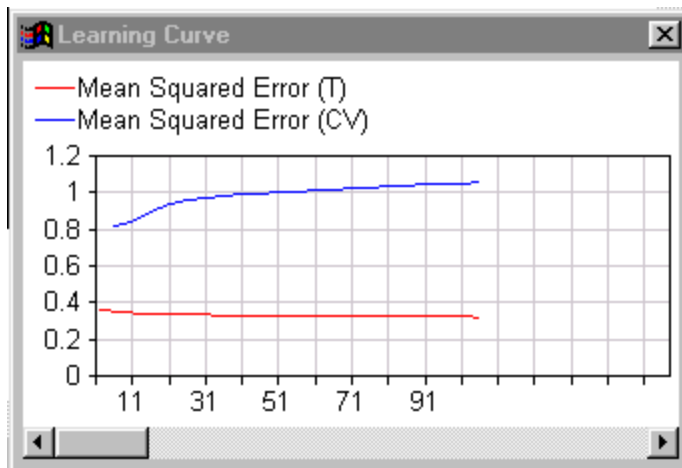


Learning Curve

This plot shows the mean squared error (MSE) of the network after each epoch of data. The epoch number is shown on the X-axis and the MSE is shown on the Y-axis. The MSE of the training set is shown in red and the MSE of the cross-validation set (if included) is shown in blue. A network that is training well should have a constantly decreasing slope of the training MSE (typically an exponential decay). As long as the training set learning curve is decreasing, the network is still training. If the training set learning curve is increasing or bouncing up and down, the network is probably not training well (try decreasing the learning rates).

When the cross validation learning curve starts to increase (even though the training learning curve is still decreasing), then the network is over specializing on the training data and the network should be stopped (or best weights saved).



Epoch

An epoch is a complete presentation of the training data to the network.

Data Sets

NeuroSolutions provides a facility to divide your data into multiple data sets. This is typically used to ensure that the network is not overspecializing on the training data (see generalization). The standard NeuroSolutions data sets are Training, Cross Validation and Testing (although others can be used).

Training data is the portion of the data used to actually train the network. This is normally the largest portion of your data.

Cross Validation data is used to intermittently validate the training. Periodically testing the network (no weight changes during cross validation) during training can help avoid overspecializing on the training data (see generalization). Typically, the network is either stopped or its state is saved (save best weights) when the cross validation error begins to rise.

Testing data is used to further validate the results of a trained network. Although the network was not trained with the cross validation data, the training may have been stopped using it. Therefore, the cross validation data is not truly “out-of-sample”. The testing data is data set aside to test the network after it has been trained and is truly “out-of-sample”.

Cross validation data is data set aside to test the network during training (the network parameters are not directly updated/trained with this data). It is used to stop the network training when the network starts to specialize too much on the training data. By constantly testing the network on data it has not seen during training (the cross validation data), we can determine when the network training should be stopped.

Train

Adjusting the parameters of the neural network to minimize the mean squared error between the output of the neural network and the desired output of the neural network.

Weight

The weights of a neural network are the parameters that determine its behavior. Training a neural network involves adjusting the weights of the neural network until it produces an output as similar as possible to the desired output.

Out-of-Sample

Out-of-sample data is data that has not been used in any way to train the network. It is used as a true test of network performance since the network has not "seen" it yet.

Test

Testing the network consists of injecting inputs to the network, computing the outputs, and possibly comparing them with the desired output of the network. Testing never modifies the network or its parameters.

Output-Desired Probe - Training Set

This probe plots the output of the network and the desired output of the network on the same plot. When the network is performing well, these two curves should be very similar – implying that the output of your network is very close to the desired output of your network. This version of the Output-Desired probe shows the data in the [training set](#), which can be misleading if the network is over specializing on the training data (see [generalization](#))

Output-Desired Probe - Cross Validation Set

This probe plots the output of the network and the desired output of the network on the same plot. When the network is performing well, these two curves should be very similar – implying that the output of your network is very close to the desired output of your network. This version of the Output-Desired probe shows the data in [the cross validation set](#), which shows the expected out-of-sample performance of the network.

Mean Squared Error Matrix Viewer - Training Set

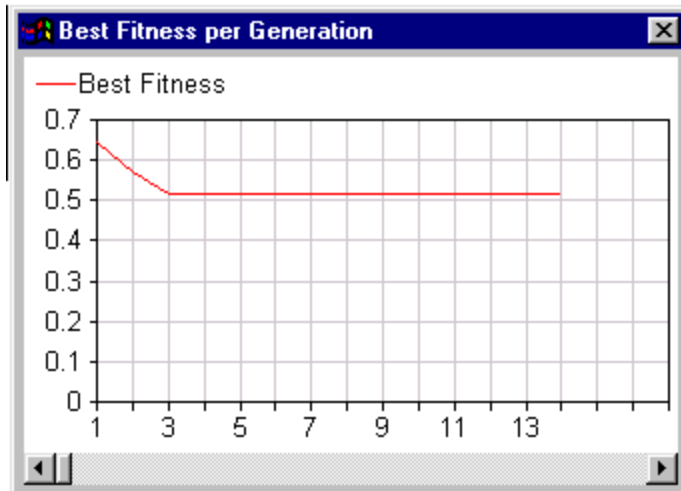
This probe shows the actual value of the current mean squared error (MSE) of the network using the [training set](#). This MSE can be misleading if the network is overspecializing on the training data (see [generalization](#))

Mean Squared Error Matrix Viewer - Cross Validation Set

This probe shows the actual value of the current mean squared error (MSE) of the network using the [cross validation set](#).

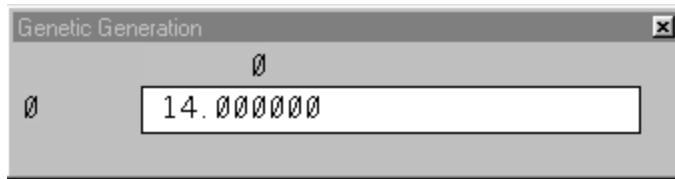
Best Fitness Plot

This probe (if genetic optimization is enabled) shows the MSE of the best network in each generation of the [Genetic optimization](#). The X-axis shows the generation number and the Y-axis shows the MSE of the best network (typically the cross validation MSE if cross validation is enabled). You can use this plot to see how well the genetic algorithm is optimizing your network (the MSE should decrease from generation to generation).



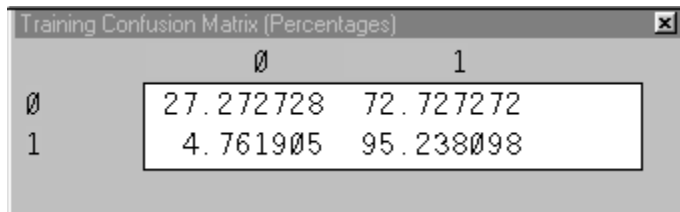
Genetic Generation Matrix Viewer

This probe shows the generation number (if genetic optimization is enabled) that the [genetic optimization](#) algorithm is currently executing.



Confusion Matrix - Training Set

This probe is used in classification problems to determine the number of correct and incorrect classifications by the network. The horizontal axis indicates the true class of the input and the vertical axis indicates the network's prediction of the class. Therefore, correct classifications are contained on the diagonal of the confusion matrix (position (1,1), (2,2), etc.). All other entries are incorrect classifications, but also indicate where the network is confused (e.g. it is incorrectly classifying many of class 1 inputs as class 3, etc.) This probe's default setting is to show percentages, but can also be configured to show absolute numbers. Since this probe is set on the [training data set](#), it may incorrectly represent the performance of the network on new data.

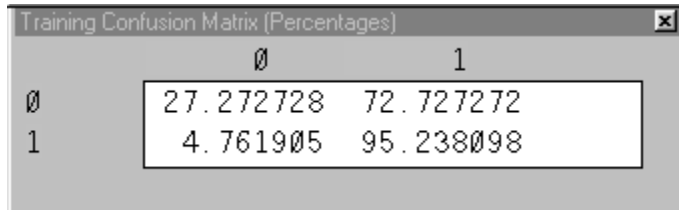


A screenshot of a software window titled "Training Confusion Matrix (Percentages)". The window displays a 2x2 confusion matrix for two classes, 0 and 1. The horizontal axis represents the true class, and the vertical axis represents the predicted class. The matrix shows that for class 0, 27.272728% were correctly classified and 72.727272% were incorrectly classified. For class 1, 4.761905% were incorrectly classified as class 0, and 95.238098% were correctly classified.

	0	1
0	27.272728	72.727272
1	4.761905	95.238098

Confusion Matrix - Cross Validation Set

This probe is used in classification problems to determine the number of correct and incorrect classifications by the network. The horizontal axis indicates the true class of the input and the vertical axis indicates the network's prediction of the class. Therefore, correct classifications are contained on the diagonal of the confusion matrix (position (1,1), (2,2), etc.). All other entries are incorrect classifications, but also indicate where the network is confused (e.g. it is incorrectly classifying many of class 1 inputs as class 3, etc.) This probe's default setting is to show percentages, but can also be configured to show absolute numbers. Because this probe is set on the [Cross Validation set](#), it is more likely to represent the true performance of the network on new data.



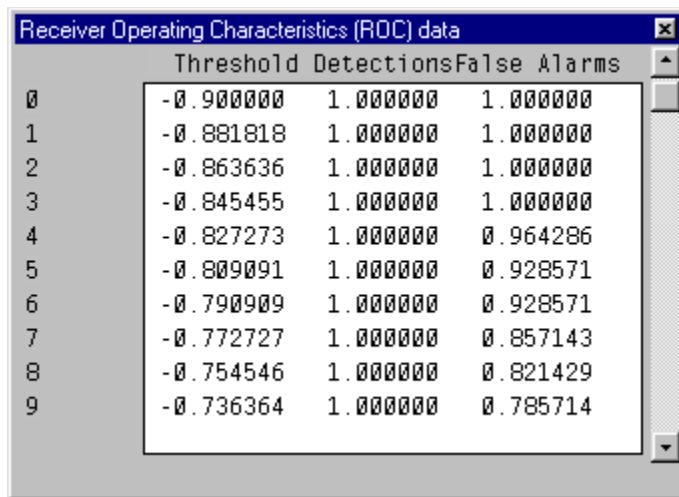
A screenshot of a software window titled "Training Confusion Matrix (Percentages)". The window displays a 2x2 confusion matrix for two classes, 0 and 1. The horizontal axis represents the true class, and the vertical axis represents the predicted class. The matrix shows that for class 0, 27.272728% were correctly classified and 72.727272% were incorrectly classified. For class 1, 4.761905% were incorrectly classified as class 0, and 95.238098% were correctly classified.

	0	1
0	27.272728	72.727272
1	4.761905	95.238098

Receiver Operating Characteristics (ROC) Curve

This probe is used to determine optimal thresholds in a two class problem (e.g. in the class or not in the class). The ROC curve shows the number of false positives (input not in class, but network says it is) and the number of false negatives (input in class, but the network says it isn't) for various threshold levels of the output PE. In some cases, false positives are much better or worse than false negatives. Using the ROC curve you can find the right threshold to create a classification based on the desire to minimize false positive or false negatives.

Although this probe does not actually show the ROC curve, it gives you the table of values needed to create this curve.



	Threshold	Detections	False Alarms
0	-0.900000	1.000000	1.000000
1	-0.881818	1.000000	1.000000
2	-0.863636	1.000000	1.000000
3	-0.845455	1.000000	1.000000
4	-0.827273	1.000000	0.964286
5	-0.809091	1.000000	0.928571
6	-0.790909	1.000000	0.928571
7	-0.772727	1.000000	0.857143
8	-0.754546	1.000000	0.821429
9	-0.736364	1.000000	0.785714

Classification Breadboard

This neural network is called a [Multi-Layer Perceptron](#) (MLP). It is the most common [supervised](#) neural network. It consists of multiple layers of [PEs](#) connected in a feedforward fashion. The PEs in NeuroSolutions are the orange circular icons and are called [axons](#). The connections between the PEs are the icons with horizontal and diagonal lines between the axons and are called [synapses](#). NeuroSolutions uses the backpropagation of errors to train the MLP. The smaller icons on top of the axons and synapses are called [backpropagation components](#) and pass the error backwards from the end of the network to the beginning. The green axons on top of the backpropagation components are called [gradient search](#) components and adjust the weights contained in the synapses and axons – this is how the network is trained.

Networks constructed using the “low complexity” setting will have one [hidden layer](#). Those with medium or high complexity will have two hidden layers. The number of PEs in the first hidden layer is contained in the properties of the 2nd AXON from the left. The number of PEs in the second hidden layer is contained in the properties of the 3rd AXON from the left. The right most axon is called the criterion and reads the desired file from the attached [file component](#) and determines the error in the network. The axon 2nd from the right is the output axon and generates the actual network outputs. The axon on the far left is called the input axon and doesn't do anything but accept the input from the file component.

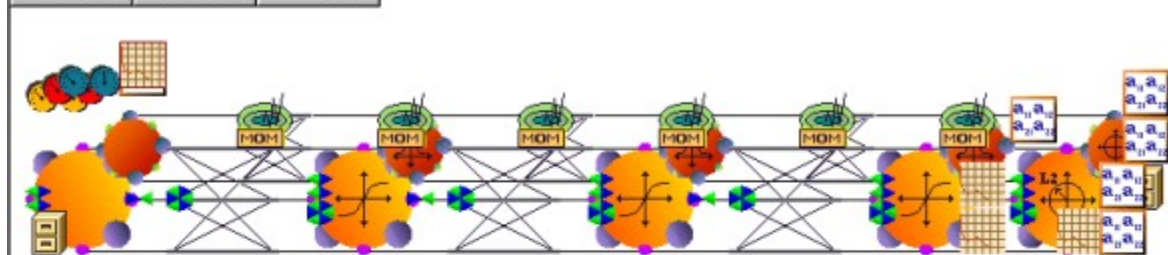
Click on any object in the figure below to get a description of the component or probe.

NeuroSolutions - [test.nsb]

File Edit Alignment Tools View Window Help



Explain Modify Test



Genetic Generation

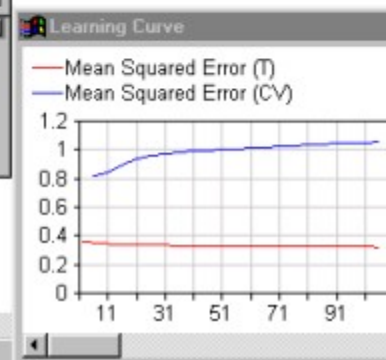
0

14.000000



Training Confusion Matrix (Percentages)

	0	1
0	27.272728	72.727272
1	4.761905	95.238098



Ready

Problem Type Selection

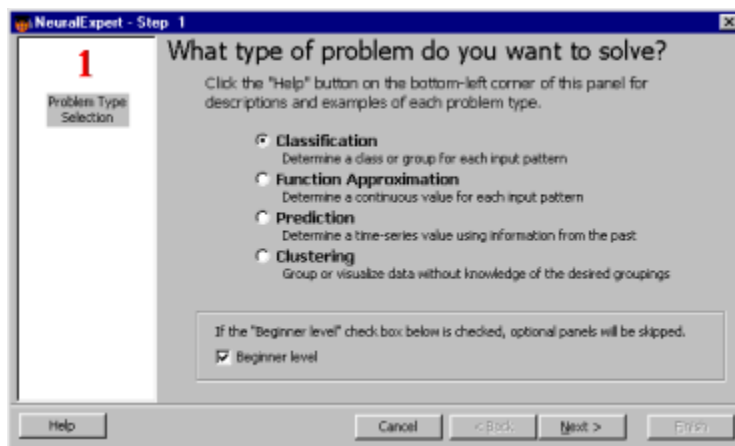
Welcome to the Neural Expert. This wizard asks you questions and automatically builds the best network and configures the parameters and [probes](#) for your problem. Navigation through this wizard is simple. At the bottom right side of the panel you will see 4 buttons (of which only some may be active). Normally you use the next button once you have finished answering the question(s) on the current page. When you are done, the “finish” button becomes active. When pressed, the Neural Expert will build your network in NeuroSolutions. “cancel” stops the process and doesn’t make any changes to NeuroSolutions. The “back” button allows you to go to previous panels and make changes or view your selections.

In addition, once you select a problem type you will see all the questions you will need to answer in the panel on the left. You can click on these steps/numbers to navigate through the question and answer session. Typically, however, you cannot move forward in the wizard unless you have successfully answered all the questions leading up to the panel you are trying to move to.

This panel asks you to select a problem type. The four currently available problem types in the Neural Expert are Classification, Prediction, Function Approximation, and Clustering. You must select one of these problem types. If your problem does not fit one of these descriptions (or you would like to completely specify the architecture and parameters yourself), please use the NeuralBuilder on the tools menu.

For a description and examples of each problem type, click [here](#).

This panel also asks you to select “beginner’s mode”. If you do, the NeuralExpert will make choices for you concerning the [cross validation data](#) and [testing data](#), thus reducing the number of choices you will have to make.



Problem Descriptions

Classification problems are those where the goal is to label each input with a specified classification. A simple example of a classification problem is to try to label people as male or female (the two classes, also the desired output) using their height and weight (the input). Notice that the input can be numeric or [symbolic](#), but the output is symbolic in nature. For instance, in the example above, the desired output is male or female, not a numeric value.

Function Approximation problems are those where the goal is to determine a numeric value given a set of inputs. This is similar to classification problems except that the output is numeric. An example is to determine the wind chill factor (desired output, numeric) given the temperature, humidity, and wind speed. These problems are called function approximation because the neural network will try to approximate the functional relationship between the input and desired output. Prediction problems are also function approximation problems except that they use temporal information (e.g. the past history of the input data) to make predictions of the data.

Prediction problems are those where the goal is to determine an output given a set of inputs and the past history of the inputs. The main difference between prediction problems and the others is that prediction problems use the current input and previous inputs (the temporal history of the input) to determine either the current value of the output or a future value of a signal. A typical example is to use the temporal history of a stock closing price (input, e.g. today's and three previous day's prices) to try to predict tomorrow's closing price (desired output).

Clustering problems are those where you want to extract information from the input data but don't have a desired output. For instance, you have survey data from various people. You would like to cluster the people into groups with similar buying habits. The fundamental difference between the clustering problem and the others is that there is no desired output (therefore you do not have an error and cannot train using [backpropagation](#)).

See also the [table of example problems and problem types](#)

Table of Problem Descriptions

Input	Goal	problem type
Height and weight of people	Determine whether person is male or female	Classification
Stock price, volume, variance, price history	Determine whether the stock is a technology stock or not	Classification
Stock price, volume, variance, price history	Predict tomorrow's value of the stock	Prediction
Temperature, humidity, and wind speed	Determine wind chill factor	Function approximation
Survey form data	Group and cluster people with similar buying habits	Clustering
Historical number of sunspots	Predict the number of sunspots tomorrow	Prediction
Size, shape, and color of mushrooms	Determine type of mushroom	Classification
Neighborhood characteristics	Predict average price of house in the neighborhood	Function approximation
Past history of a power plant's controls and output	Predict control values to drive plant to specified output	Prediction

Input File Specification

“Where is your input file located?”

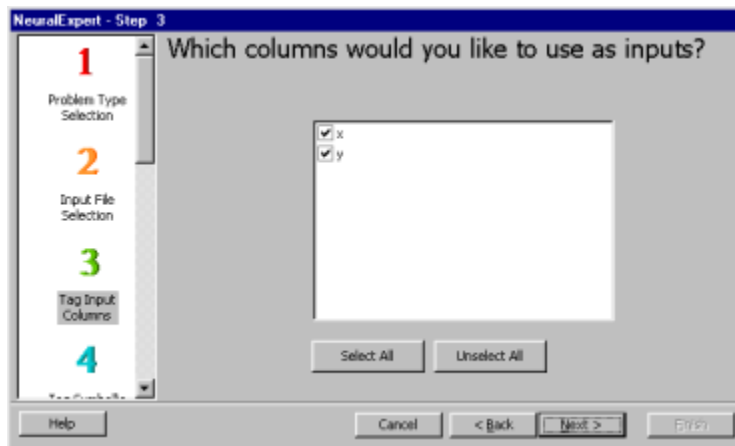
The Neural Expert works only with ascii column data (ascii files with individual inputs in columns). In this panel you select the file to be used as input to the network. There are three ways to do this. First, you may click the “browse” button and search through the standard windows tree structure to find your file. Second, you may type the path and filename in the text box. Third, you may click the triangle at the right edge of the text box and you will be presented with a list of the most recently used text files in the Neural Expert.



Which Columns Would You Like to Use as Inputs?

ASCII column data typically has column labels as the first row. If they do not, the wizard will ask you if you'd like to add column labels. This panel asks you to select which of the columns should be used as input to your network. By default, all columns are initially checked. For example, let's say your data has 4 columns labeled "temperature", "humidity", "wind_speed", and "wind_chill" and you would like to use the first 3 columns to predict the fourth. You would click on the box to the left of "wind_chill" to deselect that column – to indicate that the first three are to be used as inputs to the network. Later, you can come back and select only the two of the inputs and compare your results.

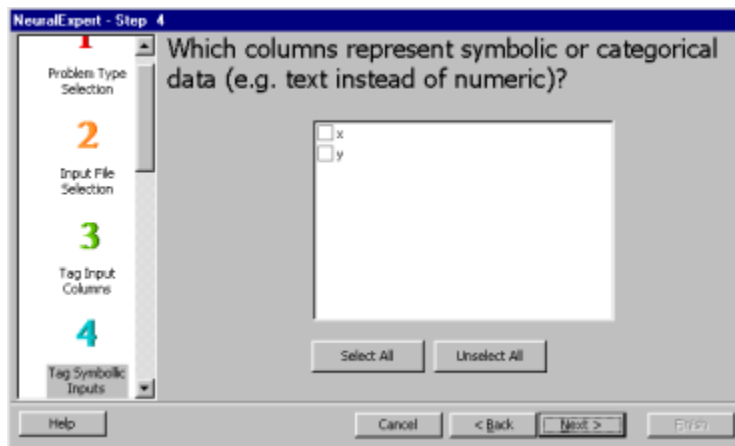
Two buttons are also included which allow you to "select all" columns or "unselect all" columns.



Tag Symbolic Columns

This panel allows you to specify which of the columns that you selected in the previous panel (columns to be used for input) actually contain symbolic data. Symbolic data is data that is not numeric. Examples of symbolic data are: yes/no, high/medium/low, etc. In NeuroSolutions, symbolic data is translated into numeric data. For instance, high/medium/low would be converted to three columns of data representing high (on/off), medium (on/off), or low (on/off). High would be represented in these three columns as 1, 0, 0. The Neural Expert will briefly scan the file to try to determine which columns are symbolic.

Two buttons are also included which allow you to “select all” columns or “unselect all” columns.

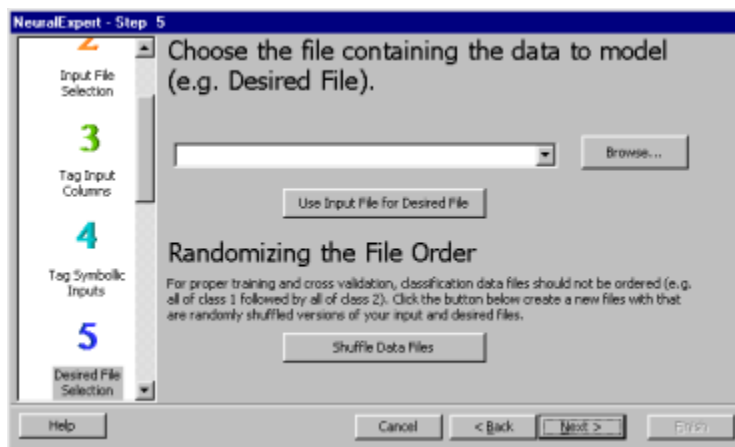


Choose Desired File

The Neural Expert works only with ascii column data (ascii files with individual inputs in columns). In this panel you select the file to be used as the desired output of the network. There are four ways to do this. First, you may click the “browse” button and search through the standard windows tree structure to find your file. Second, you may type the path and filename in the text box. Third, you may click the triangle at the right edge of the text box and you will be presented with a list of the most recently used text files in the Neural Expert. Lastly, you may click the “use Input File as Desired File” button, which will place the input file name in the desired file text box.

For classification problems you will be asked if you would like to randomize the order of your data before presenting it to the network. Neural networks train better if the presentation of the data is not ordered. For instance, if you are classifying between two classes male and female, the network will train much better if the male and female data are intermixed, rather than all the males followed by all the females. If your data is highly ordered, you should randomize the order before training the neural network.

When you click the “shuffle data files”, the order of the data in both the input and desired files will be randomized. The Neural Expert will create new file(s) that have the extension “.rnd”.

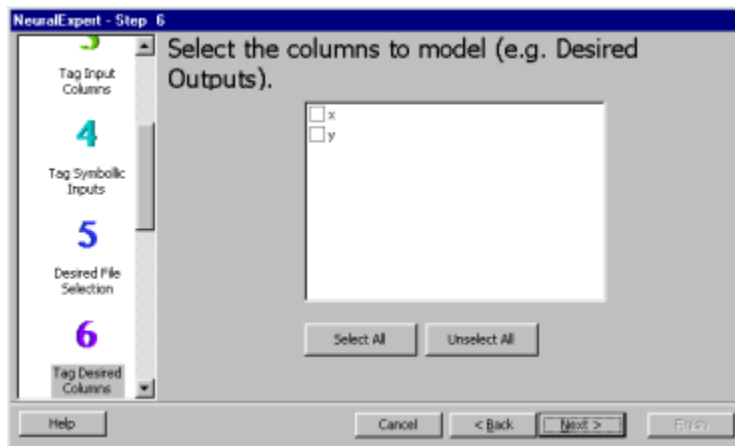


Tag Desired Columns

ASCII column data typically has column labels as the first row. If they do not, the wizard will ask you if you'd like to add column labels. This panel asks you to select which of the columns should be used as input to your network. By default, all columns are initially checked. For example, let's say your data has 4 columns labeled "temperature", "humidity", "wind_speed", and "wind_chill" and you would like to use the first three columns to predict the fourth. You would click on the box to the left of "wind_chill" to select that column as the desired output. If you use the same file for both the input and desired output, the Neural Expert will automatically select all columns not used in the input. If you use a different file, the Neural Expert will turn all columns on by default.

Two buttons are also included which allow you to "select all" columns or "unselect all" columns.

Note that for classification and function approximation you should not select the same column for both input and desired output – in this case the network can simply pass the correct input through to the output and simply solve the problem.

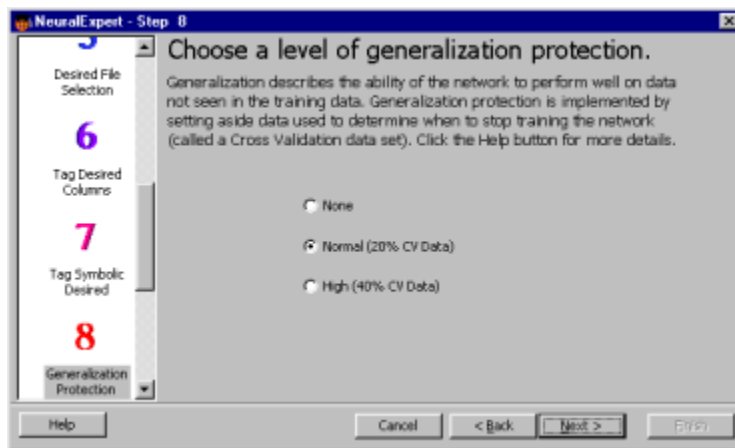


Generalization Protection

One of the primary goals in training neural networks is to ensure that the network performs well on data that it has not been trained on (called “generalization”). The standard method of ensuring good generalization is to divide your training data into multiple data sets. The most common data sets are the training, cross validation, and testing data sets. The cross validation data set is used by the network during training. Periodically, while training on the training data set, the network is tested for performance on the cross validation set. During this testing, the weights are not trained, but the performance of the network on the cross validation set is saved and compared to past values. If the network is starting to overtrain on the training data, the cross validation performance will begin to degrade. Thus, the cross validation data set is used to determine when the network has been trained as well as possible without overtraining (e.g. maximum generalization).

Although the network is not trained with the cross validation set, it uses the cross validation set to choose a “best” set of weights. Therefore, it is not truly an out-of-sample test of the network. For a true test of the performance of the network the testing set is used. It provides a true indication of how the network will perform on new data.

By default, 80% of the data in the input and desired files will be used for training and 20% will be used for cross validation. Every 5 [epochs](#), the learning will be turned off and the network will be tested on the cross validation data set. The network is “optimal” when the error in the cross validation set is at its minimum position. By selecting “high” generalization protection, 40% of your data will be set aside for cross validation. This option should only be used when you have a great deal of data (e.g. 10,000 records or more).

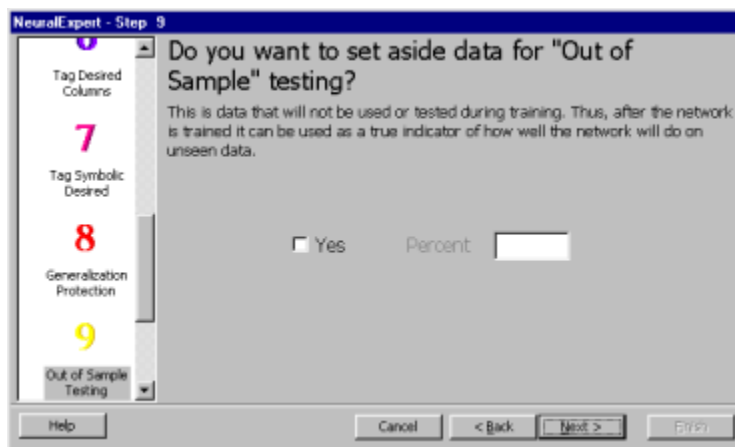


Out of Sample Testing

One of the primary goals in training neural networks is to ensure that the network performs well on data that it has not been trained on (called “generalization”). The standard method of ensuring good generalization is to divide your training data into multiple data sets. The most common data sets are the training, cross validation, and testing data sets. The cross validation data set is used by the network during training. Periodically, while training on the training data set, the network is tested for performance on the cross validation set. During this testing, the weights are not trained, but the performance of the network on the cross validation set is saved and compared to past values. If the network is starting to overtrain on the training data, the cross validation performance will begin to degrade. Thus, the cross validation data set is used to determine when the network has been trained as well as possible without overtraining (e.g. maximum generalization).

Although the network is not trained with the cross validation set, it uses the cross validation set to choose a “best” set of weights. Therefore, it is not truly an out-of-sample test of the network. For a true test of the performance of the network the testing set is used. It provides a true indication of how the network will perform on new data.

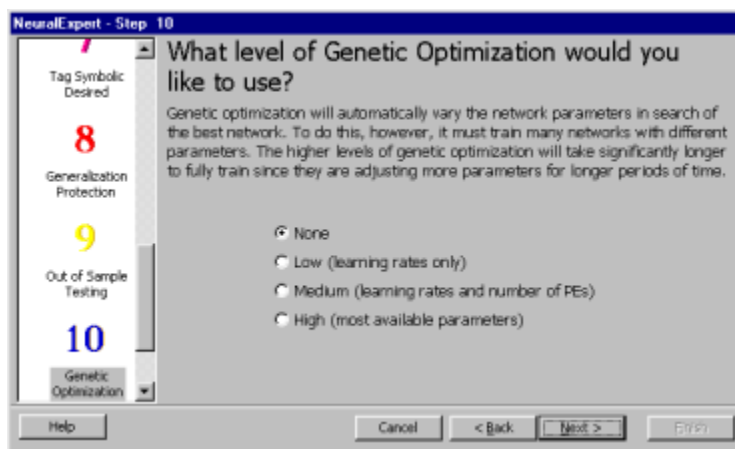
To set aside completely “out of sample” data to be used as a testing set, click the “yes” check box and select the percent of the data to be set aside. The percentage should vary depending on the amount of data you have and how rigorously you wish to test the network on out of sample data. The default value is 20%.



Genetic Optimization

Genetic optimization determines the best network parameters by successively trying and testing different combinations of parameters. Like evolution, good parameter sets are more likely to survive from one population to the next. Genetic optimization can be used to set many of the parameters in a neural network (e.g. number of [PEs](#), [learning rates](#), input selection, etc.). In NeuroSolutions, the genetic optimization takes place by repeatedly training the network over and over with various parameters and calculating the best MSE for each network. Because of this, the time needed to train a neural network with genetic optimization on may be very large.

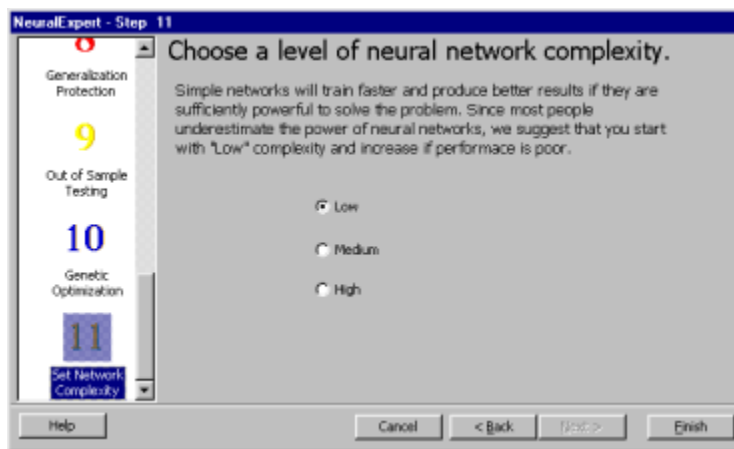
By default, the Neural Expert does not do genetic optimization. Selecting the “low” level of genetic optimization allows the genetic optimizer to determine the best learning rates. The “medium” level sets the best learning rates and number of PEs in each layer. The “high” level allows the genetic optimizer to set most of the parameters in the neural network.



Set Network Complexity

In general, smaller neural networks are preferable over large ones. If a small one can solve your problem sufficiently (you would be surprised how powerful small networks are), then a large one will not only require more training and testing time but also may perform worse on new data. This is the generalization problem. The larger the neural network, the more free parameters it has to solve the problem. Excessive free parameters may overfit the data, however, causing the network to overspecialize or memorize the training data. When this happens, the MSE of the training data will be much better than the MSE of the cross validation or testing data sets.

We strongly recommend starting with a “low complexity” network. After using a low complexity network, you can move to a “medium” or “high” complexity network and see if the performance is significantly better. But be warned, that “medium” or “high” complexity networks require a large amount of data to adequately train (a rule of thumb is no less than $30N$, where N is the number of input columns).



Prediction Offset

In prediction problems, the data should be organized such that different data fields are arranged in columns and different times that data was measured corresponds to the columns. Often times you want to predict values in the future, which corresponds to the desired data being taken from a row that is further down than the input data. To dictate the number of samples in advance you want to predict, type the number into the prediction length box. For instance, if you want to use the current values of x, y, and z to predict the next value of z, then you place 1 in the prediction length box. This will offset the desired data from the input data.

If your data is already aligned so that the value you want to predict is in the same row as the input data, then you place a 0 in the prediction length box.

If your prediction length is greater than 1, you have the option of using iterative prediction. If you wish to predict 5 samples in advance and you do not use iterative prediction, the network will use the data at time t to directly predict the desired data at time $t+5$. If every one of your inputs is also an output (e.g. you are predicting the entire next data record), then you can use iterative prediction. Iterative prediction always predicts one sample in advance (which is easier than multiple steps in advance). In order to predict 5 samples in advance, however, it uses its predicted outputs at time $t+1$ as inputs to the network to predict the values at time $t+2$. These outputs are then used to predict the values at time $t+3$, etc. until it reaches its desired prediction length. There are times when iterative prediction can greatly outperform standard prediction. If you are doing multiple step prediction, we suggest you try both ways and compare the results.

NeuralExpert - Step 7

Tag Symbolic Inputs

5

Desired File Selection

6

Tag Desired Columns

7

Prediction Offset

How many samples ahead do you want to predict the desired output?

If you want to use the current and past inputs to predict the current desired output, set prediction length to 0. To use the current and past inputs to predict future desired outputs, enter the number of samples in the future to predict.

Prediction Length

Do you want to use Iterative Prediction?

Iterative prediction uses multiple one-step predictions to predict multiple steps ahead. The one-step predictions are used as inputs to the network to predict further in the future.

☐ Iterative Prediction?

Help Cancel < Back Next > Exit

Choose Number of Clusters

When clustering data in a data mining application, it is useful to know approximately how many clusters or groups you are hoping to find in your data. The clustering algorithm will have vastly different performance if you wish to separate data (say 10,000 records) into either 3 groups or 50 groups. Enter the approximate number of clusters you wish to discriminate between in the text box. Obviously this number should not be close to the total number of data records in the input file.

The screenshot shows a software window titled "NeuralExpert - Step 5". On the left is a vertical sidebar with four steps: "Input File Selection" (orange arrow), "Tag Inputs" (green number 3), "Tag Symbolic Inputs" (blue number 4), and "Choose Number of Clusters" (blue number 5, which is the current step). The main area of the window contains the text "Approximately how many clusters would you like to find in the data?" and a text input box with the number "5" entered. At the bottom of the window are four buttons: "Help", "Cancel", "< Back", and "Next > Finish".

Learning Curve Icon

Double-clicking this icon opens the [learning curve](#) probe.



It is a [datagraph](#) probe.

Output-Desired Probe - Training Set Icon

Double-clicking this icon opens the [output-desired probe – training set](#).



It is a [datagraph](#) probe.

Output-Desired Probe - Cross Validation Set Icon

Double-click this icon to opens the [output-desired probe – cross validation set](#).



It is a [datagraph](#) probe.

Mean Squared Error Matrix Viewer - Training Set Icon

Double-clicking this icon opens the [mean squared error matrix viewer – training set](#).



It is a [matrix viewer](#) probe.

Mean Squared Error Matrix Viewer - Cross Validation Set Icon

Double-clicking this icon opens the [mean squared error matrix viewer – cross validation set](#).



It is a [matrix viewer](#) probe.

Best Fitness Plot Icon

Double-clicking this icon opens the [best fitness plot](#).



It is a [datagraph](#) probe.

Genetic Generation Matrix Viewer Icon

Double-clicking this icon opens the [genetic generation matrix viewer](#).



It is a [matrix viewer](#) probe.

Confusion Matrix - Training Set Icon

Double-clicking this icon opens the [confusion matrix – training set](#).



It is a [matrix viewer](#) probe.

Confusion Matrix - Cross Validation Set Icon

Double-clicking this icon opens the [confusion matrix – cross validation set](#).



It is a [matrix viewer](#) probe.

Receiver Operating Characteristics (ROC) Curve Icon

Double-clicking this icon opens the [receiver operating characteristics \(ROC\) curve](#).

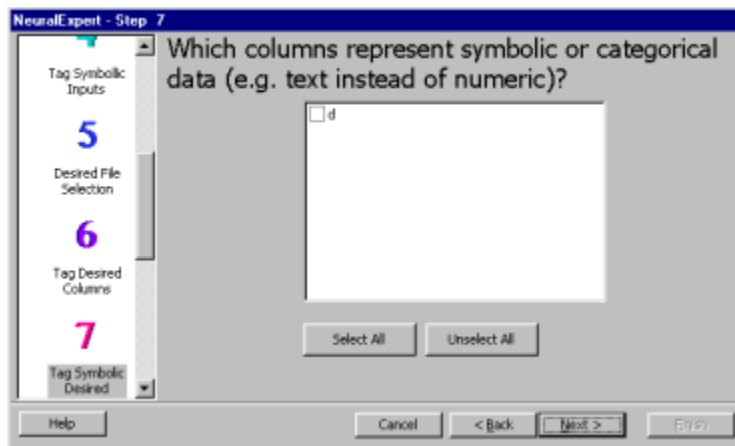


It is a [matrix viewer](#) probe.

Tag Desired Symbolic Columns

This panel allows you to specify which of the columns that you selected in the previous panel (columns to be used for input) actually contain symbolic data. Symbolic data is data that is not numeric. Examples of symbolic data are: yes/no, high/medium/low, etc. In NeuroSolutions, symbolic data is translated into numeric data. For instance, high/medium/low would be converted to three columns of data representing high (on/off), medium (on/off), or low (on/off). High would be represented in these three columns as 1, 0, 0. The Neural Expert will briefly scan the file to try to determine which columns are symbolic.

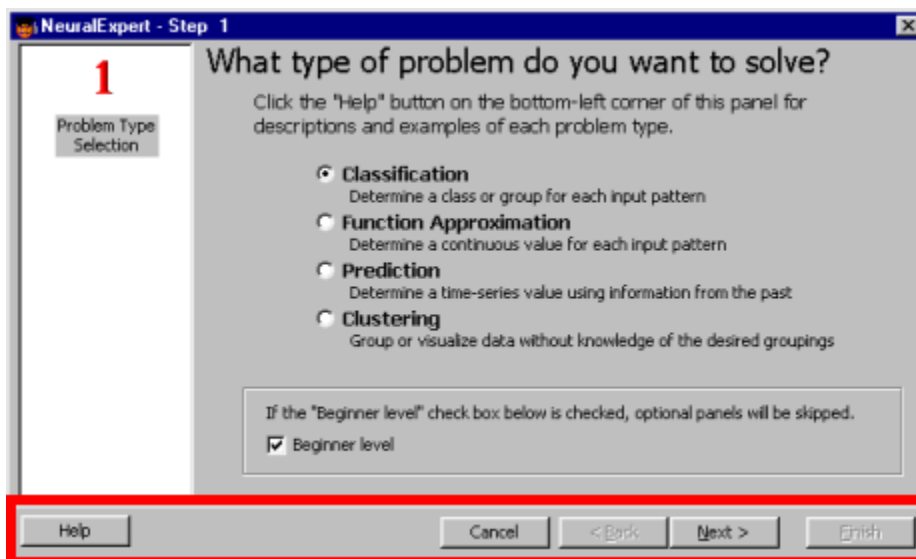
Two buttons are also included which allow you to “select all” columns or “unselect all” columns.




Introduction

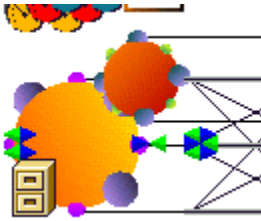
Welcome to the Neural Expert. This wizard asks you questions and automatically builds the best network and configures the parameters and [probes](#) for your problem. Navigation through this wizard is simple. At the bottom right side of the panel you will see 4 buttons (of which only some may be active). Normally you use the next button once you have finished answering the question(s) on the current page. When you are done, the “finish” button becomes active. When pressed, the Neural Expert will build your network in NeuroSolutions. “cancel” stops the process and doesn’t make any changes to NeuroSolutions. The “back” button allows you to go to previous panels and make changes or view your selections.

In addition, once you select a problem type you will see all the questions you will need to answer in the panel on the left. You can click on these steps/numbers to navigate through the question and answer session. Typically, however, you cannot move forward in the wizard unless you have successfully answered all the questions leading up to the panel you are trying to move to.



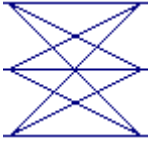
Input Axon

This component is a place holder [axon](#) that simply accepts the inputs to the system and passes it on to the rest of the neural network. The [file component](#)  on the bottom-left is used to read the data from the file system.



First Hidden Layer


These two components make up the first hidden layer of the neural network. . The [synapse](#)

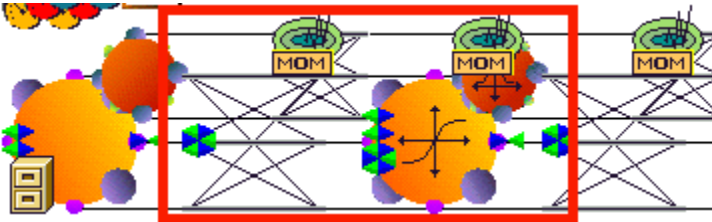


is the first component in the red box and makes the connection between each input and each [processing element](#) (PE) in the hidden layer. The synapse contains the connections and the trainable [weights](#) for each connection. The second component is the [tanh axon](#)



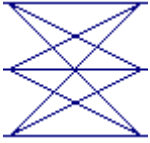
. This component has the processing elements for the hidden layer, each of which sums the weighted connections from the inputs. If you want to change the number of hidden layer PEs, open the inspector of this tanh axon and change the rows or columns. The number of weights in the synapse will be changed automatically.

The green components on top of the axon and synapse are the [momentum](#)  [gradient search components](#). They adjust the weights with information about the error in the network. To change the learning rates (a.k.a. step sizes), open the inspector on the momentum components.



Second Hidden Layer


These two components make up the second hidden layer of the neural network. The [synapse](#)



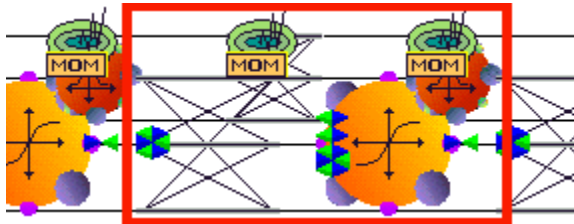
is the first component in the red box and makes the connection between each output of the first hidden layer and each [processing element](#) (PE) in the second hidden layer. The synapse contains the connections and the trainable [weights](#) for each connection. The second component is the [tanh axon](#)



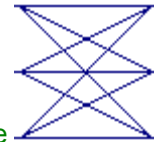
. This component has the processing elements for the hidden layer, each of which sums the weighted connections from the first hidden layer. If you want to change the number of hidden layer PEs, open the inspector of this tanh axon and change the rows or columns. The number of weights in the synapse will be changed automatically.

The green components on top of the axon and synapse are the [momentum](#)  [gradient search components](#). They adjust the weights with information about the error in the network. To change the learning rates (a.k.a. step sizes), open the inspector on the momentum components.

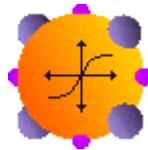
In many low and medium complexity networks, the second hidden layer will have only one PE in the second hidden layer, thus effectively creating a one-hidden layer neural network.



Output Layer (Classification)

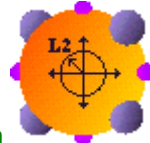


These two components make up the output layer of the neural network. The synapse is the first component in the red box and makes the connection between each second hidden layer PE and each output processing element (PE). The synapse contains the connections and the trainable weights for each connection.



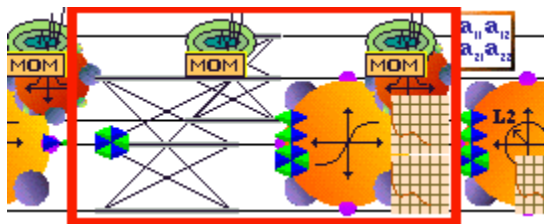
The second component is the tanh axon. This component has the processing elements for the output layer, each of which sums the weighted connections from the second hidden layer. For classification problems, the output is a tanh axon that saturates at ± 1 – this is ideal for classification.

There is one output PE for each desired output specified in the desired file. The desired file is shown in



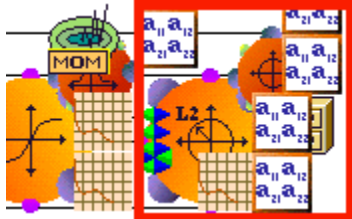
the file component on the right side of the criterion. You should not change the number of PEs in the output axon directly. It will be done automatically if you change the number of desired outputs.

The green components on top of the axon and synapse the momentum gradient search components. They adjust the weights with information about the error in the network. To change the learning rates (a.k.a. step sizes), open the inspector on the momentum components.




Criterion


This component is the criterion. It accepts the output(s) of the network and the desired output(s) and compares them. It computes the error and passes this error to the backpropagation components which adjust the weights of the network for training. Because the criterion has the information about the performance of the network, it contains many [access points](#) for network performance. In most networks, there will be a number of [probes](#) on the criterion (click on them separately for an explanation).

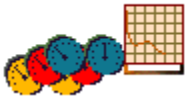


Controllers

These three controllers contain the global control parameters for the network. The image of the yellow dials  is the [static controller](#). It contains parameters such as the number of epochs per run, the number of exemplars per epoch, the data sets to use, etc.

The image of the red dials  is the [backprop controller](#). It contains learning parameters like the learning mode (batch, online, etc.).

The image of the green dials  (if present) is the [genetic controller](#). It contains the parameters associated with the genetic optimization.

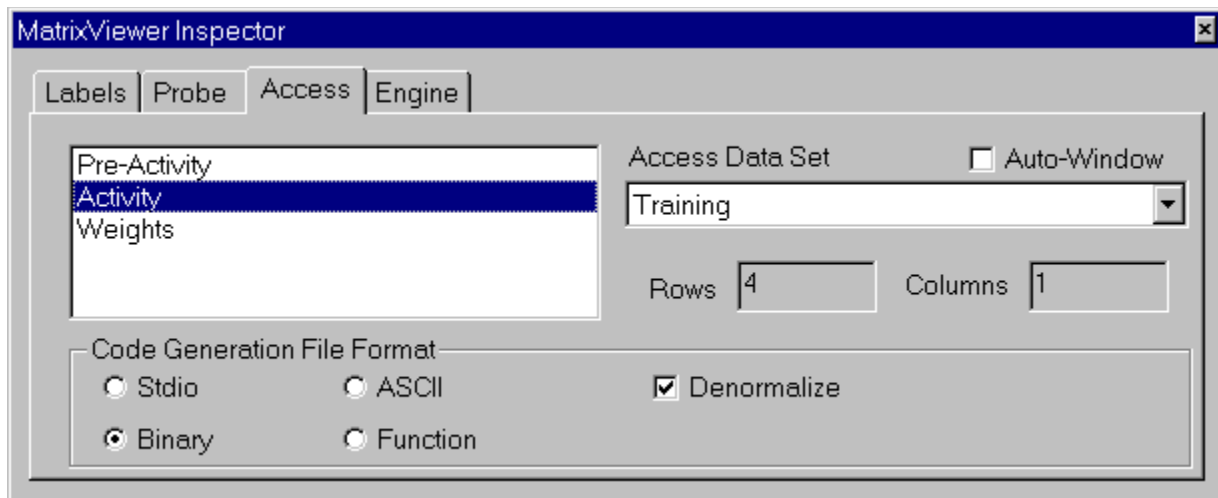


Symbolic Columns

Data in ASCII column format can be either numeric or symbolic. Symbolic data indicates that each data element in the column is a string of characters (e.g. yes/no), which may or may not be numeric. NeuroSolutions automatically translates symbolic columns by number each different string in the column and replacing each string with its numeric equivalent. When the file is translated, symbolic columns are replaced by numeric representations of the symbols contained within the symbol translation file (*.nss).

Accessing data

A very powerful feature of NeuroSolutions is the ability to view the data and parameters in all of the different locations in the network. Because many components have multiple sets of data that you might want to “probe” or view, each component has a list of “access points”. The access points are the different data for each component that you have access to. For instance, axons may have the following three access points: pre-activity (data coming into the component), activity (data going out of the component), and weights (parameters used to manipulate the data). To view the data, you stamp a probe onto a component and then use the “access” page of the inspector to select which set of data to view.



Learning Rate

The standard method of training a neural network is to compute the Gradient of performance surface and then move the weights such that we try to approach the bottom (minimum), thus giving us the smallest error. Since we do not know the distance to the minimum, only the direction, we must select a parameter which multiplies the gradient estimate. This parameter is called the learning rate or step size and is typically less than 1 (e.g. 0.1 or 0.01) to maintain a smooth approach to the minimum error.

Gradient

The gradient is a vector of partial derivatives of a function. It is like a multi-dimensional slope of a function. It always points in the direction of largest increase in the function. When using the gradient to find the minimum of a function (such as the error), you must move in the direction opposite the gradient.

Performance Surface

The performance surface is the value of the error (typically the mean squared error) of the neural network versus all of the weights of the neural network. The minimum point of the performance surface shows the optimal weights and the minimum error. Finding this point is the goal of training a neural network.

Function Approximation Breadboard

This neural network is called a [Multi-Layer Perceptron](#) (MLP). It is the most common [supervised](#) neural network. It consists of multiple layers of [PEs](#) connected in a feedforward fashion. The PEs in NeuroSolutions are the orange circular icons and are called [axons](#). The connections between the PEs are the icons with horizontal and diagonal lines between the axons and are called [synapses](#). NeuroSolutions uses the backpropagation of errors to train the MLP. The smaller icons on top of the axons and synapses are called [backpropagation components](#) and pass the error backwards from the end of the network to the beginning. The green axons on top of the backpropagation components are called [gradient search](#) components and adjust the weights contained in the synapses and axons – this is how the network is trained.

Networks constructed using the “low complexity” setting will have one [hidden layer](#). Those with medium or high complexity will have two hidden layers. The number of PEs in the first hidden layer is contained in the properties of the 2nd AXON from the left. The number of PEs in the second hidden layer is contained in the properties of the 3rd AXON from the left. The right most axon is called the criterion and reads the desired file from the attached [file component](#) and determines the error in the network. The axon 2nd from the right is the output axon and generates the actual network outputs. The axon on the far left is called the input axon and doesn’t do anything but accept the input from the file component.

Click on any object in the figure below to get a description of the component or probe.

NeuroSolutions - [BREADBOARD1.NSB]

File Edit Alignment Tools View Window Help

New

Open

Save

Start

Pause

Reset

Zero Count

NBuilder

NS Excel

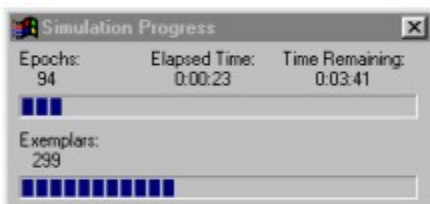
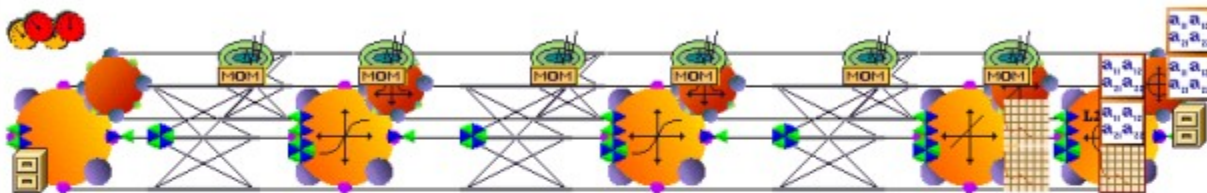
NEpert

Testing

Explain

Modify

Test

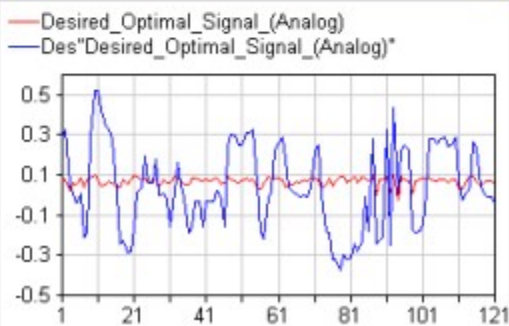


Average Cost of criterion

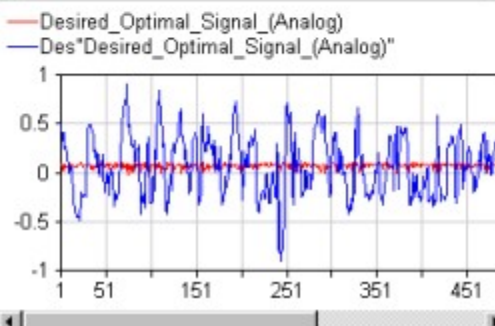
0

MSE 0.041400

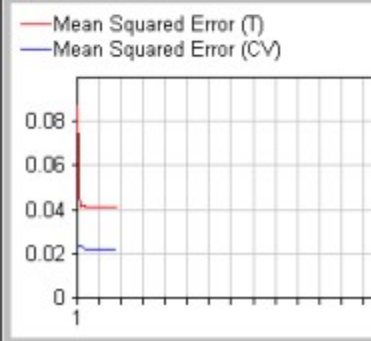
Cross Val Output vs. Desired Plot



Output vs. Desired Plot



Learning Curve




Explain Button

This button opens up this help file from NeuroSolutions.



Modify Button

This button loads the NeuralExpert wizard so that the breadboard can be modified. The current state of the breadboard is loaded into the NeuralExpert so you only have to make changes to the breadboard, not reload all of the data. Note, unlike the NeuralBuilder, the NeuralExpert will modify the existing breadboard, not create a new one. If you'd like to create a new breadboard with NeuralBuilder, relaunch it from the toolbar or Tools menu.



Modify

Test Button

This button loads the TestingWizard to easily add new data to your network for evaluation of its performance after training.



Prediction Breadboard

This neural network is called [Time-Lagged Feedforward Network](#) (TLFN). It is a [Multi-Layer Perceptron](#) (MLP) with memory components to store past values of the data in the network. The memory components allow the network to learn relationships over time. It is the most common temporal [supervised](#) neural network. It consists of multiple layers of [PEs](#) connected in a feedforward fashion. The PEs in NeuroSolutions are the orange circular icons and are called [axons](#). The connections between the PEs are the icons with horizontal and diagonal lines between the axons and are called [synapses](#). NeuroSolutions uses the backpropagation of errors to train the MLP. The smaller icons on top of the axons and synapses are called [backpropagation components](#) and pass the error backwards from the end of the network to the beginning. The green axons on top of the backpropagation components are called [gradient search](#) components and adjust the weights contained in the synapses and axons – this is how the network is trained.

Networks constructed using the “low complexity” setting will have one [hidden layer](#). Those with medium or high complexity will have two hidden layers. The number of PEs in the first hidden layer is contained in the properties of the 2nd AXON from the left. The number of PEs in the second hidden layer is contained in the properties of the 3rd AXON from the left. The right most axon is called the criterion and reads the desired file from the attached [file component](#) and determines the error in the network. The axon 2nd from the right is the output axon and generates the actual network outputs. The axon on the far left is called the input axon and doesn’t do anything but accept the input from the file component.

Click on any object in the figure below to get a description of the component or probe.

NeuroSolutions - [Breadboard2]

File Edit Alignment Tools View Window Help

New

Open

Save

Start

Pause

Reset

0
Zero Count

NBuilder

NS Excel

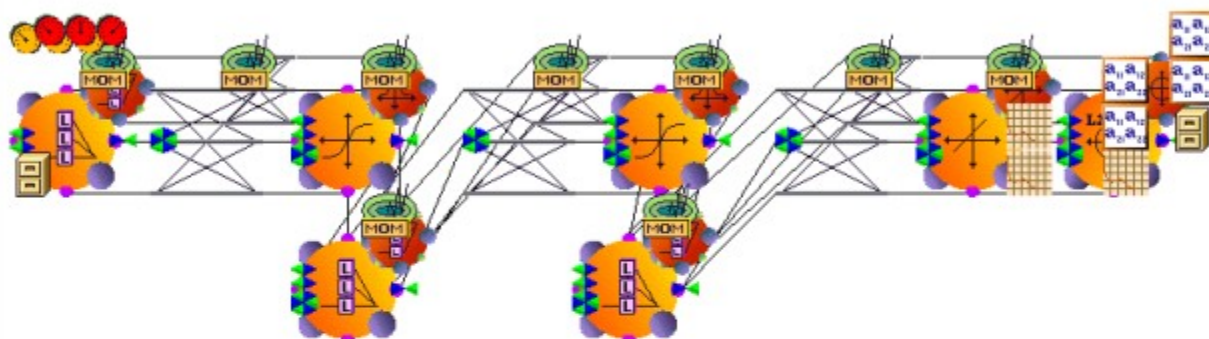
NEpert

Testing

Explain

Modify

Test



Output vs. Desired Plot

— Desired_Optimal_Signal_(Analog)



Learning Curve

— Mean Squared Error




Average Cost of criterion

MSE 0.000000


CV Average Cost of criterion

CV MSE 0.000000

Dynamic Controllers


These three controllers contain the global control parameters for the network. The image of the yellow dials  is the [dynamic controller](#). It contains parameters such as the number of epochs per run, the number of exemplars per epoch, the data sets to use, etc.

The image of the red dials  is the [dynamic backprop controller](#). It contains learning parameters like the learning mode (batch, online, etc.).

The image of the green dials  (if present) is the [genetic controller](#). It contains the parameters associated with the genetic optimization.

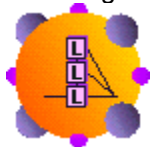


First Hidden Layer (Prediction)


These three components make up the first hidden layer of the neural network. The synapse  is the first component in the red box and makes the connection between each input and each processing element (PE) in the hidden layer. The synapse contains the connections and the trainable weights for each connection. The second component is the tanh axon

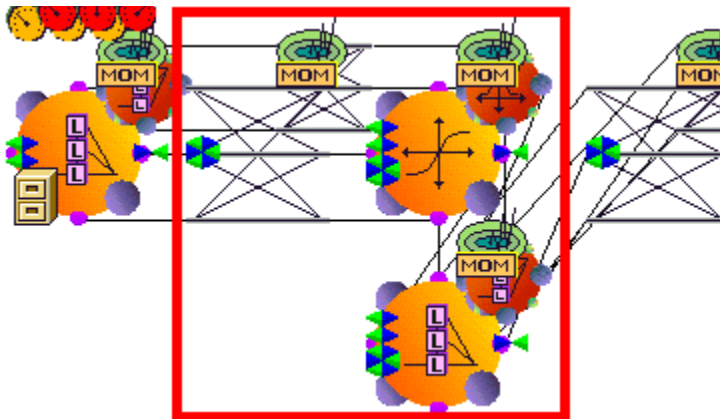


. This component has the processing elements for the hidden layer, each of which sums the weighted connections from the inputs. If you want to change the number of hidden layer PEs, open the inspector of this tanh axon and change the rows or columns. The number of weights in the synapse will be changed automatically. The third component is a memory component called the Laguerre Axon




. This component stores delayed versions of the PE output and passes it onto the next layer. This provides memory so the network can process information in time.

The green components on top of the axon and synapse are the momentum  gradient search components. They adjust the weights with information about the error in the network. To change the learning rates (a.k.a. step sizes), open the inspector on the momentum components.



Second Hidden Layer (Prediction)


These three components make up the second hidden layer of the neural network. The [synapse](#)  is the first component in the red box and makes the connection between each output of the first hidden layer and each [processing element](#) (PE) in the second hidden layer. The synapse contains the connections and the trainable [weights](#) for each connection. The second component is the [tanh axon](#)



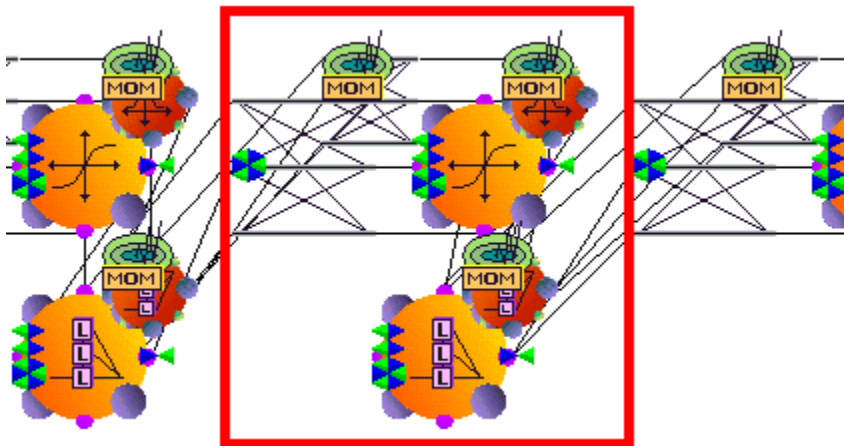
. This component has the processing elements for the hidden layer, each of which sums the weighted connections from the first hidden layer. If you want to change the number of hidden layer PEs, open the inspector of this tanh axon and change the rows or columns. The number of weights in the synapse will be changed automatically. The third component is a memory component called the [Laguerre Axon](#)




. This component stores delayed versions of the PE output and passes it onto the next layer. This provides memory so the network can process information in time.

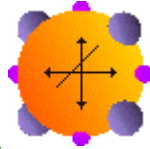
The green components on top of the axon and synapse are the [momentum](#)  [gradient search components](#). They adjust the weights with information about the error in the network. To change the learning rates (a.k.a. step sizes), open the inspector on the momentum components.

In many low and medium complexity networks, the second hidden layer will have only one PE in the second hidden layer, thus effectively creating a one-hidden layer neural network.




Output Layer (FA, Prediction)


These two components make up the output layer of the neural network. The [synapse](#)  is the first component in the red box and makes the connection between each second hidden layer PE and each output [processing element](#) (PE). The synapse contains the connections and the trainable [weights](#) for each connection.

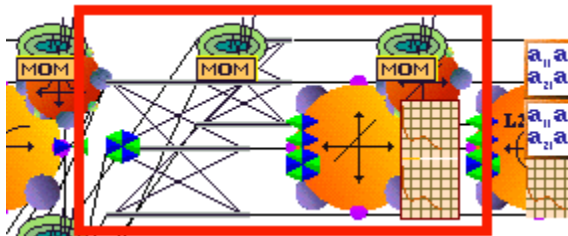


The second component is the [bias axon](#). This component has the processing elements for the output layer, each of which sums the weighted connections from the second hidden layer. For classification problems, the output is a tanh axon that saturates at ± 1 – this is ideal for classification.

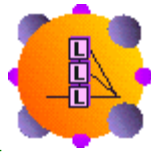
There is one output PE for each desired output specified in the desired file. The desired file is shown in

the file component on the right side of the [criterion](#) . You should not change the number of PEs in the output axon directly. It will be done automatically if you change the number of desired outputs.

The green components on top of the axon and synapse the [momentum](#)  [gradient search components](#). They adjust the weights with information about the error in the network. To change the learning rates (a.k.a. step sizes), open the inspector on the momentum components.



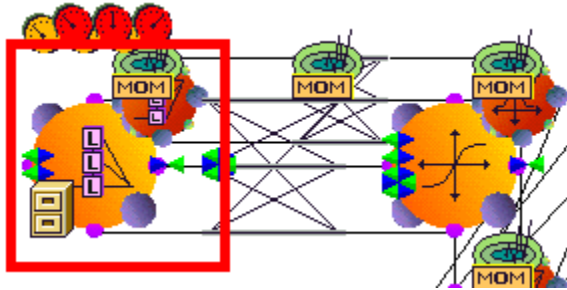
Input Axon (Prediction)



This component is a [Laguerre axon](#) that accepts the inputs to the network, delays and processes them, and passes them on to the network. This element gives the network memory into the past history of the data, allowing the network to extract relationships in time. The [file component](#)



on the bottom-left is used to read the data from the file system.

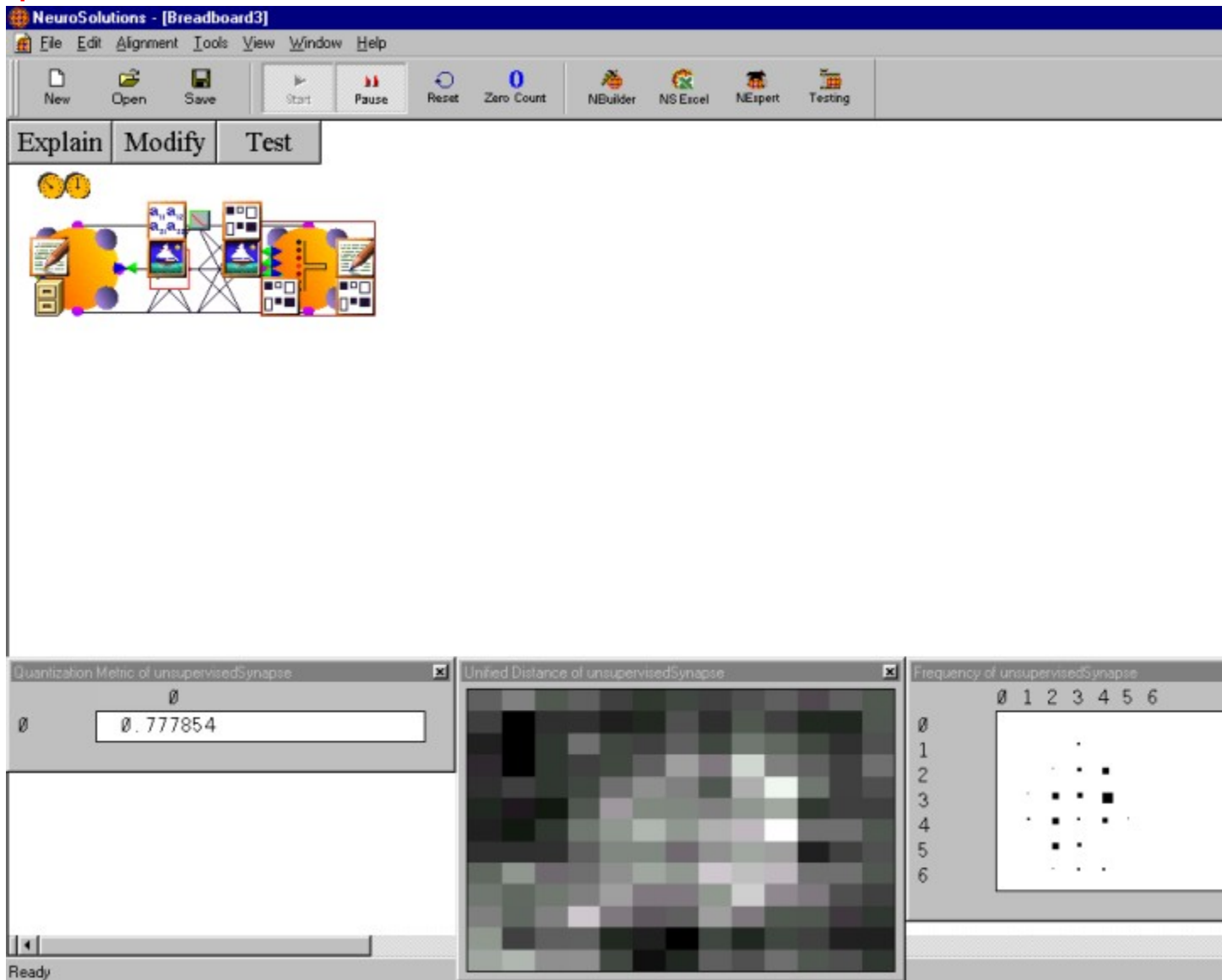


Clustering Breadboard

This neural network is called a [Kohonen](#) map or Self-Organizing (Feature) Map (SOM or SOFM). It is the most common [unsupervised](#) neural network. It consists of [PEs](#) connected in a feedforward fashion. The PEs in NeuroSolutions are the orange circular icons and are called [axons](#). The connections between the PEs are the icons with horizontal and diagonal lines between the axons and are called [synapses](#).

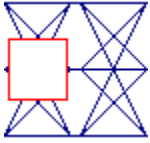
This network consists of an [input axon](#) on the far left, a [square kohonen synapse](#), and a [winner-take-all axon](#).

Click on any object in the figure below to get a description of the component or probe.



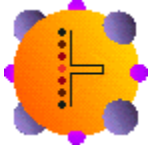
Square Kohonen Synapse

Since a Kohonen network is unsupervised (no backpropagation), the square kohonen component is responsible for both the computation of the results and the learning for the Kohonen map. Given a set of inputs and a set of PEs in the downstream axon, this synapse computes the distance from the weights of each PE (the cluster center) to the current input. If learning is active, the weights will then be updated according to the Kohonen learning rule.



Winner Take All Axon

The winner take all axon accepts a series of values from the upstream synapse and selects the one with the maximum (or minimum) value. In the Kohonen network, it selects the PE with the smallest distance to the input. This PE is the “winner”, meaning that the input will be grouped into that PEs cluster. The output of the winner take all axon is zeros for all PEs except the winning PE, which gets a 1.



Winning PE Probe Icon

Double-clicking this icon opens the [Winning PE Output probe](#).



It is a [matrix viewer](#) probe.

Winning PE Output

This window shows the sequential number of the winning PE (see the [winner take all axon](#)). The PEs are numbered starting from 0 and begin with (row 0, col 0) through (row 0, col N), then (row 1, col 0) through (row 1, col N)... This output is easier to deal with than the winner take all axon default output which is all zeros except for a one in the position of the winning PE.

PE Distances Icon

Double-clicking this icon opens the [PE Distances Window](#).



It is [Hinton](#) probe.

PE Distances Output

This shows the distances from the input to each of the PEs (cluster centers). By looking at this display, you can tell not only which PE is the winner, but also which other PEs the input was close to.

Winning PE Hinton Diagram Icon

Double-clicking this icon opens the [Winning PE Hinton Diagram](#).



It is [Hinton](#) probe.

Winning PE Hinton Diagram

This window shows the winning PE graphically, with a black square representing the position of the winning PE.

Unified Distance (umatrix) Icon

Double-clicking this icon opens the [Unified Distance Window](#).



It is an [image viewer](#) probe.

Unified Distance Window

When evaluating the clustering in a Kohonen network or SOM it is important to understand the mapping done. Typically, the number of SOM PEs is much larger than the number of clusters expected. This allows multiple PEs to capture one logical cluster. What you expect to see in the SOM map is groups of PEs representing a single cluster of the input. The question is how to determine where the clusters are in your SOM.

One way to determine the clustering in a Kohonen network or SOM is by looking at the distance between PE cluster centers. The weights from the input to each PE gives the PE cluster centers of the SOM. Inside a cluster of inputs, SOM PEs will be close to each other. Between SOM PEs the SOM map will have to stretch its PEs to map from one input cluster to the next. By finding large distances between neighboring PEs we should be able to find where inputs are clustered in the SOM. Large distances imply an input cluster boundary. Remember, in a square SOM, there are distances from one PE to each of its 8 neighbors.

The umatrix display shows the distance from each PE to its neighbors. Looking for large distances (light values on an image viewer, or large black squares on a hinton diagram) shows input cluster boundaries.

Histogram of Winning PEs Icon

Double-clicking this icon opens [Histogram of Winning PEs Window](#).



It is [Hinton](#) probe.

Histogram of Winning PEs Window

When evaluating the clustering in a Kohonen network or SOM it is important to understand the mapping done. Typically, the number of SOM PEs is much larger than the number of clusters expected. This allows multiple PEs to capture one logical cluster. What you expect to see in the SOM map is groups of PEs representing a single cluster of the input. The question is how to determine where the clusters are in your SOM. The histogram of win frequencies gives information to help determine the clustering.

You can think of the SOM as stretching its 2-D grid of PEs over the range of inputs (input space). It is helpful to imagine rubber bands between a PE and each of its neighbors. These PEs stretch to map the input space, but can only stretch so far. Inside a cluster, the PEs will be close together since all the inputs in that area are similar. Between PEs, however, often times you will have to stretch so far that a few PEs will get left in “empty” areas of the input space so that the map can stretch the appropriate distance. These PEs are called “dead PEs” since they do not win any competitions (since they are in an empty area of the input space). By finding these dead PEs, we can locate the borders of the clustering inside your SOM. By finding the PEs which win infrequently, we will find the dead PEs and these indicate cluster boundaries in your map.

Component Plane Icon

Double-clicking this icon opens the [Component Plane Window](#) .



It is an [image viewer](#) probe.

Component Plane Window

After training, the weights of the Kohonen or SOM synapse determine the PE cluster centers of the SOM. For example, the weights of the connections between the input vector and PE 1 represent the cluster center (or average value) of all the inputs that fire PE 1. By looking at the weights a different way, we can see how certain inputs affect the clustering. If we look at the weights from a single input (from the multi-dimensional input vector) to all the PEs, we see how that input varies from cluster to cluster. For example, if there are regions in the SOM map where the weights for a particular input are very high, then we can say that all the inputs clustered in that PE have a high value for that input. If all of the values for a particular input are approximately the same, then this particular input has no influence on the clustering.

As an example, after training a SOM, I created component plane images using NeuroSolutions and selected the component planes from the 5th and 6th inputs. The first one shows that the inputs clustered in the top center of the SOM have high values in the 5th column. The second figure shows that inputs clustered in the top right of the SOM have high values in the 6th column. In addition, there are some other locations where medium-high values in column 6 reside. After collecting all the component plane images, you can look for correlations among them. For example, if you see similar clustering in 4 different component planes, you can say that those 4 inputs are typically correlated – or that that area in the map requires all of those inputs to be in a certain range.

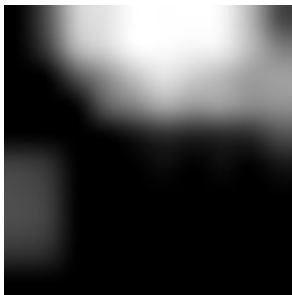


Figure 1 - Component Plane for Input #5

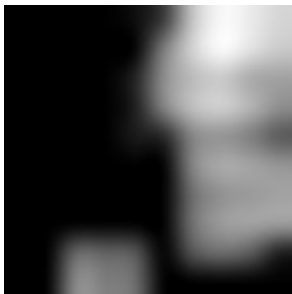


Figure 2 - Component Plane for Input #6

Quantization Metric Icon

Double-clicking this icon opens the [Quantization Metric Window](#).



It is a [matrix viewer](#) probe.

Quantization Metric Window

The average quantization error measures the “goodness” of fit of a clustering algorithm. It is the average distance between each input and the winning PE. If the quantization error is large, then the winning PE is not a good representation of the input. If it is small, then the input is very close to the winning PE. Remember, that by increasing the number of PEs you will almost always get lower quantization errors even though the clustering may logically not be much better. Also, changing the input will affect the best quantization error possible. The quantization error is best for comparing the clustering capabilities between multiple trainings of the same SOM on the same input. This display shows the average quantization error over an entire epoch of data.

{ewl RoboEx32.dll, WinHelp2000, }

