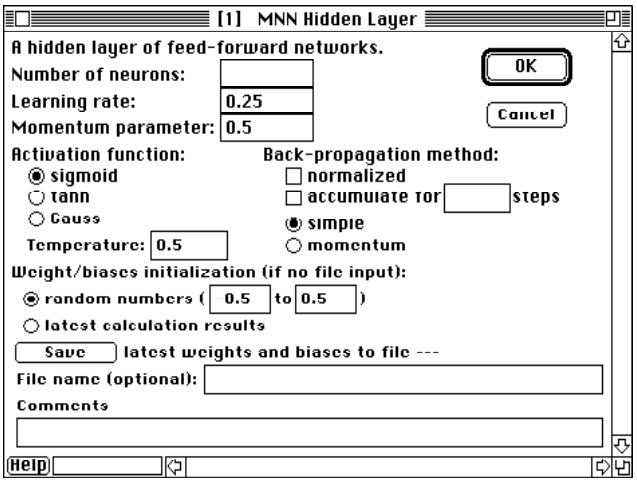
$$\Delta w_{ij} = \eta \ \ \delta_i \, u_j \, f_h'$$

For a momentum learning algorithm,

$$\Delta w_{ij}(k) = (1 - \alpha)\, \eta \ \ \delta_i \, u_j \ \ f_h' + \alpha \, \Delta w_{ij}(k-1)$$

The check box options, *normalized* and *accumulate* can be turned on individually so that there are eight combinations for the learning method.

• *Weight/bias initialization*: The method selected here will be used to initialize the initial weights and bias for this block when $\underline{w_i}$ and/or $\underline{b_i}$ connectors are not connected. Currently, there are two initialization algorithms to choose from. One is to set the initial values as uniformly distributed random numbers between two specified values; the other is to use the last simulation results to continue the simulation. The initialization method is omitted when $\underline{w_i}$ and/or $\underline{b_i}$ connectors are connected.



• *Save*: Saves current weights and biases of the layer to a text file. The file format is listed in **Weight/Bias File In** block.

• *File name*: The file name that will be used to save current weights and biases when the *Save* button is clicked. The path name must be added before the file name if you want to save the file to other than the current directory. If the file name is not specified, the standard file save dialog is displayed and requests your response.

Connectors

The η connector passes a scalar. All other connectors are array connectors.

• $\underline{x}$: The block gets its input array from this connector. The inputs are fully connected to neurons in the block. The received message activates the forward calculation.

• $\underline{\delta}$: The back-propagation delta calculated in this layer is sent to the previous layer through this connector. The message is sent after the backward calculation.

• $\underline{Y}$: The layer outputs. The message is sent after the forward calculation.

• $\underline{e}$: The block uses this connector to get the error and other necessary information from the following layer so that back-propagation can proceed. The received message activates the backward calculation.

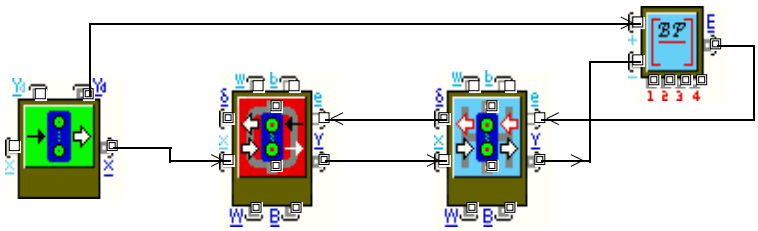
- *W*: If connected, the block sends out current weights during simulation.
- *B*: If connected, the block sends out current biases during simulation.
- *w*: If connected, the block will get initial weights from this connector before simulation.
- *b*: If connected, the block will get initial biases from this connector before simulation.
- η: Current learning rate of this hidden layer.
- *Z:* Current activity levels of this hidden layer neurons.

Animation

Displays neuron numbers and neuron nonlinearity in the block icon.

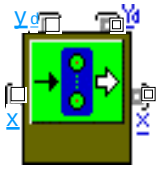
Example

The model below shows the basic structure of a feed-forward network.



☐ MNN Input [Net]

Input units that control forward calculation in a multi-layer networks. Data for network is supplied from connector, data file or data table in the dialog box.



Inputs	*x*: array, network inputs y_d : array, desired network outputs
Outputs	*X*: array, inputs to downstream networks and message Y_d : array, desired output corresponding to *X* and message

Description

This block is an important interface between networks built by ( )**NeuroLab Net Lib**, (ÿ)**NLAB_NET.LIX** library's blocks and other blocks. The block processes input and desired output information for multi-layer feed-forward neural networks. The important information of data is the number of I/O data sets and the dimensions of input and output arrays. The block operates in two ways, depending on how it is applied or connected in the model. If the array input connector *x* is not connected, the information can be read from a data file or from editable data tables in the block dialog. If the input connector *x* is connected, this block resets the number of data sets to one and simply passes the input array from *x* to *X* and passes the desired output array. In this case, the block will check the number of inputs and outputs and display on the icon. The dimensions of the input and output arrays cannot exceed 512.

The block also activates and controls the execution of the forward calculation by sending a message through *X* connector to the **MNN Hidden Layer** or the **MNN Output Layer** blocks.

Dialog choices

Demo 47

• Passes input/output data: Select one of the data sequences from *sequentially* or *in random order*. This option has meaning only there are more than one data sets. Sometimes, random data presentation gets faster leaning.

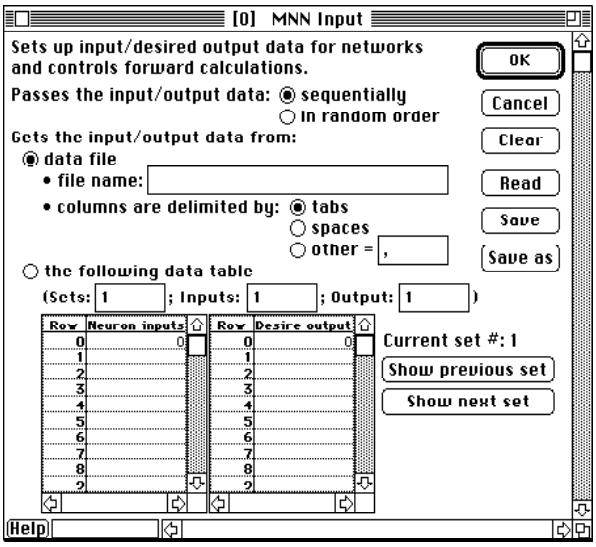
• *Clear*: Clears all entries in the dialog box (data table, file name and other variables).

• *Read*: Reads input and desired output data from a file. The data file must be saved in ASCII format with items being separated by the *column delimiter*. The first 2 lines contain the dimensional data information. From line 3, each line in the file contains one set of input-output data. Format summary is

Line 1:	// Comments (optional)¶
Line 2:	nSet, nInput, nOutput¶ (3 integers - the number of I/O data sets, inputs and desired outputs, separated by the *column delimiter*)
Line 3:	The first set of I/O data; first inputs, then outputs¶ (nInput+nOutput real numbers, separated by the *column delimiter*)
Line 4:	The second set of I/O data
.....	

• *Save*: Saves the comments, the number of data sets, inputs and outputs, and data in input and desired output tables to a file with the name specified in *file name*.

• *Save as*: The same operation as *Save* button except to save to the different file. A standard file dialog window is opened for the user to enter the file name.



• *Get I/O data from:* Choose I/O data source from a disk data file or the data table in the dialog box. If the input connector *x* is connected, this selection has no meaning.

• *data file*: Data for the network is supplied from the disk file.

• *File Name*: File path name to which the data is read or saved. If you enter a path name for Macintosh, you can specify folders by using colons (such as “My disk:Data folder:Data file”); otherwise, current path is used. For windows, the path name is ”MYDISK\DATADIRC\DATAFILE”.

• *Column delimiter*: The character which delimits the columns in the data file.

• *data table:* Data for the network is supplied from the data table in the dialog box.

- *Sets:* The number of data sets of input and desired output relationships. If the connector $\underline{x}$ is connected (often happens in the applications with networks), the number will be set to 1 automatically.
- *Inputs:* The number of inputs to the network.
- *Outputs:* The number of outputs of the network.

• *Current set #:* Indicates which I/O data set is being shown in the tables of neuron inputs and desired outputs.

• *Table of neuron inputs*: A data table with 512 rows in which users can check and modify the neuron inputs. In simulation, the table is updated in every step to track current inputs to the neural network.

• *Table of desired outputs*: A data table with 512 rows in which users can check and modify the desired outputs corresponding to each set of neuron inputs. In simulation, the table is updated in every step.

• *Show previous set*: Displays previous set of data in the tables.

• *Show next set*: Displays next set of data in the table.

Connectors

• $\underline{x}$: If connected, the received data is passed to the connected network through the X connector. In this case, data source selection is overridden.

• $\underline{X}$: Input data to the connected neural network blocks. The message is also sent to the connected blocks.

• $\underline{Y_d}$: Desired output data to **MNN BP Control** block to calculate the output error for back-propagation learning. The message is sent at every simulation step.

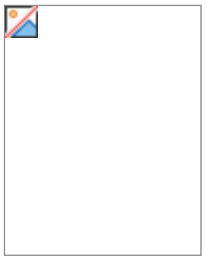
• $\underline{y_d}$: Desired output data from the outside of the networks. The received data is passed to the output connector $\underline{Y_d}$.

Animation

The icon indicates the number of data sets and dimensions of input and output arrays.

❒ MNN Output Layer [Net]

An output layer in a multi-layer feed-forward network.



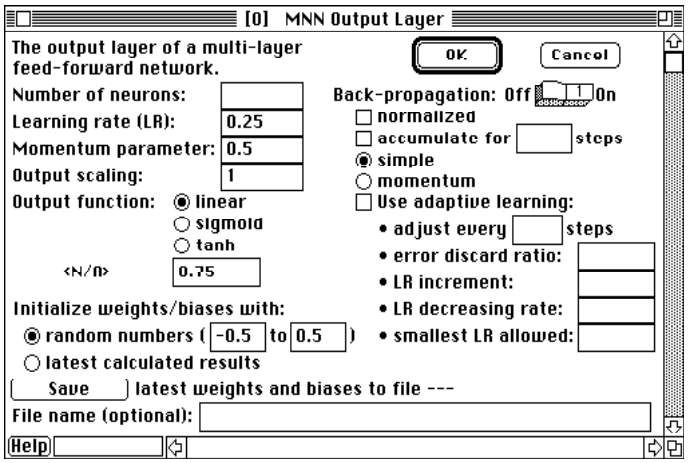
Inputs	$\underline{x}$: array, layer inputs and message $\underline{e}$: array, output error from **MNN BP Control** block and message $\underline{w}$, $\underline{b}$: array, initial layer weights and biases
Outputs	$\underline{Y}$: array, layer outputs and message $\underline{\delta}$: array, back-propagation delta and message $\underline{W}$, $\underline{B}$: array, current layer weights and biases η : scalar, current learning rate $\underline{Z}$: array, activities of neurons

Description

This layer contains all output neurons in a multi-layer feed-forward neural network. Like the **MNN Hidden Layer** block, this block performs both forward and backward calculations which are activated by the messages at each simulation step. Moreover, this block controls the execution of

back-propagation. The execution and parameters of the adaptive learning function are also set in this block. To ensure that the network works correctly, connector *e* should be connected to the **MNN BP Control** block.

Dialog Choices



• *Number of Neurons:* The number of neurons in the output layer.

• *Learning Rate:* Learning rate of the output layer. The value may change during learning if the adaptive learning option is on.

• *Momentum parameter:* Momentum parameter for back-propagation. This parameter doesn't change during adaptive learning.

• *Output scaling:* The scaling factor of the output function (default value is 1.0). This scaling is sometimes useful to use functions whose outputs are limited to fit into the real models which require certain output ranges.

• *Output Function:* Activation function of output neurons (sigmoid, hyperbolic-tangent or linear).

• *Back-propagation switch:* Performs back-propagation learning while switch is on. Otherwise, only a forward calculation occurs. Turning off the back-propagation switch is equivalent to a zero learning rate. However, the computation will be much faster since no redundant codes will be executed during simulation.

• *Back-propagation method:* Choice of *simple* or *momentum* method. Each method can be combined with accumulated and/or normalized algorithms. In total, there are eight methods to choose from.

• *Use adaptive learning:* If checked, the learning rate of the network will adjust according to the output error changes.

  • *adjust every ? steps:* The number of simulation cycle for updating learning rate. The error is accumulated during specified period and is used for judgment of update.

  • *error discard ratio:* The old weights and biases will be restored when an increasing output error exceeds this ratio. Default value is 1.04.

  • *LR increment:* The learning rate is increased by this increment when the accumulated error is decreased. Default value is 0.01.

  • *LR decreasing rate:* The learning rate is decreased by this ratio when the rate of error increase is greater than the discard ratio. Default value is 0.8.

  • *smallest LR allowed:* The minimum learning rate that the adaptive learning method can reach.

• *Weight/bias initialization*: The method selected here will be used to initialize the initial weights and biases when *w* and/or *b* connectors are not connected. There are two initialization algorithms. One is to set the initial values as uniformly distributed random numbers between two specified values; the other is to use the last simulation results to continue the simulation. The initialization method is omitted when *w* and/or *b* connectors are connected.

• *Save*: Saves current weights and biases of the layer to a text file. The file format is listed in **Weight/Bias File In** block.

• *File name*: The file name to save current weights and biases when the *Save* button is clicked. The path name must be added before the file name if you want to save the file to other than the current directory. If the file name is not specified, the standard file save dialog is displayed and requests your response.

Connectors

The η connector passes a scalar. All other connectors pass arrays.

• *X:* Input array from previous layer. The received message activates forward calculation.

• *Y:* The computation results of the neural network. The message is sent after the forward calculation.

• *e:* The output error from the **MNN BP Control** block. The received message activates backward calculation.

• δ*:* The block sends back-propagation delta and message to previous layer for back-propagation. The message is sent after the backward calculation.

• *w:* If connected, the block gets initial weights from this connector.

• *b:* If connected, the block gets initial biases from this connector.

• *W* : Current neuron weights.

• *B* : Current neuron biases.

• η: Current learning rate of the output layer.

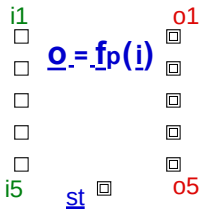
• *Z:* Activities of neurons.

Animation

Displays number of neurons and neuron nonlinearity in the block icon.

❒ N/L Cont Plant [Ctl]

Nonlinear Continuous Plant. Its dynamics are specified by first-order nonlinear differential equations.



Inputs	*i1 - i5:* scalars, plant inputs
Outputs	*o1 - o5:* scalars, equation outputs (corresponding to the equation number) *st:* array, internal states (dimension is plant's order)

Demo 51

Description

The plant dynamics are described by the first-order nonlinear differential equations which use state and input names. States are time-dependent internal variables whose dynamics are described by differential equations. Up to five equations, including possibly five first-order nonlinear differential equations, are calculated. Each equation is compiled before the simulation to speed up the calculation. The equations are typed-in to dialog boxes by the user. Differential equations are distinguished by the state name with the keyword “DOT” in the equations. The result of each equation is passed by the corresponding output connectors *o1* through *o5* on this block. The plant order is equal to the number of first-order differential equations. Usually the plant order is equal to the equation number but when it is smaller than the equation number, the user can enter non-dynamic equations. For example, if the equation number and the plant order are specified 5 and 3, then the first 3 equations should be differential equations and the following 2 equations should be non-dynamic equations.

These equations must be in the form: Result = formula; (don’t forget a semi-colon at the end of the equation). The user can use Extend's built-in operators and functions, and some or all of the input variables and state variables as part of the equations. The maximum variable names used in the equations are 5 inputs and possibly 5 state variables. If the left side of the equation is specified "xyzDOT" and "xyz" is a state variable name, this equation is treated as a first order nonlinear differential equation, otherwise it is considered a non-dynamic nonlinear equation. The differential and non-dynamic equations are

$$o_i = \frac{ds_i}{dt} = f(i_1,i_2,i_3,i_4,i_5,s_1,s_2,s_3,s_4,s_5)$$

$$o_i = f(i_1,i_2,i_3,i_4,i_5,s_1,s_2,s_3,s_4,s_5)$$

where o_i, i_j, s_k and f() are *i*-th output, *j*-th input, *k*-th state and a user-specified equation, respectively.

Each input and each state must be named in order to use it in the equation. You can use the default input value names (x1, u1, etc.) or specify new names. If a specified input is used in at least one equation, and its connector is not connected to another block, this block will warn you and stop simulation.

Extend's operators are:

> +, -, *, /, ^ (exponentiation), MOD or % (modulus), AND or &&, OR or ||, NOT or !, == (equals), != or <> (not equal), <, <=, >, >=

The *Function* buttons in the dialog display a listing of all built-in functions and procedures, including placeholders, to remind you of which arguments are required. You can copy functions from the listing to your equation with the "Copy" and "Paste" commands. You must replace the placeholders with the real arguments, and separate arguments with a comma (,). The parentheses are required to show where the arguments begin and end. User-defined functions are not allowed.

Dialog Choices

Similar to the Extend's programming script “ModL”, variable names, state names and operators are case insensitive so that SIN, sin, Sin are interpreted the same.

• *Number of equations:* Total number of equations. The maximum number is 5.

• *System order:* Number of differential equations. This value must be smaller than or equal to the number of equations. The maximum number is also 5. The same number of states’ names have to be typed-in.

• *Integration method:* Choice of three kinds of integration methods (Runge Kutta 4th order, modified Euler and simple Euler methods) are available.

• *Check&Compile:* All variables and equations typed in the editable box are checked and compiled. If there is an error, the error message will appear in the error dialog box. This action is automatically taken when the dialog box is closed.

• *Function by Alphabet, Function by Type:* Clicking these buttons opens a list of all of Extend's functions, including placeholders for their arguments. The function lists are organized by alphabet or by type. You can copy the function from the list into the equation. User-defined functions are not supported.

• *Input 1 - 5:* The names assigned to the variables at the 5 input connectors, from top (1) to bottom (5). You can use the default dialog names (u1, u2, etc.), or type in your own. These names can be used in the equations.

• *State 1 - 5:* State variable names associated with differential equations. The value entered to the lower box corresponding to each state name is its initial condition at the beginning of simulation.

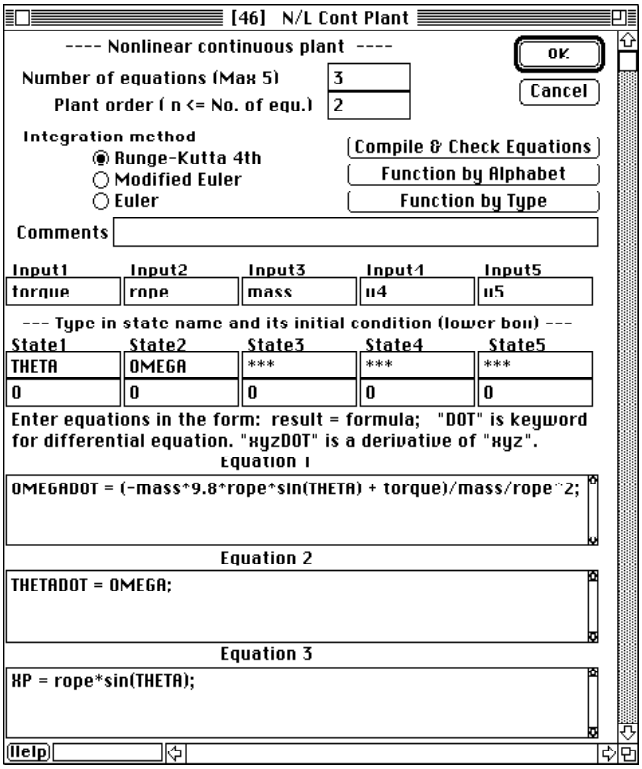
• *Equation 1 - 5:* The editable text entry box for the equations. The limit is 255 characters. The equations must be in the form: Result = Formula;. A semi-colon is required at the end of each equation.

Connectors

• *i1 - i5:* Input connectors corresponding to the dialog items *Input 1* through *5*. It is convenient for the adaptive control model to assign some of the inputs to the varying plant parameters.

• *o1 - o5*: Output connectors get the results of each equation, i.e. the left side of each equation, from the top (equation 1) to bottom (equation 5).

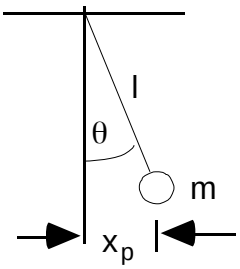
• *st:* The connector at the bottom is the plant's internal states. *State1* corresponds to element 1 of array *st* whose dimension equals the plant order.



Example

Demo 53

The previous dialog box is an example of how to write the dynamic equations of a pendulum. THETA and OMEGA correspond to the angle θ and angular velocity ω, respectively in the following equations.



$$\frac{\delta\omega}{\delta\tau} = (-\mu\gamma\lambda\sigma\nu(\theta) + \tau)/(\mu\lambda^2)$$

$$\frac{\delta\theta}{\delta\tau} = \omega$$

$$x_p = l\,sin(\theta)$$

Here m, l, x_p are mass, rope length and position of the weight.

☐ **Scalars to Array**		**[Mis]**
Organizes all input scalars into an array.		
	Inputs	*n1, n2, ..., n14* : scalars, up to 14 inputs
	Outputs	array, dimension is the number of connected inputs

Description

This block combines all scalars from connected input connectors into an array that has dimension equal to the number of connected connectors. The order of the scalars in the array is arranged sequentially by connector number. Connectors that are not used are discarded. For instance, if input connectors 2, 7, 10 of the block are connected, the output of the block would be a 3-element array with values of elements 1, 2, 3 equal to values from connectors 2, 7, 10, respectively.

Dialog Choices

None.

Connectors

All input connectors pass scalars.

The output connector passes an array.

Example

☐ **Shift Register**	**[Ctl]**

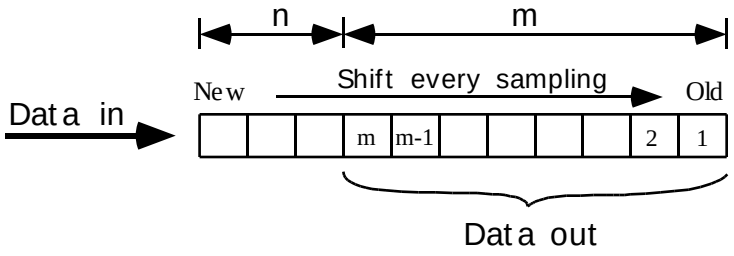
Demo 54

A signal is shifted and stored in a memory register at every simulation step.		
	Inputs	*in:* array or scalar, signal to be shifted and stored
	Outputs	*out:* array or scalar, manipulated signal

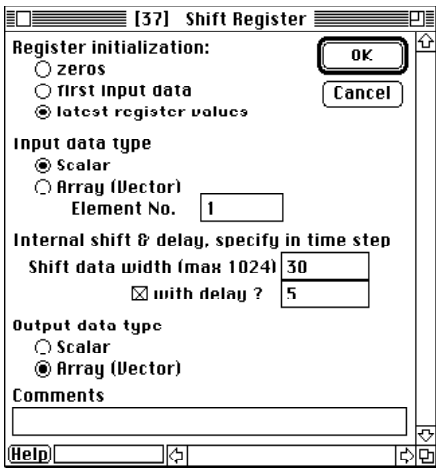
Demo 55

Description

The purpose of this block is to manipulate the signal and construct a time series data for control, identification and other applications. The input signal is stored into a memory shift register as shown in the figure below. The input signal is shifted at every simulation step so that the memory register contains the time series data. In digital control, one simulation cycle (step) corresponds to one sampling period.
The size of the shift register is determined by the "shift data width" *m* and "delay" *n* (i.e. *m*+*n* time series signals are stored in the shift register). The output is an array which contains *m* elements of the input time series signal. If the user sets 1 data width and *n* delay, this block simply becomes the block for *n* delay. The order of the array elements is the oldest signal first (element 1) and newest signal last (element m). At every simulation step, the above data manipulation is performed. Signal data type of input and output (scalar or array) can be specified in the dialog box.



Dialog choices



• *Register initialization:* All elements of shift register are initialized by zeros or the input data at first simulation step, or latest register value. The first data initialization prevents data discontinuity in shift register. The latest register value initialization is useful for multiple simulations.

• *Input data type:* If array is selected, the element number of the array is set for input signal (array elements start 1).

• *Shift data width:* Dimension of time series for output is set.

• *With delay:* If this option is checked, the output signal is delayed by the specified number of simulation steps.

• *Output data type:* Specify the output data type. If scalar type is selected, only the delay option has meaning (shift data width is fixed to 1).

Connectors

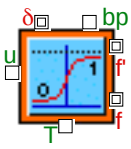
The input and output connectors can be assigned to either an array type or a scalar type depending on the dialog selections.

• *in:* Data input.

• *out:* Shifted and delayed output.

❒ Sigmoid [Edu]

Sigmoid transfer function for neuron.



Inputs	*u* : scalar, function input and message *T* : scalar, temperature of sigmoid function *bp* : scalar, output error and message
Outputs	*f* : scalar, sigmoid function value and message *f'* : scalar, derivative of sigmoid function δ : scalar, back-propagation delta and message

Description

The sigmoid logistic nonlinearity,

$$f(u) = \frac{1}{1 + exp[-(u-\theta)/\ T]}$$

is implemented in this block, with θ and *T* being known as the threshold value and temperature of the function, respectively. The temperature is adjustable through the block dialog and through the connector *T*. The threshold θ is fixed at zero since the **Adaptive Weighted Sum** block has a bias term.

This block performs two types of operations, forward and backward, activated by received messages instead of the simulation order number of the block.

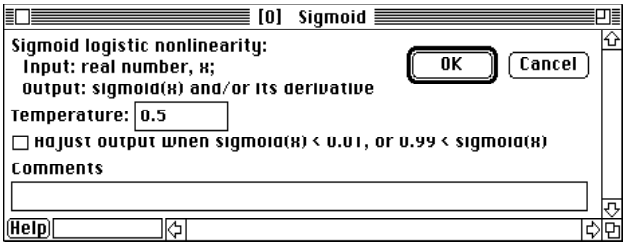
When the block receives a message from the *u* connector, the forward calculation will be executed. In forward calculation, the block outputs values of function f(*u*) and its derivative f'(*u*). On the other hand, backward calculation is executed when the block receives a message from the *bp* connector. After backward calculation, the δ connector outputs the back-propagation delta and sends a message to the **Adaptive Weighted Sum** block for back-propagation learning.

When the *u* connector is NOT connected to the **Adaptive Weighted Sum** block, the block follows the simulation order assigned by Extend and executes only forward calculation.

Dialog Choices

• *Temperature:* Parameter used in evaluating the Sigmoid function. This value is applied when the *T* connector is not connected.

• *Adjust output when sigmoid ...:* If the check box is checked, the output value of the Sigmoid function is clipped and has a maximum value of 0.99 and minimum value of 0.01.



Connectors

• *u:* The input of the Sigmoid function. The received message activates forward calculation if the block is connected with educational blocks in NeuroLab.

• *T:* Temperature parameter of the Sigmoid function. It will override the dialog value when connected.

• *f:* The output value of the Sigmoid function. It sends a message to activate forward calculation of connected blocks.

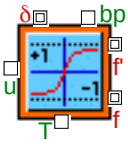
• *f':* The derivative of the Sigmoid function.

• *bp:* The output error. The received message activates backward calculation.

• δ*:* The back-propagation delta. It sends a message to connected **Adaptive Weighted Sum** blocks for back-propagation learning.

Example

A neuron with the sigmoid transfer function can be trained by back-propagation methods.

❒ **Tanh**		**[Edu]**
Hyperbolic tangent activation function		
	Inputs Outputs	*u* : scalar, function input and message *T* : scalar, temperature (optional) *bp* : scalar, output error and message *f* : scalar, function output and message *f'* : scalar, derivative of tanh(u/T) δ : scalar, back-propagation delta and message

Description

The hyperbolic tangent activation function

$$y = \tanh(u/T)$$

is implemented in this block. The parameter (temperature) T in the above equation controls the function slope or shape. The function increases the nonlinearity at lower temperatures. With the hyperbolic tangent function, the neuron output is mapped from the interval (–∞, +∞) to the interval (–1, +1). The derivative of the tanh function takes the form

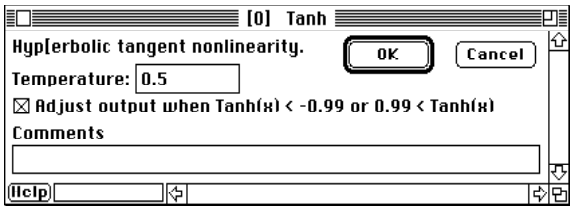
$$\F(dy,dx) = \F(1,T)\ \mathrm{sech}^2(u/T)$$

Note that the function is fully differentiable which allows the neuron to be trained by back-propagation methods.

The block performs both forward and backward operations triggered by messages instead of the simulation order number of the block. When the block receives a message from the *u* connector, the forward calculation will be executed. In the forward calculation, the function output is calculated for the given input *u*. On the other hand, a backward calculation is executed when the block receives a message from *bp* connector.

When the *u* connector is NOT connected to the **Adaptive Weighted Sum** block, the block follows the simulation order assigned by Extend and executes only forward calculation.

Dialog choices



• *Temperature*: The temperature of the tanh function when the *T* connector is not connected.

Connectors

All connectors are value connectors that pass scalars.

• *u*: The function input. This should be connected to the *ws* connector of an **Adaptive Weighted Sum** block. The received message activates the forward calculation.

• *f*: The function output. It sends a message to connected blocks.

• *f'*: The derivative of the tanh function.

• *bp*: The output error. The received message activates the backward calculation.

• δ: Back-propagation delta. It sends a message to connected blocks.

• *T*: Temperature of the tanh function. Will override the value given in block dialog , if connected.

❒ XY Array Plotter [Mis]

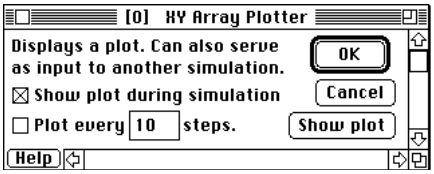
X-Y plotter for array variables (two sets of paired array).

Inputs	X : arrays, x-channel data Y : arrays, y-channel data
Outputs	arrays corresponding to the inputs

Description

This is a modified version of the **Worm Plotter** block in the plotter library. This block inputs one or two paired arrays instead of scalars. The block shows two sets of data plotted as x, y value pairs, but the number of points plotted is equal only to the array length. You must connect Y inputs in order to plot data. If X input is not supplied, the array element numbers are used for x values.

Dialog Choices



• *Show plot during simulation:* Opens plotter window and displays data during simulation if checked. Otherwise, data is only plotted at the end of simulation.

• *Plot every ? steps:* The data will be stored and displayed in the plotter window at every specified simulation steps. This option is useful for long simulation to save memory usage and to speed up calculation.

• *Show plot:* Re-draws the graph.

• *Input data type:* Selection of data *real* or *integer*.

Connectors

All connectors pass arrays. The top two inputs are for one pair of x, y values and the bottom is for a second pair.

• *X*: X-channel data. If not connected, x values are filled with the array element numbers by the block

• *Y*: Y-channel data.

Demo 60

Product Information

Requirement:

• NeuroLab™ v1.2 requires Extend™ version 3.1. Extend can be purchased in a bundled discount package with NeuroLab

( )*Macintosh or Power Macintosh*

• Macintosh II , SE/30 or later (faster machines with FPU are recommended)
• System software 6.07 or up (32 Bit QuickDraw is required)
• 4 MB of RAM (some image processing models require more RAM)
• 8 MB free space to install NeuroLab v1.2 and 8 MB for Extend on the hard disk.

(ÿ)*Windows or Windows NT*

• 80386 or greater (80486 or Pentium are recommended)
• DOS 5.0 or later, Windows 3.1 or later, and Win32s or later, **OR** Windows 95 or Windows NT 3.5 or later.
• 4 MB of RAM (some image processing models require more RAM)
• 8 MB free space to install NeuroLab v1.2 and 10 MB for Extend on the hard disk.
• A math co-processor is highly recommended.
• VGA or better graphics capabilities.

Price:

Product	Price
NeuroLab v1.2	$ 495
NeuroLab v1.2+ Extend v3.1 bundle	$ 999
Sales Tax (California resident add 8.25%)	
Shipping and Handling ($10.00, $14.00 for bundle)	
TOTAL	

Educational discount information is available upon request.
Volume discount information is also available upon request.

For additional information or to order a copy of NeuroLab, contact

NeuroLab Dept.
MIKUNI BERKELEY R&D CORP.
4000 Lakeside Dr.
Richmond, CA 94806
TEL: (510) 222 - 9880
FAX:(510) 222 - 9884
Email: neurolab-info.mikuni.com
WWW: http://www.mikuni.com/neurolab

Demo 61

NeuroLab™
Neural Network Library for Extend™

MIKUNI BERKELEY R&D CORP.
4000 LAKESIDE DR. RICHMOND, CA 94806
PHONE: (510) 222-9880 FAX: (510) 222-9884
e-mail: neurolab-info@mikuni.com

Order Form (Mail or fax this form)

Date:

Bill to:		***Ship to:*** (if different from billing address)	
Name	_____	Name	_____
Company	_____	Company	_____
Mailstop	_____	Mailstop	_____
Address	_____	Address	_____
City,State Zip	_____	City,State Zip	_____
Country	_____	Country	_____
Phone, Fax	_____	Phone, Fax	_____

Platform ❒ Mac ❒ Windows

Qty	Description	@Price	Amount
___	NeuroLab v1.2*	$495.00	_____
___	NeuroLab v1.2+ Extend v3.1	$999.00	_____
	Special discount		_____
	Add 8.25% Sales Tax for CA residents		_____
	Add $10 S/H or $14 for Bundle**		_____
	Total		_____

* NeuroLab v1.2 requires Extend v3.1, ** We ship by UPS Second Day (by Federal Express for international orders) unless otherwise specified. Approximate charges are:

Domestic (Options)	NeuroLab	Bundle	**International**	NeuroLab	Bundle
Fed-Ex Economy 2 days	$15	$20	Fed-Ex(Mexico&Canada)	$40	$60
Fed-Ex Priority Overnight	$25	$30	Fed-Ex (Other)	$45	$70

Payment: Visa, MasterCard, check or money order is acceptable.

❒ Credit Card (Visa, MasterCard)
Card #: _____
Expiration Date: _____
Card Holder's Name:_____

Signature: _____

❒ Check or money order attached payable to Mikuni Berkeley R&D Corp.
Checks and money orders must be drawn on a US bank in US dollars.

Prices are subject to change without notice. These prices are good only in the USA, Canada and Mexico.
Please contact Mikuni Berkeley R&D Corp. for volume discount/academic pricing/site license information.