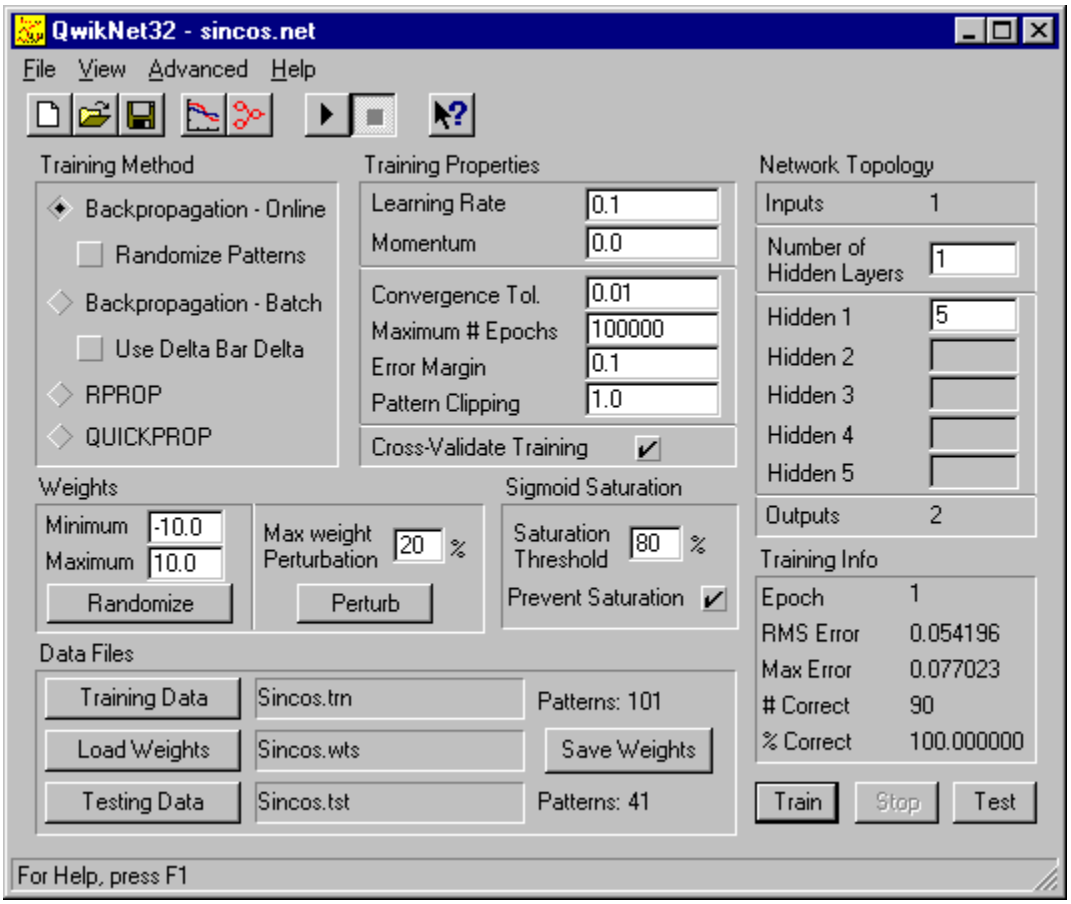
QwikNet Version 1.0 Artificial Neural Network for Windows

TABLE OF CONTENTS

Overview

Thanks for trying QWIKNET. QWIKNET is a powerful, yet easy-to-use Artificial Neural Network (ANN) simulator. QWIKNET implements several different methods to train a standard feed-forward neural network in an easy to use graphical environment.

Artificial Neural Networks can be used to learn the relationship between a set of inputs and outputs. ANNs have been used in many areas including image processing, financial forecasting, machine control, and optimization. All that is required to train an ANN is a set of training data that contains an input/output relationship.



File Menu

New Network
Clears existing network and resets all parameters to defaults.

Open Network
Loads a saved network file (*.net).

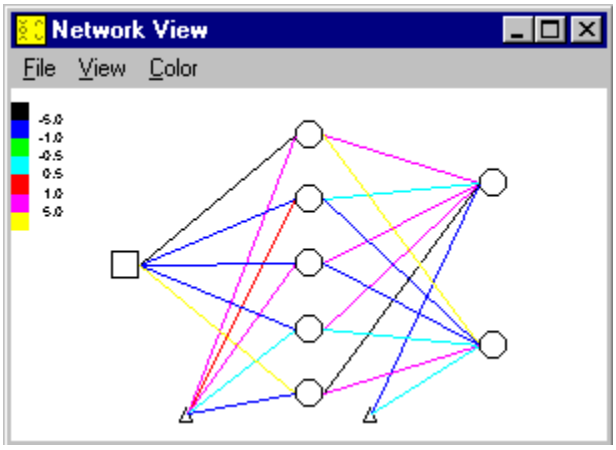
Save Network, Save Network as...
Saves the current network configuration including all network parameters and data file names. It does NOT save the weights. Weights must be saved via the *save weights* button. The default extension is *.net.

Exit
Exits program and prompts for any unsaved information.

View Menu

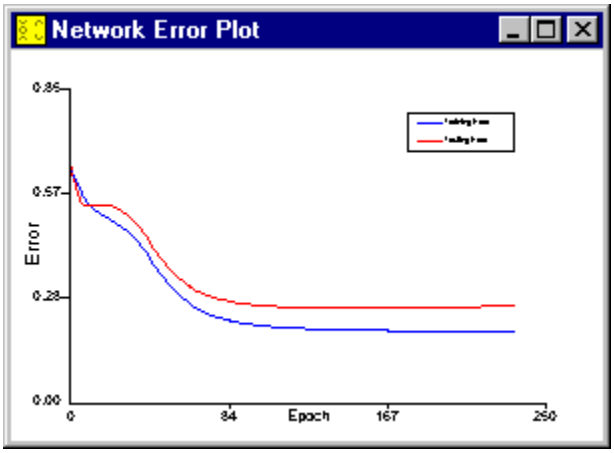
Network

Opens the network view window showing the currently loaded network. The connections can either represent the current weight values, or the learning rates for each weight. Select *view* on the menu to choose between *weights* or *learning rates*. The color scale values can be adjusted via the *color* entry on the menu. Squares represent input nodes, circles depict nodes with sigmoidal nonlinearities, the rightmost being the output layer. Triangles represent the bias for each node.



Error Plot

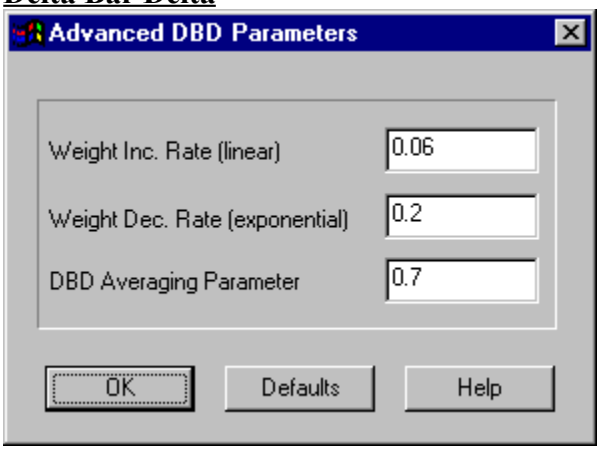
Opens the training error window. This displays a plot of the epoch versus RMS training error. If the Cross-Validate Training button is checked, a plot of the testing error is also displayed. This window is useful in determining if the network is learning properly, and when learning is completed.



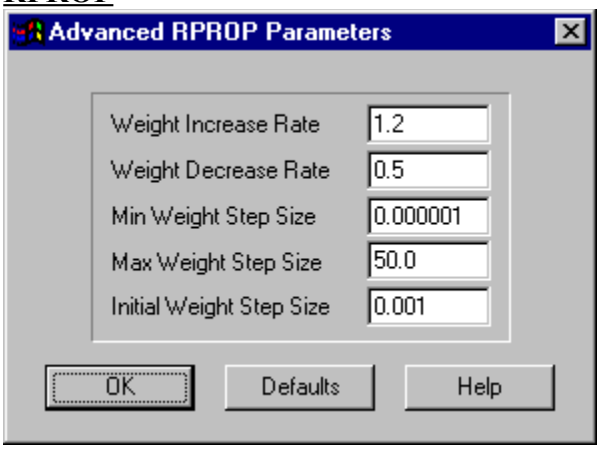
Advanced Menu

This menu allows the learning parameters for the advanced learning methods to be set. The default values should work for most cases, but certain problems may require fine-tuning of these parameters for best results.

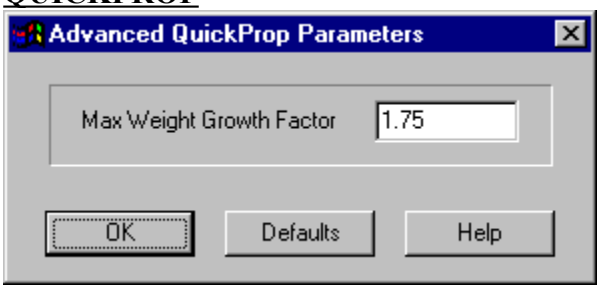
Delta Bar Delta



RPROP



QUICKPROP



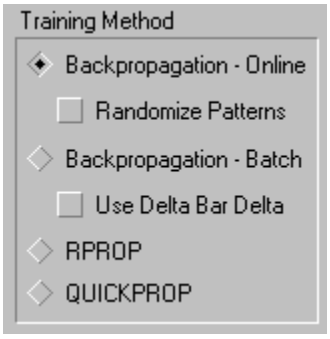
Help Menu

Help
Displays this file.

About
Displays the version number and other information about this program.

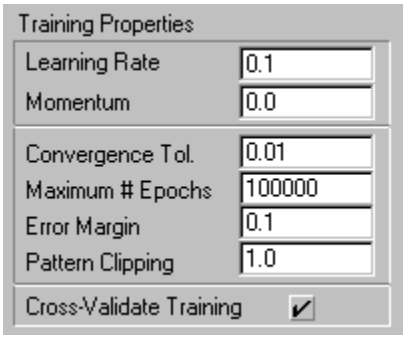
Training Method

Specifies the algorithm used to train the neural network. In some cases the choice of training method can have a substantial effect on the speed and accuracy of training. The best choice is dependent on the problem, and usually trial-and-error is needed to determine the best method.



Back-Propagation Online
Randomize Patterns
Back-Propagation Batch
Delta Bar Delta
RPROP
QUICKPROP

Training Properties



Learning Rate
The learning rate, h , controls the rate at which the network learns (see back-propagation). Usually the higher the learning rate, the faster the network learns. The valid range is between 0.0 and 100.0. A good initial guess is 0.1 when training a new network. If the learning rate is too high the network may become unstable, at which time the weights should be randomized and training restarted. The RPROP and Delta Bar Delta algorithms update this parameter adaptively.

Momentum
The momentum parameter, a , controls the influence of the last weight change on the current weight update (see back-propagation). The valid range is 0.0 to 1.0. Momentum usually results in faster learning, but can cause instability in some cases if set too large.

Convergence Tolerance
This parameter specifies the stopping criterion for the RMS training error. Training will stop when the RMS training error falls below this value.

Maximum # Epochs
Controls the maximum number of epochs to be used for training. One epoch is equivalent to one sweep of the training data set through the network.

Error Margin
Error margin is used by the network to classify the number and percent of training patterns the network has learned. Any pattern with an RMS training error less than this value is considered learned.

Pattern Clipping
Pattern clipping specifies the degree to which patterns that have been learned participate in future learning. A pattern is considered learned when its error is less than the value specified in the *Error Margin* field. The *Learning Rate*, h , is multiplied by this value for every pattern that has been learned. If this parameter is less than 1.0, the effect of previously learned patterns on future learning is reduced, thus emphasizing learning on patterns with large errors.

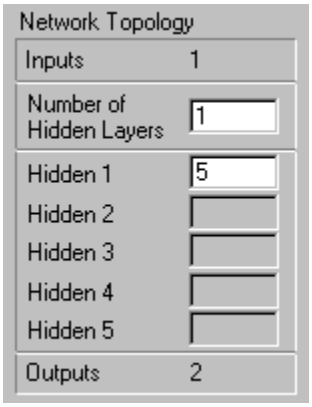
This option is useful when it is desired to reduce the maximum error in the training set. The

valid range is from 0.0 to 1.0. Setting this parameter to 1.0 removes the effect of this option, and setting it to 0.0 trains exclusively on patterns with errors above the Error Margin.

Cross-Validate Training

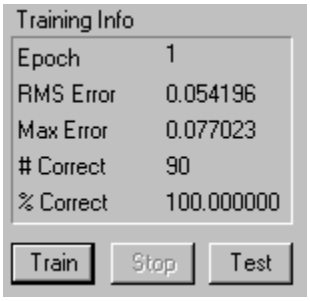
When this option is checked, the training data set is divided into two sets: 90% for training and 10% for testing. The testing set is not used to update the weights, therefore it can be used as an indication of whether or not memorization is taking place. When a neural network memorizes the training data, it produces acceptable results for the training data, but poor results when tested on unseen data. Memorization can be detected by watching for an increase in testing error compared to training error. This can be viewed via the error plot window.

Network Topology



This allows the number and size of the hidden layers to be specified. The network can contain up to 5 hidden layers, each with any number of neurons. The best network size is highly dependent on the problem at hand. A good starting point is one hidden layer with 4-6 neurons. This is sufficient for many problems. The input and output layer sizes can only be specified in the training data file.

Training Information



The training results are displayed here. When the Cross-Validate Training option is turned on, all errors are calculated only on the portion of the training data file that is used for actual training, (i.e. the first 90%).

The following quantities are displayed:

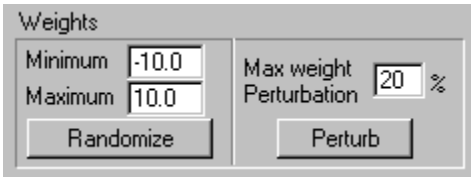
Epoch
The number of completed epochs. One epoch is defined as one presentation of the entire training data set to the network.

RMS Error
The current Root Mean Square error of the training data file. The network will stop training when the RMS error falls below the value set in the Convergence Tolerance field.

Max Error
Displays the maximum error in the training data file. The maximum error is determined by summing the magnitude of the training error for each output neuron over all the patterns, and finding the largest.

 # Correct, % Correct
Displays the number and percent of training patterns with absolute errors less than the value specified in the Error Margin field.

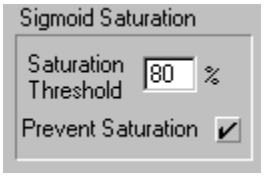
Weights



The *Randomize* button will randomly set all the weights in the range [-1,1]. The *Minimum* and *Maximum* fields allow the user to set the lower and upper bounds for each weight. During training, all weights are cropped at these thresholds. Bounding the weights can, in some cases, help prevent the network from becoming saturated.

The *Perturb* button will adjust each of the weights by a random percentage, not to exceed the value specified in the *Max weight perturbation* field. This option allows the user to *shake* the weights during training, and can cause the network to escape from a local minimum.

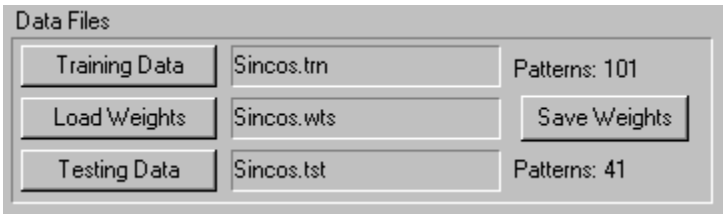
Sigmoid Saturation



A neuron is said to be saturated when a large change in input has little or no effect on the output. When a neuron becomes saturated its contribution to learning is negligible.

QWIKNET will monitor and correct for saturation when the *Prevent Saturation* box is checked. The S*aturation Threshold* field defines the minimum percentage of patterns per epoch that must saturate a neuron before it is considered saturated. A neuron is considered saturated when more than this percentage of patterns produce an output less than 0.01, or greater than 0.99. All weights that feed into a neuron that is saturated will be randomized in the range [-1,1].

Data Files



Training Data File
Weights Data File
Testing Data File

Training Data File

The training file contains the patterns used to train the network. Each row is made of X + Y values where the first X values correspond to the inputs, and the last Y values are the outputs. All values should be delimited by whitespace (space or tab). The training file must begin with a header describing the number of inputs and outputs. The header must be as follows:

[INPUTS] *number*
[OUTPUTS] *number*

A warning message is displayed if the training file does not contain the proper format. After the training data is loaded, the number of patterns read is displayed. The training data should be normalized for best results. The following is an example training file for the simple XOR problem.

```
[INPUTS] 2
[OUTPUTS] 1
0   0   0
0   1   1
1   0   1
1   1   0
```

Weights File

The network weights can be loaded from a previous simulation by pressing the *load weights* button. The currently loaded weights can be saved by pressing the *save weights* button.

The weights file contains the following header:

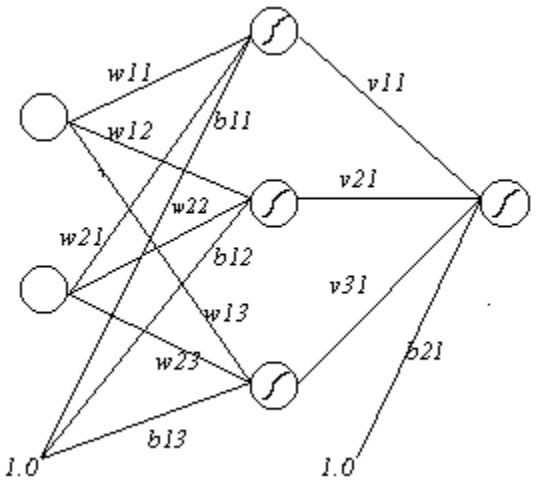
[layers] 3 total number of layers
[layersizes] 2 3 1 number of neurons in each layer (2 input, 3 hidden, and 1 output)

Next, the weight values are listed. The weights are arranged in rows. Each row is made up of connections from all neurons in the previous layer, to a neuron in the current layer. The last weight in each row is the bias. The following is a graphical explanation of the weight file structure for a simple 3 layer network with 2 inputs, 3 hidden neurons, and 1 output.

[layers] 3
[layersizes] 2 3 1

w11 w21 b11
w12 w22 b12
w13 w23 b13

v11 v21 v31 b21



Testing Data File

This file contains the patterns used to test the neural network. The testing data file contains only input data. The header is the same as the training data file without the outputs section. After QWIKNET reads the test file, the total number of patterns are displayed and the *Test* button is highlighted. When the *Test* button is pushed, each of the test patterns are forward propagated through the network and the outputs of the network are written to the specified output file. The header must be as follows:

[INPUTS] *number*

An example of a testing file for the XOR problem is as follows:

[INPUTS] 2
0 0
0 1
1 0
1 1

Back-Propagation

Back-propagation is the most commonly used training algorithm for neural networks. The weights are updated as follows

$$\Delta w_{ij}(t) = -\eta \frac{\partial E(t)}{\partial w_{ij}(t)} + \alpha \Delta w_{ij}(t-1)$$

where η is the learning rate, and
α is the momentum.

Back-Propagation Online
Online back-propagation updates the weights after each pattern is presented to the network.

Randomize Patterns
Online back-propagation with the order of the input patterns randomized prior to each epoch. This makes the learning process stochastic and is preferred in most cases.

Back-Propagation Batch
Back-propagation with weight updates occurring after each epoch.

Delta Bar Delta

An adaptive learning rate method in which every weight has its own learning rate. The learning rates are updated based on the sign of the gradient. If the gradient does not change signs on successive iterations then the step size is increased linearly. If the gradient changes signs, the learning rate is decreased exponentially. In some cases this method seems to learn much faster than non-adaptive methods. Learning rates $h(t)$ are updated as follows:

$$\Delta h(t) = \begin{cases} k & \text{if } \bar{d}(t-1)d(t) > 0 \\ -f h(t) & \text{if } \bar{d}(t-1)d(t) < 0 \\ 0 & \text{else} \end{cases}$$

where $d(t) = \frac{\partial E}{\partial w}$ at time t and $\bar{d}$ is the exponential average of past values of d.

$$\bar{d}(t) = (1-\theta)d(t) + \theta\bar{d}(t-1).$$

This method is implemented in conjunction with the batch back-propagation algorithm.

RPROP

RPROP stands for *resilient propagation*. This is an adaptive learning rate method where weight updates are based only on the sign of the local gradients, not their magnitudes. Each weight, w_{ij}, has its own step size or update value,
Δ_{ij} which varies with time
t according to:

$$\Delta_{ij}(t) = \begin{cases} \eta^{+} \times \Delta_{ij}(t-1), & \text{if } \frac{\partial E}{\partial w_{ij}}(t-1) \times \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ \eta^{-} \times \Delta_{ij}(t-1), & \text{if } \frac{\partial E}{\partial w_{ij}}(t-1) \times \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ \Delta_{ij}(t-1), & \text{else} \end{cases}$$

where $0 < \eta^{-} < 1 < \eta^{+}$. The weights are updated according to:

$$\Delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t), & \text{if } \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ +\Delta_{ij}(t), & \text{if } \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ 0 & \text{else} \end{cases}$$

QUICKPROP

QUICKPROP is a training method based on the following assumptions:

1) that $E(w)$ for each weight can be approximated by a parabola that opens upward, and;

2) that the change in slope of $E(w)$ for this weight is not affected by all other weights that change at the same time.

The weight update rule is:

$$\Delta w(t) = \frac{S(t)}{S(t-1) - S(t)} \Delta w(t-1) - h\,S(t)$$

where $S(t) = \frac{\partial E}{\partial w}(t)$. The numerator is the derivative of the error with respect to the weight and $(S(t-1) - S(t)) / \Delta w(t-1)$ is a finite difference approximation of the second derivative. Together these approximate Newtons method for minimizing a one-dimensional function $f(x)$: $\Delta x = -f'(x) / f''(x)$. To avoid taking an infinite backward step, or a backward uphill step, a *maximum growth factor* parameter m is introduced. No weight change is allowed to be larger than m times the previous weight change. QUICKPROP has a fixed learning rate parameter, h, that needs to be chosen to suit the problem.

Registration

You may evaluate this software free for 30 days, after which you must either purchase the registered version, or delete the software from your machine.

To register, send a check for $40 US, along with the completed registration form to:

Craig Jensen
9935 NE 125th Ln #4
Kirkland, WA 98034

Upon receipt of payment you will receive the full registered version and all future upgrades. A registration code will be sent via E-mail, making registration fast and easy.

The shareware version is limited to 1 hidden layer with a maximum of 10 hidden neurons. Entering the registration key removes all restrictions on layer sizes and allows for up to 5 hidden layers.

System Requirements

QWIKNET is available in both 16 bit and 32 bit versions. If you have Windows 95 or NT use the 32 bit version. The 32 bit version is a multi-threaded application and provides much higher performance.

The 16-bit version, QWKNET16.EXE, requires at least a 286 processor and Windows 3.1. Due to the segmented memory of 16-bit Windows, QWKNET16.EXE is limited to training/testing data files of at most 10,000 patterns, with at most 500 neurons in each layer.

The 32-bit version, QWKNET32.EXE, requires a minimum 486 processor and Windows 95 or NT. This version removes all practical limits on network/data file size.

Installation of both versions, including the DLLs and help files, requires less than 1.5 MB of disk space.

Example Data Files

Several example data files are included for testing:

SINCOS.TRN	-	Training data file with one input (x) and two outputs Sin(x) and cos(x).
SINCOS.TST	-	Testing data for SINCOS.
SINCOS.WTS	-	Example of saved weights.
SINCOS.NET	-	SINCOS network file.
XOR.*		Exclusive OR training data. Two inputs and one output.
SECURITY.*	-	A more complex data set for Power System Security analysis. The data set contains 33 inputs (power system features) and 1 output (security classification).

Installation

16 bit version - copy the following files to your directory:

QWKNET16.EXE	- 16-bit executable
BWCC.DLL	- 16-bit DLL needed by QWKNET16
QWIKNET.HLP	- Windows help file
README.TXT	- This file
REGISTER.TXT	- A registration form in ASCII format
FILE_ID.DIZ	- Information file for bulletin boards
Sample data files	(optional)

32 bit version - copy the following files to your directory:

QWKNET32.EXE	- 32 bit executable for Win95 or NT
BWCC32.DLL	- 32-bit DLL needed by QWKNET32
QWKNET.HLP	- Windows help file
README.TXT	- This file
REGISTER.TXT	- A registration form in ASCII format
FILE_ID.DIZ	- Information file for bulletin boards
Sample data files	(optional)

Questions/Comments

Please send any questions, comments or bug reports to:

Craig Jensen
9935 NE 125th Ln #4
Kirkland, WA 98034
email - cjensen@ee.washington.edu

Registration Form

Craig Jensen
9935 NE 125th Ln #4
Kirkland, WA 98034
USA

REGISTRATION FORM FOR QWIKNET
==========

Name:.....

Company Name:.....

Address: .....

.....

.....

Phone Number: .....

e-mail Address: .....

(Important! Reg. Code is sent by E-mail)

No. of copies: .....

Additional comments, suggestions, etc. .

.....

.....

.....

Registration fee is $40 US/copy. Send this order form along with your payment to:

Craig Jensen
9935 NE 125th Ln #4
Kirkland, WA 98034, USA.

If you have any questions, comments or suggestions please contact Craig Jensen at the above address or via e-mail: cjensen@ee.washington.edu.

As this software is shareware it comes *as is*, there is no warranty implied or otherwise, nor is support provided. However, if you discover any bugs or problems please contact the developer at the above e-mail address.