Version 9.0

VisSim/Neural-Net User's Guide

By Altair Engineering, Inc.

Altair Engineering, Inc.
Altair Engineering User's Guides for VisSim Products - Version 9.0

Contents

Introduction

This section contains…

What is VisSim/Neural-Net

"What are the uses of neural network?" The answer can be as unique as the person who asks the question. Neural networks became known in the academic, scientific, and business communities for solving such problems as credit card application approval, target recognition, and speech recognition. Then in the late 1980s, the commercial community began using neural networks for process control applications in oil refineries, to design chemicals, track student performance, make stock market and other numeric predictions, optimize biological experiments, predict employee retention, analyze manpower, optimize athletic training, forecast sales, identify chemical compounds, flag product defects on assembly lines, and analyze waveforms. Customer ideas swelled the list to hundreds of different applications.

When to use a neural network

Neural networks excel at problem diagnosis, decision making, prediction, and other classifying problems where pattern recognition is important and precise computational answers are not readily available. The best examples are problems where input patterns must be classified into two or more categories or problems where numeric forecasts are required. For supervised learning, there must be sample patterns or historical cases available from which the networks may learn, but the system excels over traditional computational or rule-based methods when both the samples and the patterns to be categorized later are not always precisely defined. This "generalized" learning and recognition is the type of pattern recognition where neural networks are at their best.

In VisSim/Neural-Net, a neural network consists of a system of neurons connected by weighted links. In the most fundamental way, neural network mimics your brain's own problem-solving process. Just as you apply knowledge gained from past experiences to new problems or situations, VisSim/Neural-Net takes previously solved examples to train the weight values in a network. This is known as training a neural network. For a network to be trained, sample patterns or historical data must be available to teach the network. Once the training process is completed, the neural network is able to make decisions, classifications, or forecasts when presented with new data.

VisSim/Neural-Net excels at problem diagnosis, decision making, prediction, system identification and other problems where pattern recognition is important and precise computational answers are not readily available.

VisSim/Neural-Net can solve two kinds of problems: identification of input patterns into discrete categories, and smooth numeric predictions based on input data streams

The VisSim product family

The VisSim product family includes several base products and product suites, as well as a comprehensive set of targeted add-on modules that address specific problems in areas such as data communications, data acquisition, linearization and analysis, and digital signal processing.

Base products and product suites

Product	Function
Professional VisSim	Model-based design, simulation, testing, and validation of dynamic systems. A personal version, VisSim PE, is also available. VisSim PE limits diagram size to 100 blocks.
VisSim/Comm Suite	Simulates end-to-end communication systems at the signal level using 200+ communications, signal processing, and RF blocks. Includes Professional VisSim and VisSim/Comm blockset. A personal version, VisSim/Comm Suite PE, is also available. VisSim/Comm PE limits diagram size to 100 blocks and limits the Communication blockset. See the VisSim/Comm datasheet for details. VisSim/Comm Suite add-on modules are available for real-time data acquisition (Red Rapids digital tuner card); modeling PCCC turbo codes, including UMTS specification; and for support of Bluetooth, 802.11 a/b/g (Wi-Fi), and ultrawideband wireless designs.
VisSim/Embedded Controls Developer Suite	Rapidly prototypes and creates embedded controls for DSPs, DSCs, and MSP430 microcontrollers. You can simulate and generate scaled, fixed-point ANSI C code, as well as code for on-chip peripherals. Includes Professional VisSim, VisSim/C-Code, VisSim/Fixed-Point, and one user-specified target support. A personal version, VisSim/Embedded Controls Developer PE, is also available. VisSim/Embedded Controls Developer PE limits diagram size to 100.
VisSim Viewer (free)	Lets you share VisSim models with colleagues and clients not licensed to use VisSim.

Add-on modules

Add-On Module	Function
VisSim/Analyze	Performs frequency domain analysis of a linearized nonlinear subsystem.
VisSim/CAN	Interfaces with a USB CAN device to read and write CAN messages on the CAN bus.
VisSim/C-Code	Generates highly-optimized, ANSI C code that can be compiled and run on any platform that supports an ANSI C compiler.
VisSim/C-Code Support Library Source	Provides source code for the Support Library.
VisSim/Comm blockset	Simulates end-to-end communication systems at the signal level using 200+ communications, signal processing, and RF blocks. A personal version, VisSim/Comm PE, is also available. VisSim/Comm PE is a subset of the Communication blockset. See the VisSim/Comm datasheet for details. You can purchase VisSim/Comm add-on modules for real-time data acquisition (Red Rapids digital tuner cards); for modeling PCCC turbo codes, including UMTS specification; for support of Bluetooth, 802.11 a/b/g (Wi-Fi), and ultrawideband wireless designs.
VisSim/Digital Power Designer	Provides high-level blocks for simulation and code generation of power supply and digital power components and controls.
VisSim/Fixed-Point	Simulates the behavior of fixed-point algorithms prior to code generation and implementation of the algorithm on the fixed-point target.
VisSim/Knobs and Gauges	Provides dynamic gauges, meters, and knobs for process control, and measurement and validation systems.
VisSim/Model-Wizard	Generates transfer function model from historic or real-time data.
VisSim/Motion	Simulates motor control systems with customizable amplifiers, controllers, filters, motors, sensors, sources, tools, and transforms.
VisSim/Neural-Networks	Performs nonlinear system identification, problem diagnosis, decision-making prediction, and other problems where pattern recognition is important.
VisSim/OPC	Connects to any OPC server and log data or run a virtual plant in VisSim for offline tuning.
VisSim/OptimizePRO	Performs generalized reduced gradient method of parameter optimization.
VisSim/Real-TimePRO	Performs real-time data acquisition and signal generation using I/O cards, PLCs, and DCSs.
VisSim/Serial	Performs serial I/O with other computers.
VisSim/State Charts	Creates, edits, and executes event-based systems.
VisSim/UDP	Performs data exchange over the internet using UDP.
VisSim Viewer (free)	Lets you share VisSim models with colleagues and clients not licensed to use VisSim.

Resources for learning VisSim/Neural-Net

For those of you that are new to VisSim, we have provided several free services to make your transition to VisSim fast, smooth, and easy:

Interactive webinars

Interactive webinars offer you the opportunity to meet with Altair product specialists who will introduce and demonstrate our software products live on your computer and answer any questions you have. Each webinar is approximately 45 minutes long. To learn more about our interactive webinars, go to http://www.vissim.com/webinars/webinars.html.

Sample diagrams

VisSim 9.0 includes a directory of fully documented sample diagrams. These diagrams illustrate both simple and complex models spanning a broad range of engineering disciplines, including aerospace, biophysics, chemical engineering, control design, dynamic systems, electromechanical systems, environmental systems, HVAC, motion control, process control, and signal processing.

To access sample diagrams

Click on the **Diagrams** menu in VisSim.

Click on **Examples** > **Applications**.

Training

Altair offers training sessions for learning and gaining expertise in VisSim and the VisSim family of add-on products. Training sessions are conducted at Altairtraining facility in Westford, MA, as well as at customer sites and as online webinars.

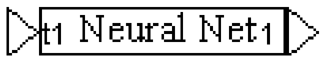
For information on setting up a training session, contacts sales@vissol.com.

Setting Up a Neural Network

This section contains…

Setting up your neural network

When you insert a neuralNet block into a diagram, it appears as follows:

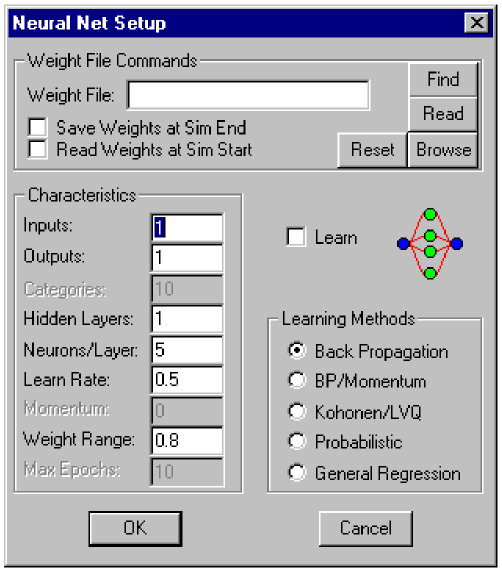


The input marked with a *t* represents training set input. Typically, when you train a neural network, you present it with a data set containing input facts (or parameters or variables) with the corresponding answers or results. This is usually called the "training set." For more information on training neural networks, see "Training a neural network."

To describe a network

1. Do one of the following:
    - Choose **Edit > Block Properties** and click over the **neuralNet** block.
    - Right-click over the **neuralNet** block.

    The Neural Net Setup dialog box appears.


Neural Net Setup

Weight File Commands

Weight File:

☐ Save Weights at Sim End

☐ Read Weights at Sim Start

Characteristics

Inputs:	1
Outputs:	1
Categories:	10
Hidden Layers:	1
Neurons/Layer:	5
Learn Rate:	0.5
Momentum:	0
Weight Range:	0.8
Max Epochs:	10

☐ Learn

Learning Methods

- [✓ Back Propagation]
- BP/Momentum
- Kohonen/LVQ
- Probabilistic
- General Regression

2. Make the appropriate selections. (See the descriptions below for more information about each parameter.)

3. Click **OK**, or press **ENTER**.

Choosing a learning method

There are two kinds of learning for the neuralNet block: supervised and unsupervised. There are also two kinds of learned outputs: discrete and continuous. The table below shows the relationship between the learning method and the type of output.

Output Type	Learning Method	
	Supervised	**Unsupervised**
Continuous	Back Propagation BP/Momentum General Regression	
Discrete	Probabilistic	Kohonen/LVQ

Continuous output networks can have continuous real valued outputs. Discrete output networks can only produce an integer that describes the category into which the input pattern falls.

In supervised learning, you train a network by presenting a sequence of known input and output patterns. For discrete valued outputs, you train by supplying the category number in the output pattern. Note that you may have more than one input pattern in the same output category.

The supervised continuous learning methods available in VisSim/Neural-Net are Back Propagation, BP/Momentum, and General Regression. The method used for supervised discrete learning is Probabilistic.

Unsupervised learning is a discrete output method that learns its own classification category from the training data. Generally, you need only supply one training data point per output category; however, the network may decide that two data points are too similar to warrant their own category. You can see that it makes sense to comb your data to look for representative examples. The method used for unsupervised discrete learning is called Kohonen Learning Vector Quantizer (Kohonen/LVQ).

No learning method is guaranteed to give an absolutely correct answer, especially if patterns are incomplete or conflicting. Results should be evaluated in terms of the percentage of correct answers. In this regard, the technology is similar to biological neural functioning after which it was designed, and differs significantly from other conventional computer software.

Choosing network characteristics

Choosing the appropriate characteristics for a network is part of the art of developing a neural network. This section briefly examines the various characteristics you can apply to your network.

Choosing inputs

There are two types of input to the neuralNet block:

- System parameter input
- Training set input

Unlike with other blocks, you don't add and delete inputs with the Edit menu's Add and Remove Connector commands, but rather through the Inputs and Outputs parameters in the Neural Net Setup dialog box.

System parameter input: The Inputs parameter controls the number of system parameter input connector tabs for the neuralNet block.

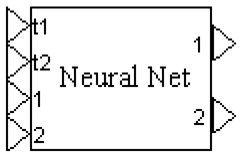
Training set input: Training set input is connected to the input connector tabs labeled with a *t*. You control the number of training set input connector tabs with the Output parameter. For more information, see "Choosing outputs."

When you're training a neuralNet block, the test input data tells the neuralNet block the correct output for each input.

Choosing outputs

The Outputs parameter controls the number of output connector tabs and the number of *t* input connector tabs that VisSim adds to the neuralNet block when the Back Propagation, BP/Momentum, or General Regression learning method is activated. If either the Kohonen/LVQ or Probabilistic learning method is activated, this parameter is dimmed and unavailable for use.

The input connector tabs labeled with a *t* are reserved for training input data. If, for example, you specify two outputs and two inputs for a neuralNet block, VisSim adds four input connector tabs and two output connector tabs to the block, and labels the top two input connector tabs with *t*s.



When you're training a neuralNet block, the test input data tells the neuralNet block the correct output for each input.

Choosing the categories

The Categories parameter lets you choose the number of discrete states that the network can discriminate for the Kohonen/LVQ and Probabilistic learning methods.

Choosing the number of hidden layers

The Hidden Layers parameter indicates the number of hidden layers in the neural network. There can be up to 32 hidden layers in a network.

There are no hidden layers in a Kohonen/LVQ neural network. When you activate Kohonen/LVQ learning, the Hidden Layers parameter is automatically reset to 0 and dimmed. Similarly, there is only one hidden layer in a Probabilistic neural network. When you activate Probabilistic learning, the Hidden Layers parameter is automatically reset to 1 and dimmed.

Choosing the number of neurons per layer

The Neurons/Layer parameter indicates the number of neurons per hidden layer. There can be up to 32,767 neurons per hidden layer.

Choosing the learn rate

The Learn Rate parameter lets you regulate how weights are changed when a network is trained. The amount of weight modification is proportional to the error count. A learning rate is the constant of proportionally. For example, when the Learn Rate parameter is set to 0.5, the weight change is a function of only half of the error.

As you increase the value of the Learn Rate parameter, not only do the weight modifications increase but also the speed at which the network learns. Valid values for this parameter range between 1e-5 and 1. If the Learn Rate parameter exceeds 1, the network may become unstable.

Choosing the momentum

The Momentum parameter keeps a Back Propagation network from oscillating when higher learning rates are used. You can only use the Momentum parameter when the BP/Momentum learning method is activated. Valid values for this parameter range between 0 and 1.

Choosing the neighborhoods

The Neighborhoods parameter controls the number of adjacent nodes initially affected by the Kohonen/LVQ learning method. Valid values for this parameter range from 0 to the number of neurons in the hidden layers.

Choosing the smoothing factor

The Smoothing parameter controls the probability density estimates being output for a Probabilistic network. Valid values for this parameter range from 0.01 to 1.

Choosing the weight ranges

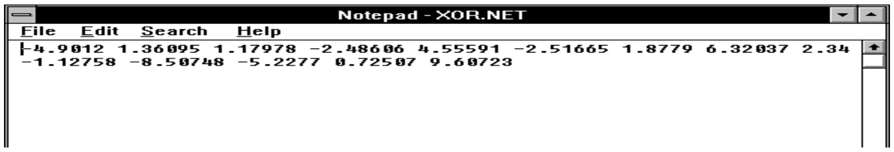
When the network is first created, the values of the weights will be initialized to ± the value specified for the Weight Range parameter. Valid values for this parameter are greater than 0 and less than 1. The recommended values are between 0.3 and 0.6.

Choosing the maximum epochs

The Max Epochs parameter lets you set the maximum number of iterations through a data set for a Kohonen/LVQ network. Any value greater than 0 is a valid value for this parameter. Typical values range between 200 and 1000.

Using weight files

A weight file is an ASCII text file containing the weights of the connections between neurons. An example of a weight file that has been opened in the Microsoft Notepad utility is shown below.



```
-4.9012 1.36095 1.17978 -2.48606 4.55591 -2.51665 1.8779 6.32037 2.34
-1.12758 -8.50748 -5.2277 0.72507 9.60723
```

You save weights at the end of a training session, and you can use saved weights at the beginning of a training session.

Saving learned weights

You can save learned weights at the end of a training session by activating the Save Weights at Sim End parameter.

To save learned weights

1. Do one of the following:
   - Choose **Edit > Block Properties** and click over the neuralNet block.
   - Right-click over the neuralNet block.

   A Neural Net Setup dialog box appears.
2. Under Weight File Commands, do the following:
   - Activate **Save Weights at Sim End**.
   - In the Weight File box, enter the name of the weight file into which the learned weights are to be saved.
3. Click **OK**, or press **ENTER**.

Reading in saved weights

You can read in saved weights at the beginning of a training session by activating the Read Weights at Sim Start parameter.

To read saved weights

1. Do one of the following:
   - Choose **Edit > Block Properties** and click over the neuralNet block.
   - Right-click over the neuralNet block.

   A Neural Net Setup dialog box appears.
2. Under Weight File Commands, do the following:
   - Activate **Read Weights at Sim Start**.
   - In the Weight File box, enter the name of the weight file from which the learned weights are to be read.

3. Click **OK**, or press ENTER.

Finding weight files

Use the Find command button to select the weight file into which weights are to be saved or from which weights are to be read.

To select a weight file

1. Do one of the following:
   - Choose **Edit > Block Properties** and click over the neuralNet block.
   - Right-click over the neuralNet block.

   A Neural Net Setup dialog box appears.
2. Under Weight File Commands, enter the name of the weight file in the Weight File box.
3. Click **OK**, or press enter.

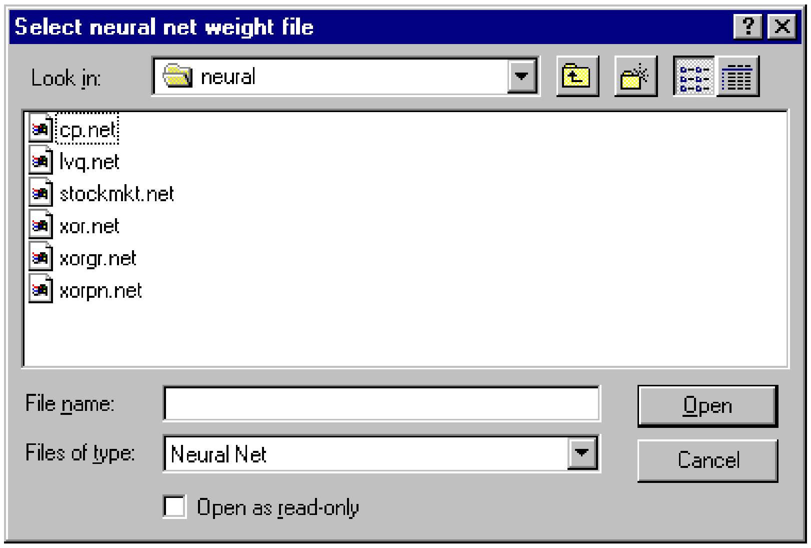
To select a weight file using the Find command button

1. Do one of the following:

- Choose **Edit > Block Properties** and click over the neuralNet block.
   - Right-click over the neuralNet block.

   The Neural Net Setup dialog box appears.
2. Click **Find**.

   The Select Neural Net Weight File dialog box appears.



3. Move to the appropriate directory and select a weight file.
4. Click **Open**, or press ENTER.
5. VisSim displays the previous dialog box. The selected weight file will appear in the Weight File box.

Opening weight files

You use the Microsoft Notepad utility to open weight files for examination or editing

To open weight files

1. Do one of the following:
   - Choose **Edit > Block Properties** and click over the neuralNet block.
   - Click the right mouse button over the neuralNet block.

   The Neural Net Setup dialog box appears.
2. Enter the name of the weight file to be browsed or find it using the **Find** command.
3. Click **Browse**.
4. VisSim starts the Microsoft Notepad utility with the specified file.

Resetting weights

The Reset command button resets the weights to random initial states. The Reset command button is typically used when your network has learned incorrect data.

To reset weights

1. Do one of the following:
   - Choose **Edit > Block Properties** and click over the neuralNet block.
   - Right-click over the neuralNet block.

   The Neural Net Setup dialog box appears.
2. Click **Reset**.

Training a neural network

Once you have created a network the next step is to train the network from the data in your training set. How you train a neural network depends on the learning method you use. The following table describes the type of learning you should use based on the size of the training data.

Training data size	Multiple pass training	Single pass training
Unlimited	Back Propagation BP/Momentum	
Limited to neurons in hidden layer	Kohonen/LVQ	General Regression Probabilistic

For either of the Back Propagation methods, you can train over any number of data points, and you keep training multiple times through your training set until the average error between the network output and the training output goes below a certain threshold.

For the remaining methods, you can only have as many training data points as you have neurons in the hidden layer. The training set contains sample input data with known outputs. You should also have a separate test data set to test the network after training is complete.

For unsupervised learning, you supply representative input patterns and the network assumes each pattern represents a unique category.

As you might expect, Probabilistic and General Regression, which are one-pass learning methods, are fast to train. However, they can take longer to execute since they may require more neurons to adequately learn the data set, and thus require more processing at classification time.

Choosing training sets

An adequate number of training sets must be available to allow the network to cover the problem domain. If you do not have enough actual data to create an adequate training set, you may want to contrive combinations of inputs and outputs to produce training sets. In this way, you can assure a wide variety of training sets and make sure that there are enough training sets for each of the output categories.

It is normal to find problems where several different patterns of inputs will result in the same classifying outputs. If, however, there are samples with identical inputs that result in different outputs, the training may not go below a specified learning threshold.

Initiating training with the Learn parameter

Learn parameter in the Neural Net Setup dialog box. You may also want to specify the use of a weight file to be read in at the start of the training session or to be written to at the end of the training session.

Before you can initiate the training process, you should connect a plot block to the output side of the neuralNet block in which to observe the learning process. During training, network output represents the mean square error for the Back Propagation learning methods.

To train a network

1. Do one of the following:
   - Choose **Edit > Block Properties** and click over the neuralNet block.
   - Right-click over the neuralNet block.

   The Neural Net Setup dialog box appears.
2. Make the following selections:
   - Activate **Save Weights at Sim End** to save the learned weights to a weight file after training is complete.
   - Optionally, activate **Read Weights at Sim Start** to read in learned weights that have been previously saved.
   - In the Learning Methods box, choose a learning method.
   - In the Characteristics box, choose the network characteristics.
   - Activate **Learn**.
3. Choose **System > Go**, or press ▶ in the toolbar.
4. In the plot block, observe the network estimation error. When the error is at an acceptable level, you can terminate the simulation with **System > Stop**.

When to stop training

A challenging aspect of building a successful neural network is knowing when to stop training. If you train too little, the network will not learn the patterns. If you train too much, the network will learn the noise or memorize the patterns and not generalize well. You can tell how close your network is by examining the error factor (which is a value that represents the sum of the squares of the differences between the actual output values with which the network is trained) and the network's predicted outputs.

Verifying results

The best way to measure the adequacy of the model after you have built it is to compare its answers with those in your test set. Only simple networks will be 100% accurate.

If the number of correct classifications is not high enough, there are a number of things you can try to correct the problem:

- **Change the inputs and outputs to better represent the problem.** Use inputs that are more predictive. This is usually the most effective change you can make.
- **Change the training sets.** Make sure training sets are representative of all classifications, and add more if necessary. Try removing anomalies, if there are any. Make sure all training sets have correct data and classifications.
- **Increase the number of hidden neurons.**
- **Increase the training.** Sometimes a problem will eventually fall below a specified learning threshold if the error factors are still getting smaller.
- **Broaden the classifications.** When more than a few answers are close to more than one category, try expanding the classifications.
- **Decrease the learning rate and momentum.** A learning rate of 0.1 with a momentum of 0 often works well.
- **Randomly vary the order of the training sets.** This prevents like sets from clustering together, and removes time series dependencies.

Applying VisSim/Neural-Net to financial applications

If you're working on financial applications or other time-series problems, you may find it helpful to create additional input variables derived from the raw input data they already have. These additional inputs are usually of the following types:

- **Lagged versions of raw data.** For example, if your raw data is today's price and you're trying to predict tomorrow's price, then it may be helpful to include yesterday's price, and the price the day before, as well.
- **Averages of previous raw data.** For example, rolling averages.
- **Changes (deltas) in the raw data.** For example, you might want the difference in today's price and yesterday's.

Financial data may have to be normalized if it is so old that the price patterns are much higher now. There are many ways to do this, like measuring prices from a moving baseline and introducing percentage differences. You can also omit very old data.

Remember that the most successful financial models have less to do with learning rate, momentum, hidden neurons, etc., and depend most of all on how predictive or leading your input indicators are.

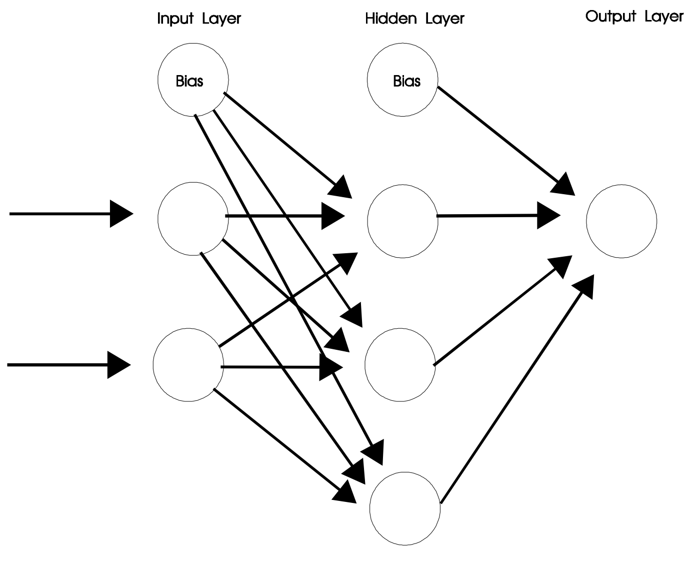
Back Propagation Learning Methods

This section contains…

How a Back Propagation network works

To train a neural network using a Back Propagation learning method, you must present the neuralNet block with a data set containing sample inputs and corresponding desired outputs. This data is typically referred to as your training set. You should also have a second set of data, in the same format, with which to test the network later on. This data set is called your test set.

As illustrated in the figure below, a Back Propagation neural network usually has three or more layers of neurons. Each layer is connected or linked to the neurons in the next layer. These links have numeric weights that are applied to the current value held by the neuron. Input values in the first layer are weighted and passed to the next layer, called the hidden layer. Neurons in the hidden layers produce outputs that are based on the sum of the weighted values passed to them. The hidden layers pass values to the output layer in the same fashion. The output layer then produces the desired results, such as a prediction or classification



The neural network learns by adjusting the interconnection weights between layers. The answers the network produces are repeatedly compared with the correct answers supplied in the training set. After each comparison, the connecting weights are adjusted slightly in the direction of the correct answers.

Eventually, if the problem can be learned, a stable set of weights adaptively evolves, which will produce reasonable answers for all of the sample decisions or predictions. The real power of the neural network is evident when it can produce good results for data it has never seen before.

A Back Propagation with momentum network

A Back Propagation with momentum, or BP/Momentum, network works in much the same way as a Back Propagation network. The difference lies in the momentum variable which allows you to control the proportion of the last weight change to be added into the new weight change. Momentum keeps a network from oscillating when higher learning rates are used. The momentum variable usually ranges from 0 to 1; however, typical values are between 0.3 and 0.7.

Common Back Propagation networks

A three-layer neural network is suitable for the vast majority of working applications, including those involving pattern classification. Each neuron in the input layer contains data corresponding to an input variable of the problem. The neurons in the output layer contain the network's classifications or results.

If a three-layer network doesn't solve your problem, there's a good chance that you need to work on your input variables or your sample training set.

- In predictive networks, check that the input variables in your training set are predictive.
- If you're classifying patterns, make sure that your training set is complete and that the patterns are not conflicting. If the training set contains conflicting patterns, more variables may be required to make adequate determinations.

Learning nonlinear relationships with Back Propagation

Nonlinear problems often exhibit better learning dynamics with a Back Propagation learning method if a small learning rate (≤ 0.1) is used with a small (≤ 0.1) or no momentum term. (There are no firm rules of thumb on this, and experience and trial-and-error are still the best way to know for sure.) If you choose a Back Propagation learning method, the size of the weight matrices will be cut in half. If your network is large and you are running out of memory, then do the same thing.

If all your inputs and outputs are binary (0 or 1), you may want to choose the BP/Momentum learning method and select a learning rate of 0.6 and momentum of 0.9. If your data patterns contain considerable noise, try lowering the learning rate and momentum. For example, a lower learning rate of 0.05 or 0.1 with a momentum of 0.5 or 0.6.

The Back Propagation network structure

Normalizing inputs and outputs for Back Propagation

Inputs and outputs of Back Propagation networks need to be normalized.

Inputs to the neuralNet block need to be in the range of 0 to 1, and outputs work best when in the range of 0.1 to 0.9. For instance, if an input's actual value can range from 0 to 20, then that input needs to be divided by 20 before being fed to the neuralNet block.

Be sure to remember to re-normalize Back Propagation output results by multiplying the result by the inverse of the input scaling. For instance, if a training input was normalized with a gain of 0.1, then the corresponding output should be multiplied by 10.

Choosing the hidden layers for Back Propagation

In a Back Propagation network, there is usually one hidden layer; however, there can be more. The purpose of the hidden layers is to enable the network to learn more complex patterns. They serve as feature detectors.

Choosing the neurons per layer for Back Propagation

Part of the art of designing a neural network is building it with the appropriate number of hidden neurons for the application. In a Back Propagation network, if the middle layer is too large, the neuralNet block will memorize the training sets and not be able to generalize when presented with data with which it hasn't been trained. If too few hidden neurons are used, the neuralNet block may not be powerful enough to hold all of the unique situations found in the training sets. Training may never get to low enough error factors, or, if it does, the system may not be able to reproduce the output patterns.

Choosing the learn rate for Back Propagation

The learning rate lets you regulate how much the weights are changed when the network is trained. The amount of weight modification is proportional to the amount of error. The learning rate is the constant of proportionality. For example, if the learning rate is set to 0.5, the weight change is a function of only half of the error. The larger the learning rate, the larger the weight changes, and therefore the faster the learning.

Learning rates should not exceed 1 or the system could become unstable.

Choosing the weight ranges for Back Propagation

The weight range sets the limits for the range of random generated values that are used to initialize the network weights when the network is first created, or reset. The randomly generated weight values will be in the range of ± the weight range parameter.

For instance, if a weight range of 0.3 is selected, then the weight values will initially be randomly generated in the range of -0.3 to +0.3. The initial weight range is important for determining the learning speed of the neural network. Typical values vary between 0.2 and 0.7.

Solving the Exclusive OR problem with a Back Propagation network

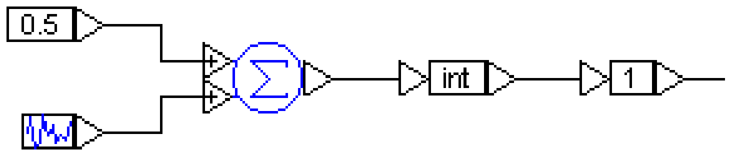
The following example illustrates how to build a three-layer Back Propagation neural network to solve the exclusive OR problem. In this problem, there are two inputs and one output. When either input is ON, the output is ON. When both inputs are OFF, the output is OFF. When both inputs are ON, the output is OFF. The data would look like this:

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

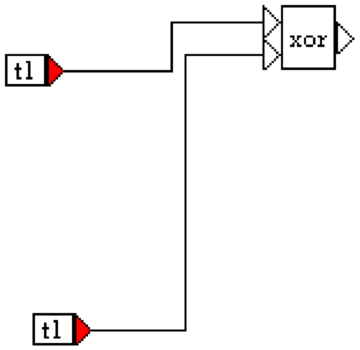
A three-layer neural network can solve this problem provided the hidden layer has at least three neurons. If the hidden layer has two neurons, the answer is correct only half the time.

To construct a block diagram that solves the exclusive OR problem

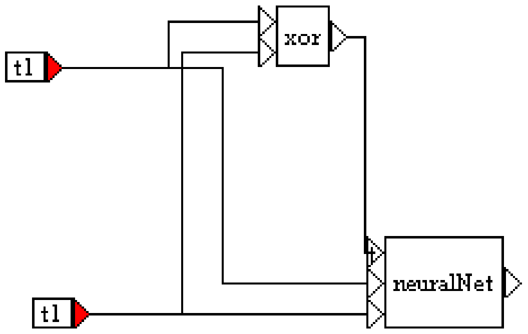
1. Construct a block algorithm that generates random sequences of 0s and 1s, as shown below:



2. Encapsulate the algorithm in a compound block called T1 and make a copy of it.
3. To generate a training set, feed the T1 compound blocks into the xor block. The output from the xor block is the training set, or known correct output.

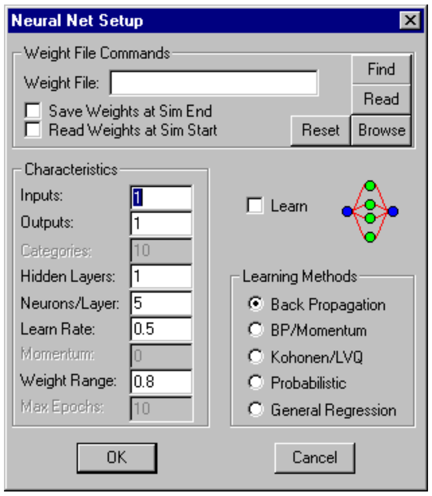


4. To create a three-layer Back Propagation neural network, insert a neuralNet block in the diagram.
5. Wire the output from the T1 compound blocks, along with the training set generated by the xor block into the neuralNet block. The training set is wired to the connector tab labeled with a *t* to indicate that it is training input data.



6. Do one of the following:
   - Choose **Edit > Block Properties** and click over the neuralNet block.
   - Right-click over the neuralNet block.

   The Neural Net Setup dialog box appears.


Neural Net Setup

Weight File Commands

Weight File:

☐ Save Weights at Sim End
☐ Read Weights at Sim Start

Characteristics

Inputs:	1
Outputs:	1
Categories:	10
Hidden Layers:	1
Neurons/Layer:	5
Learn Rate:	0.5
Momentum:	0
Weight Range:	0.8
Max Epochs:	10

☐ Learn

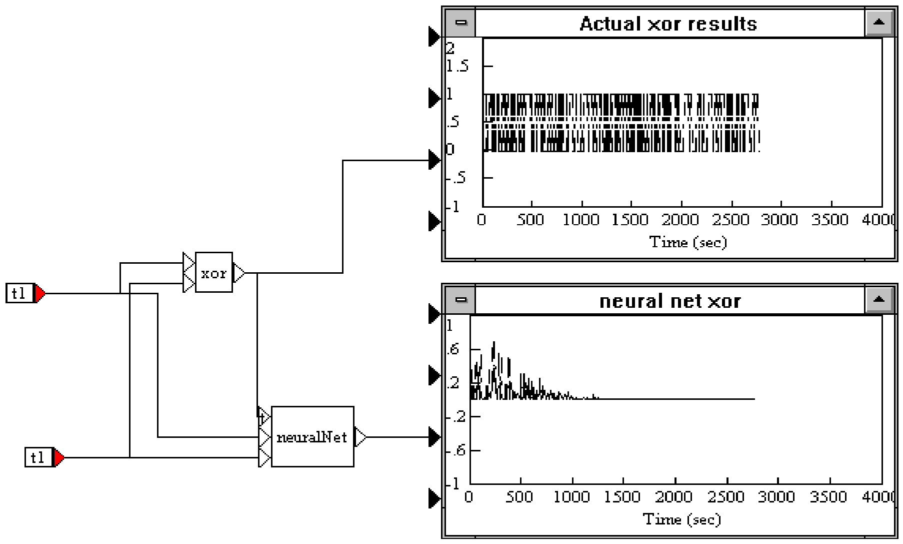
Learning Methods

- [✓ Back Propagation]
- BP/Momentum
- Kohonen/LVQ
- Probabilistic
- General Regression

7. Make the following changes in the dialog box:
   - In the Weight File box, enter **XOR**.**NET**.
   - Under Characteristics, do the following:
     - In the Inputs box, enter **2**.
     - In the Outputs box, enter **1**.
     - In the Hidden box, enter **1**.
     - In the Neurons/Layers box, enter **4**.
     - In the Learn Rage box, enter **0.7**.
     - In the Weight Range box, enter **0.4**.
   - Activate **Learn**.
   - Under Learning Methods, activate **Back Propagation**.
   - Click **OK**, or press **ENTER**.

The training process begins when you initiate a simulation with the System > Go command or the ▶ button in the toolbar. The neuralNet block learns by repeatedly comparing its output with the training patterns. Each time, the interconnection weights between layers in the network are adjusted slightly in the direction of the training patterns.

To observe the training process you can wire two plot blocks into the diagram. As shown in the figure below, the top plot block displays the correct, known exclusive OR results; the bottom plot block shows the neuralNet block learning.



When the output from the neuralNet block approaches 0, learning is complete and you can halt the training process by stopping the simulation.

Experimenting with advanced Back Propagation

Below are a few suggestions for ways you can define your models. Remember, you are not dealing with a science, but rather an art. Very often something that works with one type of problem will not work with another.

- **Adjust the number of hidden neurons.** Larger numbers store more patterns but tend to cause the model to memorize patterns instead of being able to generalize about them.
- **Adjust the learning rate and momentum.** Momentum enables learning rates to be higher without de-stabilizing the learning. You can also try varying the learning rate and momentum as learning proceeds by stopping the simulation, changing the values, and then continuing the simulation.

Probabilistic Learning Method

The Probabilistic supervised learning method is a type of pattern classifier that uses probability density function estimates to make classifications. A Probabilistic network is similar in effect to a k-nearest neighbor classifier, because it uses a degree of averaging of nearest neighbors dictated by the density of training patterns.

When using the Probabilistic learning method, training is very fast because there is no error feedback and subsequent adjustment of weights. Training a Probabilistic network in VisSim/Neural-Net really only involves loading the weight matrices. The work is done during propagation when the network inspects all training patterns to decide which category contains the patterns that the new pattern most closely matches.

Since training a Probabilistic network is almost instantaneous, training can be done in real time. Training patterns can be added or replaced at any time, and as soon as there are patterns for each possible category, classification can take place. Classification will improve as more training patterns are added.

If your training sets are very large, classification may take longer when using Probabilistic learning than when using Back Propagation or BP/Momentum. In such cases, using the Kohonen/LVQ and Probabilistic learning methods together will decrease the time to classify. Kohonen/LVQ can be used to find representative exemplars from the full database. These, in turn, can be used as the reduced training set for Probabilistic learning.

When to use Probabilistic learning

Typically, you should use the Probabilistic learning method in place of Back Propagation when:

- The input data is sparse.
- The training patterns need to be classified into *n* categories, as in unsupervised Kohonen/LVQ. Unlike Kohonen/LVQ, Probabilistic training involves showing the network the correct categories during training. If, however, your problem requires that outputs be continuous predictions rather than categories, you cannot use Probabilistic training. Use General Regression instead.

The Probabilistic network structure

Probabilistic networks are three-layer networks wherein the training patterns are presented to the input layer and the output layer has one neuron for each possible category. The network produces activations in the output layer corresponding to the probability density function estimate for that category. Thus, the highest output represents the most probable category. There must be as many neurons in the hidden layer as there are training patterns.

Choosing discrete state categories for Probabilistic

Probabilistic requires that you supply a finite number of discrete states among which it can discriminate.

Choosing the smoothing factor for Probabilistic

Probabilistic requires that you supply a smoothing factor like the one used with General Regression. Note that the smoothing factor only takes effect after the network has been trained. You may change it and experiment with it; however, the proper setting of the smoothing factor is critical for most of the problems you will encounter. Too large a smoothing factor will cause the network to ignore features of the output set, while too small a smoothing will cause exact recognition of the original training set, but fail to recognize input patterns that are close to a training set member.

If you map your inputs between 0 and 1, smoothing factors from 0.001 to approximately 0.5 are typical. If you are not scaling inputs, another range of smoothing factors will be more appropriate.

A systematic approach is recommended for determining the best smoothing factor for each Probabilistic network. You should create an independent test set with patterns not found in the training set. Process the entire test set with a number of smoothing factors. Each time the test set is processed, compute the mean squared error for the test set. Use the smoothing factor that produced the smallest mean squared error on the test set.

To compute the mean squared error for a test set with one output

1. Square the difference between the actual outputs and network-produced outputs for each pattern.
2. Add the squares for all the patterns.
3. Divide by the number of patterns.

To compute the mean squared error for a test set with more than one output

1. Scale the outputs into the same range.
2. Sum the squares of differences for all of the outputs for a pattern.
3. Sum them over all patterns.
4. Divide them by the number of patterns.

If you iterate through a number of smoothing factors, you will find that one produces a smaller mean squared error than the others. Use this smoothing factor for your new data.

Training a Probabilistic network

Training a Probabilistic network consists of only one pass of the patterns. Each pattern will be presented to the network only once during training.

Unlike Back Propagation, Probabilistic training does not require that patterns be propagated forward before the training. All that is required is that a pattern be represented to the input layer.

In a Probabilistic network, the outputs will be integers representing categories starting at 0.

Combining Probabilistic and Kohonen/LVQ learning

The Probabilistic learning method is exceptionally well-suited to handle multiple interacting networks; however, because Probabilistic learning can slow down significantly with higher numbers of categories, you may want to categorize at a high level first. In this situation, you can use Kohonen/LVQ networks to create a smaller set of training patterns for each category. This process involves building a Kohonen/LVQ network for each Probabilistic output category. For example, suppose you want a specific category to have only 50 training patterns, but the original set of training patterns contains 3,500 training patterns for that category. You can do the following:

1. Build a Kohonen network with the same number of inputs as the Probabilistic network.
2. Set the number of neurons in the Kohonen/LVQ output layer to 50.
3. Train the Kohonen network with the 3,500 original patterns. The 50 weight vectors that result in the link will be average, or "center of cluster," vectors for the original 3,500.
4. Train the Probabilistic network with the average vectors.
5. Repeat these steps for each of the Probabilistic categories.

General Regression Learning Method

VisSim/Neural-Net contains another very fast supervised learning method called General Regression, which responds well to localized regions of input space.

Both Probabilistic and General Regression were invented by Dr. Don Specht. The key difference between them is that Probabilistic produces only integer-valued outputs, whereas General Regression produces continuous-valued outputs.

General Regression excels over Back Propagation when handling continuous function approximation. For functions as simple as $y = x^2$ or $z = y^2 - x^2$, General Regression will approximate very accurately, while Back Propagation can make only rough approximations after many learning epochs. On the other hand, noisy data without any known underlying function – such as, financial or market predictions – may fair better with Back Propagation, which is less localized. Unlike Back Propagation, General Regression will not produce outputs outside the range of those with which it was trained.

The General Regression network structure

General Regression is a three-layered network where there must be one hidden neuron for each training pattern (unless you cluster them in some way). There are no training parameters, such as learning rate and momentum. The only adjustment which must be made is a smoothing factor like the one used with Probabilistic. The smoothing factor for General Regression is used only after the network is trained.

Choosing the smoothing factor for General Regression

General Regression requires that you supply a smoothing factor like the one used with Probabilistic. Note that the smoothing factor only takes effect after the network has been trained. You may change it and experiment with it; however, the proper setting of the smoothing factor is critical for most of the problems you will encounter. Too large a smoothing factor will cause the network to ignore features of the output set, while too small a smoothing will cause exact recognition of the original training set, but fail to recognize input patterns that are close to a training set member.

If you map your inputs between 0 and 1, smoothing factors from 0.001 to approximately 0.5 are typical. If you are not scaling inputs, another range of smoothing factors will be more appropriate.

A systematic approach is recommended for determining the best smoothing factor for each General Regression network. You should create an independent test set with patterns not found in the training set. Process the entire test set with a number of smoothing factors. Each time the test set is processed, compute the mean squared error for the test set. Use the smoothing factor that produced the smallest mean squared error on the test set.

To compute the mean squared error for a test set with one output

1. Square the difference between the actual outputs and network-produced outputs for each pattern.
2. Add the squares for all the patterns.
3. Divide by the number of patterns.

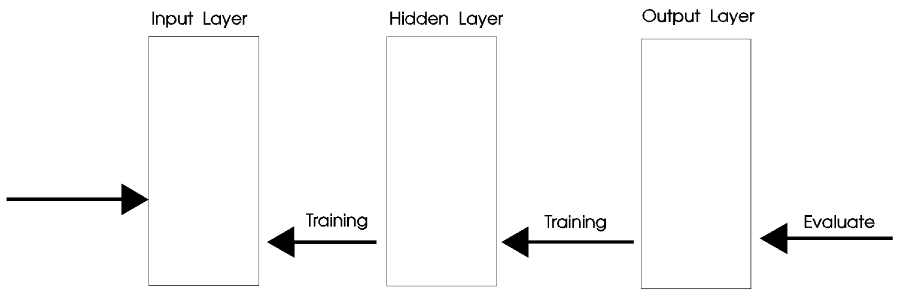
To compute the mean squared error for a test set with more than one output

1. Scale the outputs into the same range.
2. Sum the squares of differences for all of the outputs for a pattern.
3. Sum them over all patterns.
4. Divide them by the number of patterns.

If you iterate through a number of smoothing factors, you will find that one produces a smaller mean squared error than the others. Use this smoothing factor for your new data.

Training a General Regression network

Like a Probabilistic network, a General Regression network is essentially trained after only one pass of the training patterns. In addition, General Regression is capable of functioning after only a few training patterns have been entered, although it obviously improves with additional patterns.



.

Kohonen/LVQ Learning Method

VisSim/Neural-Net offers a powerful and popular unsupervised learning technique called Kohonen/LVQ, which stands for Learning Vector Quantizer. This algorithm learns to make pattern classifications by making its own clustering scheme for patterns. The patterns (or vectors) are clustered into categories based on their proximity to each other.

Uses of Kohonen/LVQ

Unsupervised Kohonen/LVQ networks have been used for:

- Image processing
- Control
- Speech processing
- Data compression
- Combinatorial optimization

Another emerging use of Kohonen/LVQ is as a pattern classifier front-end for supervised learning. A Kohonen/LVQ network looks at input patterns and classifies them. Based on this classification, the pattern is then sent to one of two or more specialized Back Propagation networks for training and further pattern recognition. Later, when the back propagation networks are trained and being utilized, the Kohonen/LVQ network is still acting as a front-end, routing each pattern to the correct Back Propagation network.

The Kohonen/LVQ network structure

In a Kohonen/LVQ network, there are only two layers:

- An input layer, where patterns of P variables are placed.
- An output layer, which has one neuron for each possible category. If you want your patterns to be categorized into at most N classifications (clusters), then the output layer will contain N neurons. Kohonen/LVQ may decide your patterns have less than N categories, but N will be the maximum it can find.

Kohonen/LVQ works much like supervised learning techniques in the sense that patterns are presented to the input layer, then propagated to the output layer and evaluated. Only one output neuron is the winner. That is, the weight vector (all the weights) leading to the winning neuron is closer in space to the input pattern than that of any other output neuron. The network weights are then adjusted during training by bringing this weight vector slightly closer to the input pattern. This process is repeated for all patterns for a number of epochs usually chosen in advance.

Choosing discrete state categories for Kohonen/LVQ

Kohonen/LVQ requires that you supply a finite number of discrete states among which it can discriminate.

Choosing the learn rate for Kohonen/LVQ

Kohonen/LVQ learning differs from Back Propagation learning with respect to its sensitivity to learning rate. In Kohonen/LVQ, the learning rate must be lowered slightly but steadily as the training progresses, causing smaller and smaller weight changes. This forces the weights to stabilize to a state where input vectors close to one another are consistently categorized together because the weight vectors to the output neurons adjust themselves to a kind of center of gravity for the cluster they define.

Choosing the neighborhoods for Kohonen/LVQ

To function properly, Kohonen/LVQ also depends on another technique, commonly referred to as “on center/off surround.” Without this technique, one neuron could end up winning all the time. During training, the weights leading to the winning neuron are adjusted; however, the winning neuron needs to have a positive influence on its neuron neighbors (the neurons that physically surround it). Therefore, the weights for the neurons in the neighborhood around the winning neuron are changed during training, too. The size of the neighborhood can vary. For example, it may start off fairly large (sometimes even close to N) and decrease with learning until during the last training events, the neighborhood is zero, meaning by then only the winning neuron’s weights are changed. By that time, the learning rate is also very small, and the clusters are fairly well defined. The subsequent (small) changes in weight are only affecting refinements on the cluster arrangements.

The output slab can be viewed as either a one-dimensional slab of N elements with one-dimensional neighborhoods, or as a two-dimensional slab with two-dimensional neighborhoods. In the one-dimensional case, neighborhood size refers to the number of neurons to the left and right of the winning neuron that participate in the winner’s reward of weight changes.

In the two-dimensional case, the slab is viewed as a matrix or grid, with some number of rows and columns. The product of rows and columns must be equal to the total number of neurons, N. The size of the neighborhood refers to the number of rows up or down, and then number of columns left and right that define a small square around the winner.

The schemes, or equations, with which you lower learning rate and neighborhood size are up to you. They are largely problem dependent. Some common techniques are suggested in this chapter. For each problem, experimentation will be required before your Kohonen/LVQ network is successful. It is not at all as automatic as supervised Back Propagation learning.

When learning is complete, propagation of pattern will produce activation in the output neurons.

Initializing weights for Kohonen/LVQ

The main problem with Kohonen/LVQ is that sometimes one neuron starts winning all the time, not giving the others a chance to change weights and get closer. To avoid this problem you can use large neighborhoods, or you can adjust the weight vectors before you initiate training so that they are closer to the training patterns. The magnitude of your random weights should be approximately the magnitude of your pattern elements.

Choosing the maximum epochs for Kohonen/LVQ

Kohonen/LVQ lets you control the maximum number of iterations through the data set. With each cycle, the neighborhood and learn rate are reduced proportionally such that each parameter reaches zero by the end of the last cycle. Kohonen/LVQ performs this adjustment automatically as training progresses. At the end of the last cycle, training is complete.

Typical iterations through the data set range between 200 and 1000.

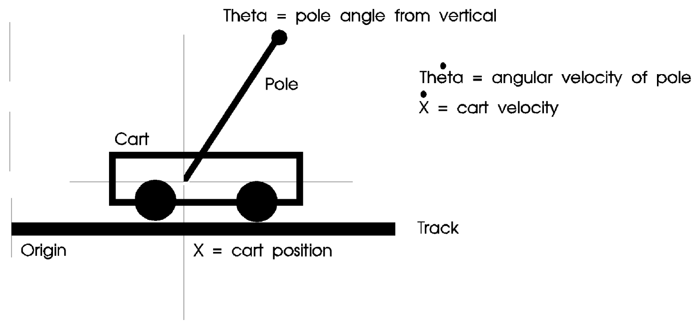
Types of Kohonen/LVQ

In Kohonen/LVQ, proximity of the pattern to the weight vectors is determined by measuring the Euclidean distance between them in P dimensional space, where P is the number of patterns. The activation in the neuron is actually the square of the distance between the pattern and the weight vector for that neuron. Therefore, the winner is the neuron with the minimum activation.

Unlike Back Propagation, Kohonen/LVQ will actually run fine without mapping to [0,1].

Balancing an Inverted Pendulum

Balancing an inverted pendulum is a classic example that can be solved with a neural network. As shown in the diagram below, the inverted pendulum is a two-dimensional dynamic problem, in which both the cart and pole are able to move in the same one-dimensional plane.

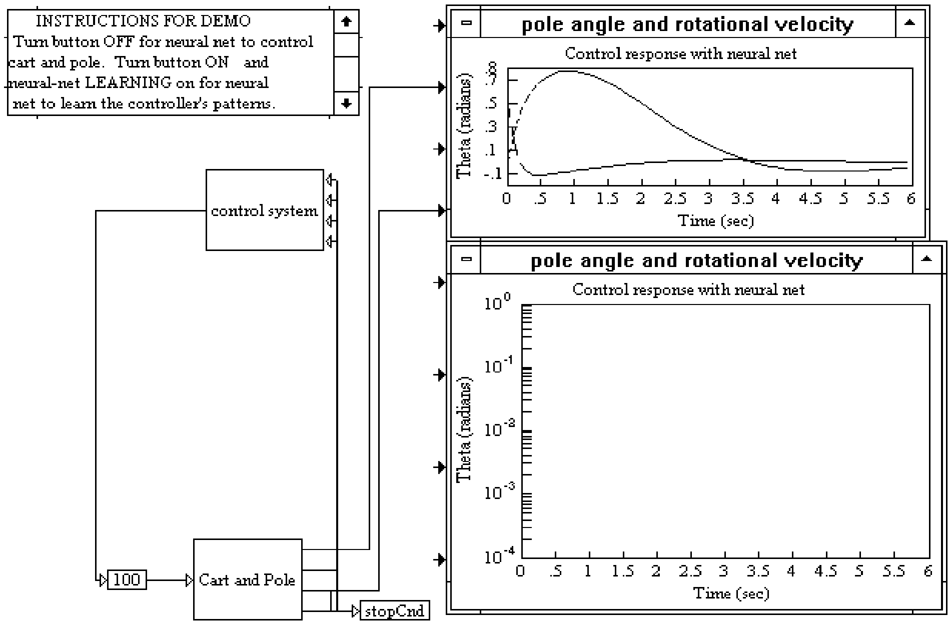


Theta = pole angle from vertical
Pole
Theta = angular velocity of pole
X = cart velocity
Cart
Track
Origin
X = cart position

The task for the neural network is to learn how to keep the pole balanced by moving the cart back and forth to counteract the non-zero angular velocity of the pole.

Balancing the inverted pendulum problem with a controller

A sample block diagram named CARTPOLE.VSM is automatically installed in \VISSIM90\NEURAL. This diagram simulates the control of an inverted pendulum using a classical controller is shown below.



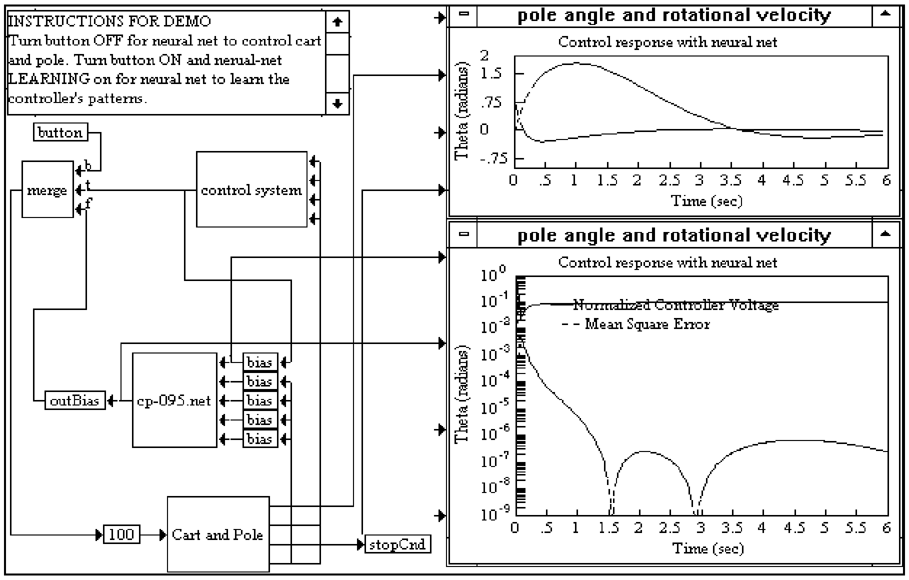
INSTRUCTIONS FOR DEMO
Turn button OFF for neural net to control cart and pole. Turn button ON and neural-net LEARNING on for neural net to learn the controller's patterns.

control system
100
Cart and Pole
stopCnd
pole angle and rotational velocity
Control response with neural net
Theta (radians)
Time (sec)
pole angle and rotational velocity
Control response with neural net
Theta (radians)
Time (sec)

The controller sends a voltage between 0 and 5 V to the cart. The voltage tells the cart which way and how far to move in order to keep the pole upright. The pole starts at 0.8 radians (Theta's initial value) from the vertical position. The STOPCND compound block stops the simulation if the pole falls too far past a vertical position.

Balancing the inverted pendulum with a neural network

The diagram below shows how to add a neural network in parallel to the classical controller, or known good system, and train the network to control the inverted pendulum.



INSTRUCTIONS FOR DEMO
Turn button OFF for neural net to control cart and pole. Turn button ON and nerual-net LEARNING on for neural net to learn the controller's patterns.

Several blocks have been added to the original diagram to integrate the neural network with the controller.

- The button and merge blocks are used to select either the controller or the neuralNet block output.
- The BIAS and OUTBIAS compound blocks are used to normalize the input data and re-normalize the output data. Using a Back Propagation algorithm, all inputs to a neural network must have values in the [0,1] range. Except during training, if you normalize the input data, you'll have to re-normalize the output data to its original or preferred range.

Setting the inputs and outputs for the inverted pendulum

To establish the inputs and outputs used for training the inverted pendulum neural network, click the right mouse button over the neuralNet block to access its dialog box.

There are four inputs for the inverted pendulum neural network:

- X, which controls the cart's position
- $\dot{X}$, which controls the cart's velocity
- Theta, which controls the angle position from vertical in radians
- $\dot{\text{Theta}}$, which controls the angular velocity

All four values must be normalized to the [0,1] range with the BIAS compound block for a Back Propagation algorithm to work.

The output from the inverted pendulum neural network is a continuous number in the [0,1] range that is re-normalized to be a voltage in the [0,5] range, just like the controller.

During training, the output of the neural network is the root mean square error of the neural network’s answer to the given inputs, and the correct result.

Terminology

This section contains…

Activation Function

A function that maps the output of the neuron from a possibly infinite domain into a pre-specified range. In VisSim/Neural-Net, activation functions are taken care of for you.

Back Propagation

The technical term for one of the algorithms used for supervised learning. Weight changes are sent back, or back propagated, through the network to modify weights after the output layer has been evaluated. Back Propagation is currently the most popular algorithm because, although it may train slowly, it nevertheless provides the best results for a wide variety of problems.

Bias Neuron

A single neuron in every slab (supervised learning). You cannot modify a bias neuron because it is always turned on (for example, set to 1), and because it is internal.

Epoch

A complete pass through the network of all the patterns in the training set.

Error Factor

A computed value equal to the sum of the squares of the differences between the predicted values and actual values for all output neurons in the slab (supervised learning).

Inputs

The variables used to make pattern classifications or forecast decisions. If, for example, you try to predict the Standard and Poor's stock price index for the following month based on the New York gold price, the average three-month treasury bill rate, and retail sales, the gold price, treasury bill rate, and retail sales are the inputs to the neural network, and the stock price index is the network's output.

Layer

A group of slabs in a network. There can be a single slab or multiple slabs in the same layer; however, there can only be one input layer and one output layer in a network. There may be multiple hidden layers.

Learning Rate

A variable that you set for regulating how much weights are changed when the network is trained. The amount of weight modification is proportional to the amount of error. The learning rate is the constant of proportionality. For example, if the learning rate is set to 0.5, then the weight change is a function of only half of the error. The larger the learning rate, the larger the weight changes, and therefore the faster the learning. Learning rates should not exceed 1 or the system could become unstable.

Link

A term used to designate the connection between slabs in a network. A single link represents all the weighted connections between individual neurons in one slab to individual neurons in a connected slab. For example, if you have one slab with three neurons connected to another slab with two neurons, the connection will be a single link, but that link will represent six weighted connections between individual neurons. In supervised learning, the bias neuron is also connected.

Momentum

A variable that computes the proportion of the last weight change to be added into the new weight change. Momentum provides a smoothing effect to the weight changes and allows for the use of larger learning rates. When momentum is not included in large learning rates, the results are an oscillation of weight changes. Learning may never complete, or the model may converge to a solution that is not optimal.

Network

A neural network that consists of groupings of layers plus links connecting the slabs in the layers.

Neuron

A data element that contains a value, usually in the range of 0 to 1 (written in this manual as [0,1]).

Slab

A grouping of neurons.

Outputs

The pattern classifications or predictions made by the neural network.

Propagate

A method of moving data from one layer to the next in a neural network. Neuron values in the preceding layer are multiplied by the weights to a neuron in the succeeding layer. The products are then summed. An activation function is applied to the sum and the result is placed in the neuron in the succeeding layer.

Supervised Learning

A method of training a neural network by presenting it with a data set containing sample input facts (or parameters or variables) and corresponding answers (or results). The four supervised learning methods available in VisSim/Neural-Net are Back Propagation, BP/Momentum, Probabilistic, and General Regression. See also “Unsupervised Learning.”

Test Set

A second set of patterns from which to determine the correct classification or answer. You may use the patterns in the test set to verify how well your network is working. You should not include the test set as part of your training set because the way to assess a neural network’s performance is to examine how well it generalizes on patterns it has never seen before.

Training Set

The set of patterns of inputs (and, for supervised learning, correct outputs) used to train the network.

Unsupervised Learning

A method of training a neural network where correct outputs in the sample patterns are not available. Kohonen/LVQ is the unsupervised learning algorithm available with VisSim/Neural-Net. Kohonen/LVQ learns to make pattern classifications by making its own clustering scheme for patterns. The patterns are clustered into categories based on their proximity to each other. See also “Supervised Learning.”

Weights

The values that represent connection strengths between neurons. The weights are changed as values are passed from one layer to another. To reinforce a connection positively, the weight is raised. Conversely, to inhibit a connection, the weight is lowered.

Installing VisSim/Neural-Net

The Install program that comes on your VisSim/Neural-Net disk installs the VisSim/Neural-Net program and other utility files on your hard disk.

Installation requirements

VisSim/Neural-Net runs on personal computers using the Intel 80286 or higher processor, including the IBM Personal System/2 Series, the IBM PC AT, and 100% compatibles. To use VisSim/Neural-Net, your computer must have the following components:

- Visual Solutions VisSim 9.0+
- 3MB of free hard disk space

Installation procedure

You use the Install program to install VisSim/Neural-Net on your computer for the first time or to upgrade your existing copy of VisSim/Neural-Net to a more recent version of the software.

To install VisSim/Neural-Net

1. After you download the setup program from www.vissim.com, run the setup program.
2. Follow the on-screen instructions.

Index