G2 Neural Network Engine

User's Guide

Version 5.1 Rev. 0

G2 Neural Network Engine User's Guide, Version 5.1 Rev. 0
June 2016

Gensym Corporation
52 Second Avenue
Burlington, MA 01803 USA
Telephone: (781) 265-7100
Fax: (781) 265-7101

Part Number: DOC131-510

Contents

Preface

Describes this guide and the conventions that it uses.

About this Guide

G2 Neural Network Engine (GNNE) is a set of plugin modules for developing and running intelligent neural network applications. It provides the functionality to manage the data flow of neural network applications and the interaction with other applications. Its principal component is a set of neural network models and data storage objects that lets you build your data flow procedures through a set of API calls.

This guide describes general information about how to use GNNE objects. It provides a reference for each object and its API. It assumes you are familiar with G2. In addition, the Educational Services Department at Gensym holds classes that are excellent ways to become familiar with these products.

This guide consists of these chapters:

This chapter...	Describes...
Introduction to GNNE	The features and benefits of the GNNE module.
Integration of GNNE and NOL Studio	How to integrate GNNE and NOL Studio.
Object Reference	The GNNE objects.
Application Programmer's Interface	GNNE's application programmer's interface (API).

Audience

This guide is for application developers and system integrators to develop end-user applications. Users should be familiar with the NeurOn-Line Studio environment. Users should also be familiar with G2.

Conventions

This guide uses the following typographic conventions and conventions for defining system procedures.

Typographic

Convention Examples	Description
g2-window, g2-window-1, ws-top-level, sys-mod	User-defined and system-defined G2 class names, instance names, workspace names, and module names
history-keeping-spec, temperature	User-defined and system-defined G2 attribute names
true, 1.234, ok, "Burlington, MA"	G2 attribute values and values specified or viewed through dialogs

Convention Examples	Description
Main Menu > Start KB Workspace > New Object create subworkspace Start Procedure	G2 menu choices and button labels
conclude that the x of y ...	Text of G2 procedures, methods, functions, formulas, and expressions
new-argument	User-specified values in syntax descriptions
text-string	Return values of G2 procedures and methods in syntax descriptions
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save Properties	GMS and native menu choices
workspace	Glossary terms
`c:\Program Files\Gensym\`	Windows pathnames
`/usr/gensym/g2/kbs`	UNIX pathnames
`spreadsh.kb`	File names
`g2 -kb top.kb`	Operating system commands

`public void main()`
`gsi_start`
	Java, C and all other external code

Note Syntax conventions are fully described in the *G2 Reference Manual*.

Procedure Signatures

A procedure signature is a complete syntactic summary of a procedure or method. A procedure signature shows values supplied by the user in *italics*, and the value (if any) returned by the procedure <u>*underlined*</u>. Each value is followed by its type:

```
g2-clone-and-transfer-objects
    (list: class item-list, to-workspace: class kb-workspace,
     delta-x: integer, delta-y: integer)
    ->  transferred-items: g2-list
```

Related Documentation

NeurOn-Line

NeurOn-Line Release Notes

NeurOn-Line User's Guide

NeurOn-Line Reference Manual

NeurOn-Line Studio User's Guide

Gensym Neural Network Engine

G2 Core Technology

- *G2 Bundle Release Notes*
- *Getting Started with G2 Tutorials*
- *G2 Reference Manual*
- *G2 Language Reference Card*
- *G2 Developer's Guide*
- *G2 System Procedures Reference Manual*
- *G2 System Procedures Reference Card*
- *G2 Class Reference Manual*
- *Telewindows User's Guide*
- *G2 Gateway Bridge Developer's Guide*

G2 Utilities

- *G2 ProTools User's Guide*
- *G2 Foundation Resources User's Guide*
- *G2 Menu System User's Guide*
- *G2 XL Spreadsheet User's Guide*
- *G2 Dynamic Displays User's Guide*
- *G2 Developer's Interface User's Guide*
- *G2 OnLine Documentation Developer's Guide*
- *G2 OnLine Documentation User's Guide*
- *G2 GUIDE User's Guide*
- *G2 GUIDE/UIL Procedures Reference Manual*

G2 Developers' Utilities

- *Business Process Management System Users' Guide*
- *Business Rules Management System User's Guide*
- *G2 Reporting Engine User's Guide*
- *G2 Web User's Guide*
- *G2 Event and Data Processing User's Guide*
- *G2 Run-Time Library User's Guide*
- *G2 Event Manager User's Guide*
- *G2 Dialog Utility User's Guide*
- *G2 Data Source Manager User's Guide*
- *G2 Data Point Manager User's Guide*
- *G2 Engineering Unit Conversion User's Guide*
- *G2 Error Handling Foundation User's Guide*
- *G2 Relation Browser User's Guide*

Bridges and External Systems

- *G2 ActiveXLink User's Guide*
- *G2 CORBALink User's Guide*
- *G2 Database Bridge User's Guide*
- *G2-ODBC Bridge Release Notes*

- *G2-Oracle Bridge Release Notes*
- *G2-Sybase Bridge Release Notes*
- *G2 JMail Bridge User's Guide*
- *G2 Java Socket Manager User's Guide*
- *G2 JMSLink User's Guide*
- *G2 OPCLink User's Guide*
- *G2-PI Bridge User's Guide*
- *G2-SNMP Bridge User's Guide*
- *G2-HLA Bridge User's Guide*
- *G2 WebLink User's Guide*

G2 JavaLink

- *G2 JavaLink User's Guide*
- *G2 DownloadInterfaces User's Guide*
- *G2 Bean Builder User's Guide*

G2 Diagnostic Assistant

- *GDA User's Guide*
- *GDA Reference Manual*
- *GDA API Reference*

Customer Support Services

You can obtain help with this or any Gensym product from Gensym Customer Support. Help is available online, by telephone, by fax, and by email.

To obtain customer support online:

➔ Access G2 HelpLink at `www.gensym-support.com`.

You will be asked to log in to an existing account or create a new account if necessary. G2 HelpLink allows you to:

- Register your question with Customer Support by creating an Issue.
- Query, link to, and review existing issues.
- Share issues with other users in your group.
- Query for Bugs, Suggestions, and Resolutions.

To obtain customer support by telephone, fax, or email:

➔ Use the following numbers and addresses:

	Americas	Europe, Middle-East, Africa (EMEA)
Phone	(781) 265-7301	+31-71-5682622
Fax	(781) 265-7255	+31-71-5682621
Email	`service@gensym.com`	`service-ema@gensym.com`

Introduction to GNNE

Provides an overview of the features and benefits of the GNNE module.

Introduction

G2 Neural Network Engine (GNNE) is an API-based plugin module, which works in conjunction with other Gensym products for developing applications that monitor and control real-time processes. It is the preferred deployment option for G2 Neural Network products. It is composed of a set of core objects that you can clone from GNNE palettes, and a set of API procedures to express a flow of data and perform neural network calculations.

An application that uses GNNE generally performs these steps:

- **Acquires data** from real-time processes.
- **Makes inferences** based on the data.
- **Takes actions** based on the inference values, such as raising alarms, sending messages to operators, or concluding new setpoints.

The GNNE palettes are divided into these categories:

- **Data Objects**, which store numeric data.
- **Data Controls**, which include data filters.
- **Neural Network Objects**, which store network parameters.
- **Statistical Model Objects**, which store statistical model parameters.

You typically clone model objects and data set objects from GNNE palettes and place them on your working folder. You can then configure the attributes of the objects through API calls to control their behavior.

A typical GNNE workspace consists of GNNE objects and user-defined procedures, which process data, analyze the results, and pass the results to other parts of your application for monitoring and analysis. This set of objects and procedures does the following:

- **Acquires data**, filters the data, and performs operations on the data, such as arithmetic or SPC operations. This part of the application uses data items and associated APIs.
- **Makes inferences** about the data and performs logic operations on the inference values. This part of the application uses neural network items and their APIs.
- **Performs actions** based on the inference values. This part of the application uses user-defined procedures.

To facilitate the development of neural network applications, the G2 NeurOn-Line bundle integrates the GNNE deployment environment with the offline development tool, NOL Studio. When NOL Studio and G2 are connected, model weights or parameters of any model developed in NOL Studio can be passed to model objects in the online GNNE environment through a procedure call. The data collected in the G2 application can be passed to NOL Studio for visualization and preprocessing, and for building and validating models.

GNNE Features

GNNE provides these basic categories of functions:

- Data processing
- Model execution
- Neural network model retraining
- Neural network model validation
- File operation

- Remote process management
- Online interaction with NOL Studio

Data Processing

The GNNE data processing functionality provides:

- Data storage in the forms of scaler, vector data, and data pair.
- Data filtering such as scaling, novelty filter, maximum age filter, and limit-size filter.

Model Execution

The GNNE APIs for neural network model execution provide ways to calculate output based on new input data for these types of models:

- Backpropagation network model
- Radial basis function network model
- Autoassociative network model
- Rho net model
- Ensemble model
- Predictive Mode
- Optimization Model
- Partial Least Square Model
- Principal Component Analysis Model

Neural Network Model Retraining

Normally, neural network models are trained in NOL Studio environment and deployed in GNNE. However, GNNE provides the ability to retrain the model based on new data gathered in G2. The GNNE APIs for neural network model training provide training for these types of models:

- Backpropagation network model on a particular data set, using a training method that you specify
- Radial basis function network model
- Autoassociative network model
- Rho net model

- Ensemble model
- Predictive model

Neural Network Model Validation

The GNNE APIs for neural network validation provide:

- Fit test that applies a neural network to see how well it fits a data set, using a fitting criteria you specify. It also fills the prediction matrix of the data set with predicted values corresponding to each validation case.
- Sensitivity tests to find the best input-output structure of the model.
- Validation tests of neural network models in NOL Studio with data passed from GNNE.

File Operations

GNNE normally manages large data sets through text files. You can load training and validation data from data files with predefined format. For neural networks, GNNE lets you save and load what they have learned so far. GNNE saves the information to text files, which you can examine yourself.

The GNNE APIs for file operations provide:

- Loading and saving data in different formats, such as array, matrix, or a complete data set.
- Loading and saving neural network parameters.
- Loading and saving statistical model parameters.

Remote Process Management

GNNE uses **remote procedure calls (RPCs)** for numerically intensive tasks and data exchange. GNNE uses three types of remote processes:

- A GNNE remote process performs neural network training, fit testing, sensitivity testing, and running autoassociative networks.
- A JavaLink process performs predictive and optimization model calculation.
- A nols-gateway bridge communicates with NOL Studio.

The type of RPCs is based on the type of models it handles. The RPCs used to handle backpropagation network models, radial basis function network models, autoassociative network models, rho net models, and ensemble models are defined as classic RPCs. The RPCs used to handle predictive and optimization models are defined as NeurOn-Line JavaLink RPCs. The RPCs used to interact with NOL Studio for data exchange and parameter passing are defined as NOL Studio RPCs. The remote processes can coexist in G2.

The remote procedures run as a concurrent process separate from G2. G2 communicates with the remote process by using a bridge, based on TCP/IP. It also uses text files for data exchange.

For GNNE to perform these tasks correctly, you must:

- Launch the remote process.
- Establish the communication connection, using the proper protocol.
- Supply valid file names for data passing.

GNNE usually handles the RPC operation transparently. You can also handle the startup of the remote process manually, either through an API call in G2 or externally. The classic RPC is described in "Starting the Remote Process in an External Window" in Chapter 6 "Managing NOL's Remote Process" in the *NeurOn-Line User's Guide*. The NOL JavaLink RPC is described in "Launching a Remote Process Using Procedure" in Chapter 13 "Model deployment" in the *NeurOn-Line Studio User's Guide*. The NOL Studio RPC is described in next chapter.

Online Interaction with NOL Studio

NOL Studio is recommended as the modeling tool for developing neural network and statistical models. An instance of NOL Studio can be launched from a G2 server and individual Telewindows session. Each Telewindows session owns its own NOL Studio instance. The NOL Studio console closes when its owner Telewindows session is closed. If an independent NOL Studio already exists and is connected to G2 from its console, the owner will be the G2 server. Once a NOL Studio console is connected to G2, all functionality provided by the NOL JavaLink exists; it is not necessary to launch a new NOL JavaLink remote process for neural network calculation. Before a NOL Studio interface is launched, if a remote process is already established, the internal process decides which process to use for the remote procedure calls.

Module Integration

GNNE is a plugin module that can be integrated with other G2 applications, especially these Optegrity bundle modules:

This module...	Performs these tasks...
Optegrity	Provides intelligent object classes for building process maps that integrate with GEDP, SymCure, and GEVM.
SymCure Also known as CDG (Causal Directed Graphs)	Defines generic diagnostic models for intelligent object classes, which reason about events to determine root causes and to take corrective actions.
G2 Neural Network Engine (GNNE)	Provides intelligent prediction for reasoning and basis for future actions. Provides integration functions for integrating NOL Studio with GNNE.
NOL Studio	Provides deployment functions of NOL Studio models and basic functions for integrating NOL Studio with GNNE.

GNNE integrates with these modules, as follows:

- Optegrity intelligent objects:
  - Listen for property value changes from intelligent objects and set property values of those objects.
  - Listen for any type of event that occurs on an intelligent object, including raw events, operator messages, and SymCure events.
  - Associate generic GEDP diagrams with intelligent object classes. Specific GEDP diagrams generate events for specific instances of those intelligent object classes.
- SymCure
  - Generates specific SymCure events for intelligent object instances or any object instance.
  - Detects SymCure events for intelligent domain objects or any object instance.

GNNE Objects

GNNE provides these categories of objects:

- **Data objects**, which store numeric data.
- **Data controls**, which include data filters.
- **Neural network objects**, which store network parameters.
- **NOL Studio objects**, which store predictive model, optimization and statistical model parameters.

You typically place neural network, data control objects and data set objects on a working folder that is not visible, such as the subworkspace of an intelligent object, to represent the data flow of the object. It is also possible to build a class-level data flow scheme, using GNNE objects and to associate those objects with an intelligent object class.

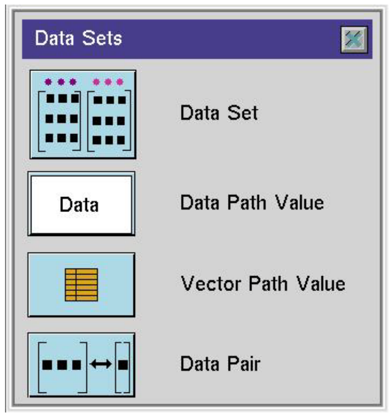
GNNE provides objects on palettes. To create a data flow scheme, you choose appropriate objects from the palettes, place them on a workspace, and create procedure to manage the data flow and handle abnormal situations.

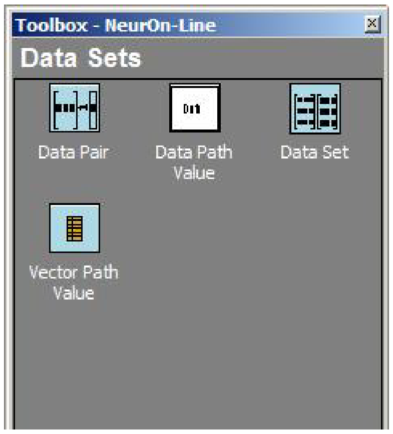
Data Objects

GNNE data objects are used for storing and manipulating static or dynamic data for your application:

- **Data Pair** is a data class by combining two vectors: one vector becomes the data pair's input vector, the other becomes the data pair's output vector.
- **Data Set** stores data pairs for training and testing neural networks.
- **Data Path Value** stores a scaler value with given timestamp.
- **Vector Path Value** stores a vector.

Here are the Data Sets palettes in the G2 server and Telewindows:





Data Controls

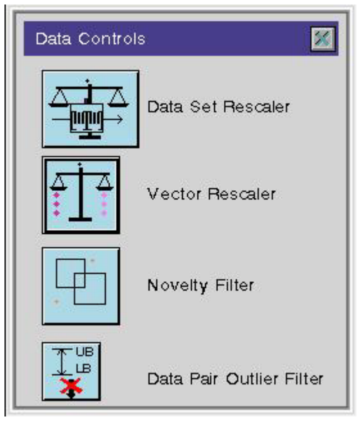
GNNE data controls are used to manipulate data for your application:

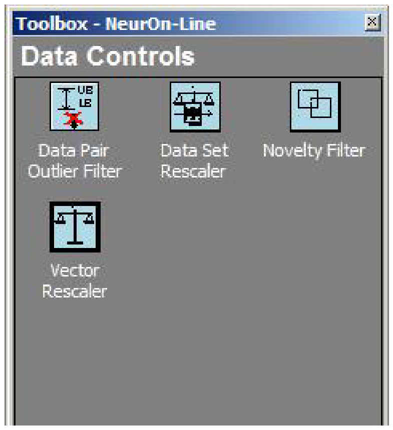
- **Data Set Rescaler** scales the input and target data in a Data Set. You can specify your own scaling factors or let the Data Set Rescaler create them using one of two standard scaling methods: Min-Max scaling or Mean-Standard Deviation scaling. It can also create two Vector Scaling blocks that scale vectors in the same way that the Data Set Rescaler scales a Data Set's input and target data. If your data has wide variations, you may need to rescale it to train the network best. However, after you train the network with scaled data, you will get invalid results if you apply that network to raw data. Also the network's output data is scaled and is different from the raw target data. To solve these problems, the Data Set Rescaler not only scales the training data, it also creates two Vector Rescalers that undo the scaling so you can apply the network to raw input data and interpret the network's output.
- **Vector Rescaler** rescales the elements of the input vector by applying additive and multiplicative factors to each element. The block has different factors for each element. Each element Input_i is rescaled according to this formula, where A_i is the additive factor for that element and M_i is the multiplicative factor for that element:

$$\text{Output}_i = M_i(\text{Input}_i + A_i)$$

- **Novelty Filter** is a filter that prevents a Data Set from being filled with redundant data. Whenever you add a data pair to the attached Data Set, the Novelty Filter checks whether there are more than a specified number of data pairs within a specified distance of the new data pair. If there are, the Novelty Filter removes the older data pair.
- **Data Pair Outlier Filter** separates any data pairs whose elements do not fall within specified bounds. The data pairs whose elements fall within the bounds are passed as the filter output.

Here are the Data Controls palettes in the G2 server and Telewindows:



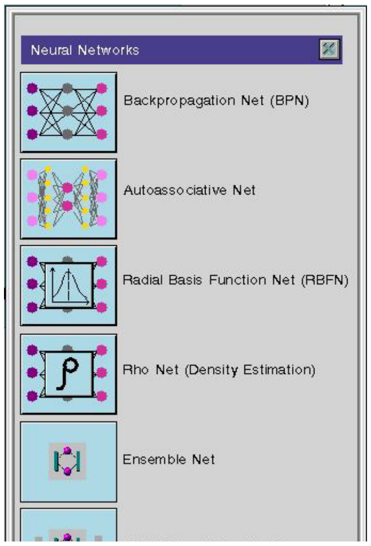


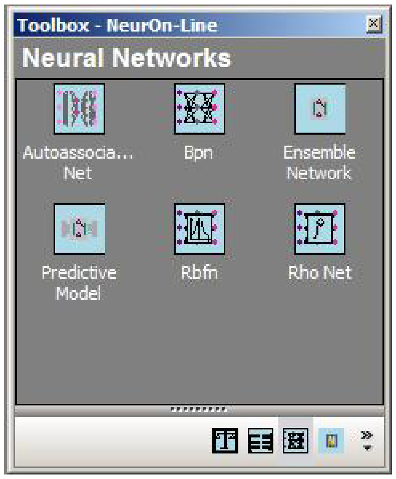
Neural Network Objects

The neural network objects are the key part of GNNE:

- **Back Propagation Net** is a feed-forward network with multiple layers.
- **Autoassociative Net** is a feed-forward network with multiple layers, with a specific architecture that is especially good for handling certain types of problems, including sensor validation.
- **Radial Basis Function Net** is a 3- layer, feed-forward networks, whose middle layers use a multivariate Gaussian function.
- **Rho Net** is a type of Radial Basis Function Net. The difference is the network output, which is a number between 0.0 and 1.0 that represents the probability that the input value is in a particular class of feed-forward network with multiple layers.
- **Ensemble Network** is a set of Backpropagation Nets, which have been trained in NOL Studio. It has a specific architecture, which gives it accuracy and robustness.
- **Predictive Model** is a model that is based on the Ensemble Network. In addition to the neural network, the Predictive Model manages variable information, such as name and delays. The Predictive Model is trained in NOL Studio and can be retrained within GNNE.

Here are the Neural Networks palettes in the G2 server and Telewindows:


Neural Networks
Backpropagation Net (BPN)
Autoassociative Net
Radial Basis Function Net (RBFN)
Rho Net (Density Estimation)
Ensemble Net

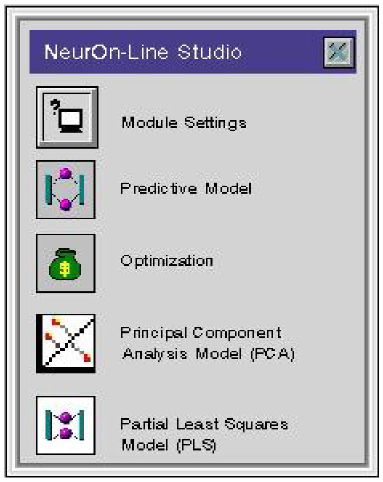


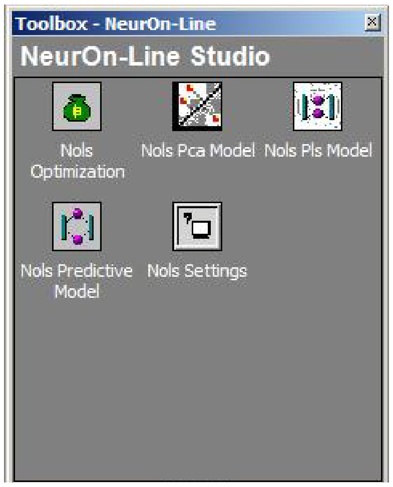
NOL Studio Objects

NOL Studio objects include NOL Studio models running through NOL JavaLink RPCs and statistical models:

- **Predictive Model** is a NOL Studio object for ensemble networks. This object contains the preprocessor defined in NOL Studio. The preprocessor can use formulas to treat the input data before feeding them into neural network models.
- **Optimization** contains the specification of an optimization problem.
- **Partial Least Square (PLS)** is a multivariate linear regression model.
- **Principal Component Analysis (PCA)** is a statistical model used to reduce the dimensionality of a data set while retaining as much information as is possible.

Here are the Neural Networks palettes in the G2 server and Telewindows:


NeurOn-Line Studio
Module Settings
Predictive Model
Optimization
Principal Component Analysis Model (PCA)
Partial Least Squares Model (PLS)



Accessing the GNNE API

You normally control GNNE objects from within a G2 procedure or function with GNNE's application programmer's interface (API). The API includes procedures that perform all available actions for every GNNE object.

Many of the API procedures require the following arguments:

- The object to execute.
- The input value or values required by the object.
- The output value or values produced by the object.

For more information about GNNE API, see Application Programmer's Interface

Integration of GNNE and NOL Studio

Describes how to integrate GNNE and NOL Studio.

Introduction

G2 Neural Network Engine (GNNE) and G2 NeurOn-Line (NOL) Studio are closely integrated to form an online development and deployment environment for neural network applications. NOL Studio is the preferred tool for data treatment, model building, and model validation. GNNE is the preferred deployment option for G2 Neural Network products. An integrated environment facilitates neural network application development. Once NOL Studio and GNNE are connected online, model weights or parameters of any model developed in NOL Studio can be passed on to the model object in GNNE environment through procedure calls. The data collected in the G2 application can be passed into NOL Studio to be visualized and preprocessed and used to build and validate models.

With the online integration of GNNE and NOL Studio, users can perform following interactions from the GNNE side or the NOL Studio side:

- **Launch NOL Studio** from G2.
- **Connect GNNE** from NOL Studio.

- **Acquire data** from real-time processes. G2 provides the capability of reading data from virtually any source through its family of bridge products. You can send a data set collected in GNNE into a connected NOL Studio console as training or validation data.
- **Export model parameters** into the model object in GNNE.
- **Validate model** with new data collected in G2.

Integrated Module Hierarchy

The G2 side of GNNE consists of several G2 modules. The top-level module is gnne, which stands for G2 Neural Network Engine. It requires the nolstudio deployment module. To use the integrated environment for neural network development, your application must require the gnne top-level module.

For more information about creating, populating, and saving a module, see the chapter on "Modules and Modularized KBs" in the *G2 Reference Manual*.

In addition to the core functionality of data processing and neural network calculation, the gnne module provides:

- Standard palette behavior for GNNE and NOL Studio objects.
- Procedures for exchanging data between GNNE and NOL Studio.
- Management of the remote processes.

Connecting NOL Studio and GNNE

To provide an integrated environment for neural network development, you must connect GNNE with NOL Studio. There are two ways to connect NOL Studio and GNNE:

- Launch a NOL Studio console from G2. If a NOL Studio console is launched from G2, it is automatically connected to this G2 process. An individual G2 window, which includes G2 server and Telewindows, can own its own NOL Studio instance. Each window can only own one NOL Studio instance. The NOL Studio console is closed if its owner Telewindows is closed.
- Connect G2 from an independent existing NOL Studio console. If you connect G2 from the NOL Studio, this NOL Studio instance is owned by the G2 server. This means that only one independent existing NOL Studio can connect to a G2 process.

Launching NOL Studio from G2

You can launch a NOL Studio console using a procedure. You provide the home directory, the listener port, and the name of the interface in a module setting object called nols-setting. To facilitate this process, the NolG2Gateway class is available in the *nolstudio\com\gensym\nols\deploy* directory.

To set up nols-settings object:

1 Start G2.

2 Clone a nols-settings object from the NOLStudio palette, and place it on a workspace in your top-level module, *not* *nolstudio.kb*.

3 Edit the table attributes of the nols-settings object as follows:

Attribute	Description
nols-studio-home-directory	The directory in which NOL Studio is installed, such as "C:\Program Files\Gensym\g2-2015\nolstudio".
nols-remote-process-listener-port	The port that the NolG2Gateway class uses, which is 22044, by default.
nols-connection-timeout	A connection timeout, which is 10 seconds, by default.
nols-interface-object-name	The name of the nols-studio-gateway object that provides communication, which is a subclass of gsi-interface. This object is created transiently by the launch procedures and referenced by the initialization procedures.
nols-execution-command	The name of execution file to launch the interface.
nols-host	The name of host machine.
nols-remote-process-id	The ID return by the remote process.
others	Not used in this version.

4 Launch the NOL Studio using the nols-settings object.

The procedure used to manage the NOL Studio console are:

nols-launch-nolstudio-by-setting
(*settings*: class nols-settings, *client*: class ui-client-item
-> *id*: float

nols-kill-remote-studio
(*win*: class g2-window)

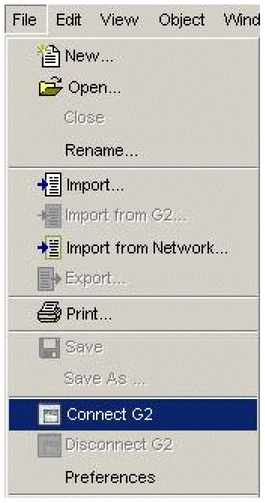
The launching procedure uses the setting you create for your application. You can create action buttons on a workspace in your module to start these procedures. When the remote process is started successfully, you will see that the NOL Studio console is launched and connected with the G2. If you terminate the remote process successfully, the NOL Studio console will no longer exist.

Connecting G2 from NOL Studio

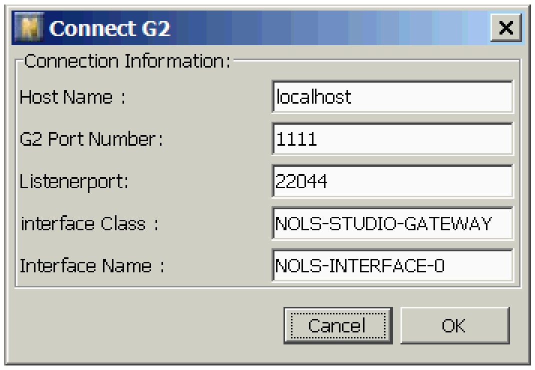
You can connect from NOL Studio to an existing G2 process by using nols-studio-gateway bridge.

To connect to G2 from NOL Studio:

➔ Choose File > Connect G2:



The Connect G2 dialog appears, for example:


Connect G2

Connection Information:

Host Name :	localhost
G2 Port Number:	1111
Listenerport:	22044
interface Class :	NOLS-STUDIO-GATEWAY
Interface Name :	NOLS-INTERFACE-0

If the connection is successfully established, the connection information appears in the toolbar, for example:

localhost:1111

You can now exchange data and export model parameters between NOL Studio and G2.

Note The NOL Studio to G2 gateway is associated with each G2 window. Each G2 window can only have one NOL Studio connection. If you connect G2 from the NOL Studio side, the gateway is associated with the G2 server. To have a NOL Studio for any particular Telewindows, launch the NOL Studio console from that Telewindows.

Accessing the Integration API

The complete set of API for interactions between NOL Studio and G2 are listed in Interaction with NOL Studio.

Actions for Data Exchange and Parameter Passing

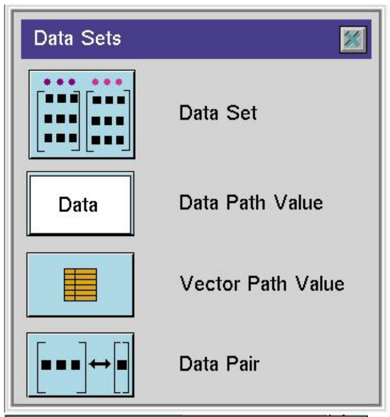
The description of actions to exchange data between NOL Studio and G2 are given under each type of blocks in Object Reference.

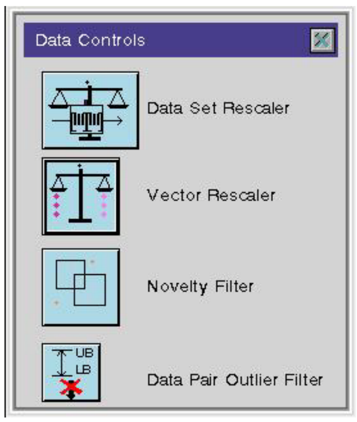
Object Reference

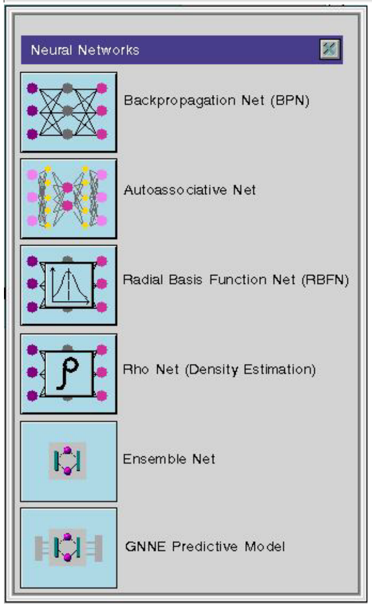
Provides a reference for the GNNE objects.

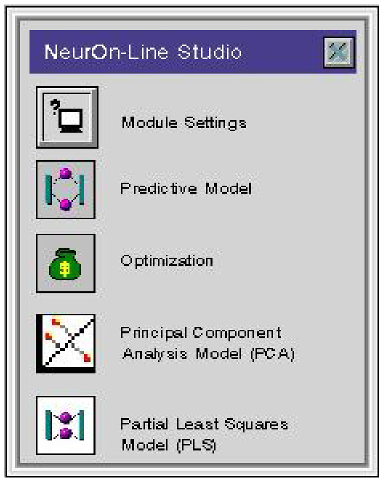
Introduction

This chapter describes the behavior and specific properties of each GNNE object. The objects are organized by the palette on which they appear. The palettes for GNNE objects in G2 server are:

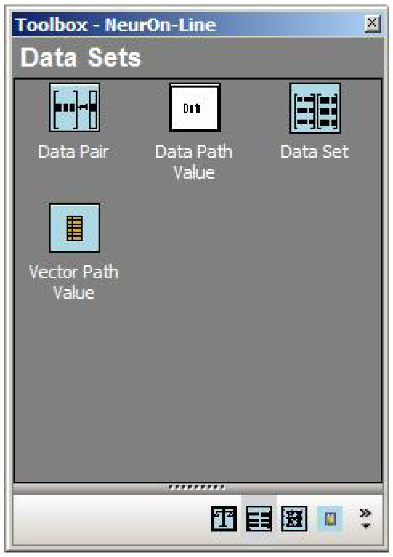


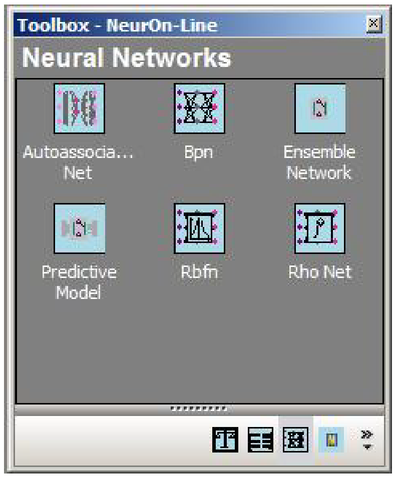


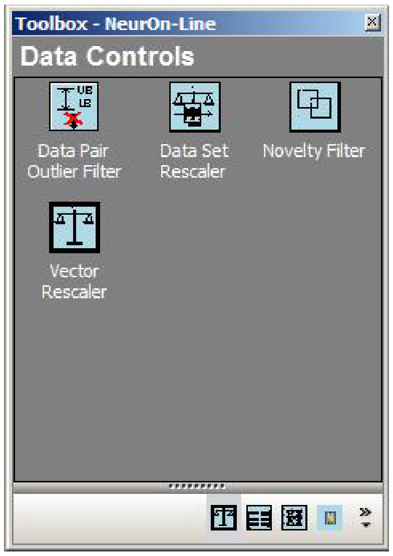

Neural Networks
Backpropagation Net (BPN)
Autoassociative Net
Radial Basis Function Net (RBFN)
Rho Net (Density Estimation)
Ensemble Net
GNNE Predictive Model

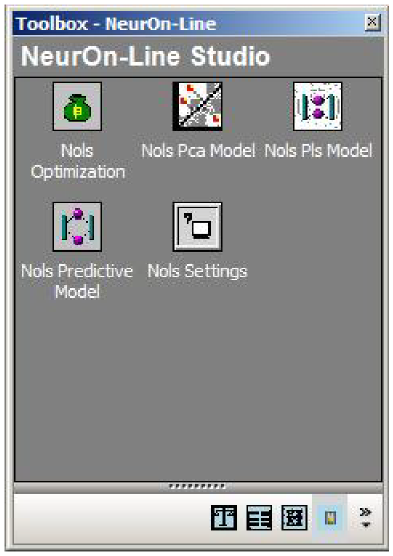

NeurOn-Line Studio
Module Settings
Predictive Model
Optimization
Principal Component Analysis Model (PCA)
Partial Least Squares Model (PLS)

The corresponding palettes in Telewindows are:





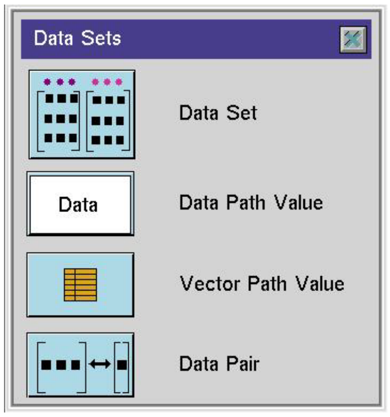


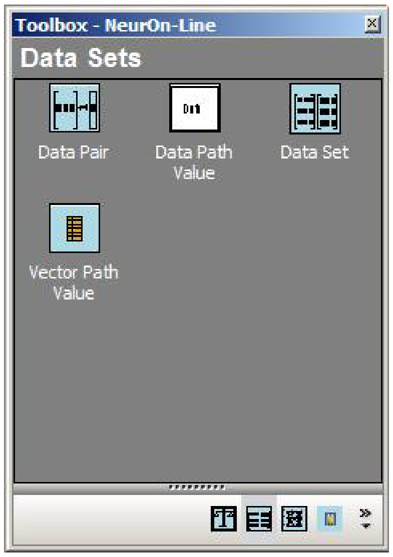


Data Objects

The blocks in the Data Sets palette are:

- Data Set
- Data Path Value
- Vector Path Value
- Data Pair





Data Set

A Data Set stores data pairs for training and testing neural networks.

A Data Set contains three matrices: input, target, and predictions. Whenever the Data Set receives a data pair, it adds the data pair's X vector to the end of the input matrix and the data pair's Y vector to the end of the target matrix. When a Fit Tester tests a neural network with a Data Set, it fills the predictions matrix with the values that the network predicts for each element of the input matrix.

The number of columns in the input matrix is the same as the dimension of the largest X vector. The number of columns in the target and predictions matrices is the same as the dimension of the largest Y vector. If the Data Set receives a data pair with an X or Y vector that is smaller than the input or target matrix, the Data Set pads that vector with zeros. If the Data Set receives a data pair with an X or Y vector that is larger than the input or target matrix, the Data Set adds a column to the appropriate matrix and pads the previous elements with zeros.

A Data Set has no configurable attributes.

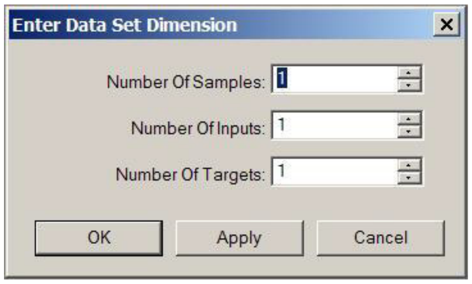
Editing the Data Set

To edit a data set, you must:

- Set the dimensions of the data set.
- Edit the data set.

Setting the Dimensions of the Data Set

To set the dimensions of the data set, select the edit data set menu choice on the Data Set object. When you first edit a Data Set that contains no data, GNNE displays this dialog for entering the Number of Samples, the Number of Inputs, and the Number of Targets:



Number Of Samples: 1
Number Of Inputs: 1
Number Of Targets: 1

Enter values for each of these attributes, and click the OK button to display the spreadsheet for editing the data set.

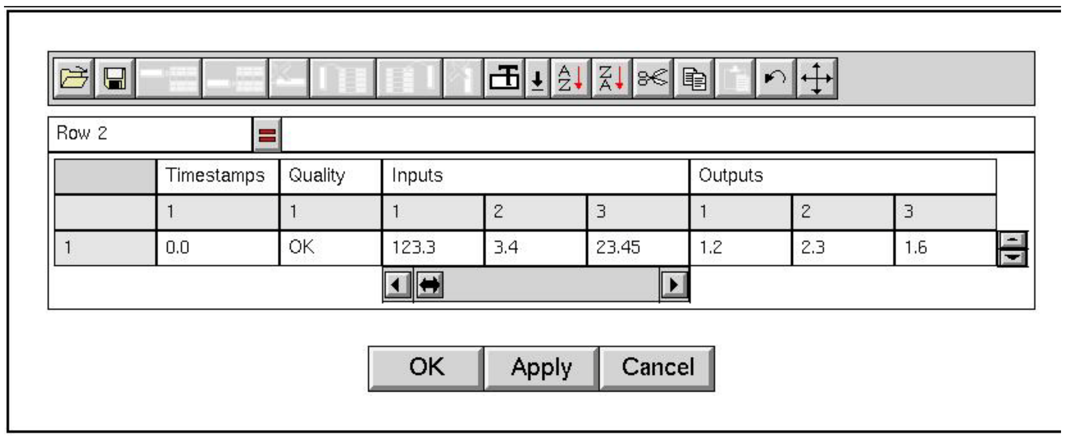
If your data set already contains data and you select the edit data set menu choice, GNNE does not display this dialog. Instead, GNNE displays the spreadsheet directly.

Entering and Viewing Data

To edit the contents of a Data Set that is initially empty, click OK in the Enter Data Set Dimensions dialog displayed above. GNNE displays a spreadsheet for editing the inputs and targets of the data set, and for viewing the predictions, timestamps, and quality.

To view or edit the contents of a Data Set that already contains data, simply select the edit data set menu choice. GNNE displays the spreadsheet directly.

Here is a spreadsheet for a data set with four inputs and three targets:



	Timestamps	Quality	Inputs			Outputs		
	1	1	1	2	3	1	2	3
1	0.0	OK	123.3	3.4	23.45	1.2	2.3	1.6

The samples are numbered down the left side of the editor. The editor shows samples 1. To see the other samples, use the vertical scroll bar. The data is split into four sections labeled Timestamps, Quality, Inputs, and Outputs. If there is more than one input or output in each sample, these sections can contain several columns, numbered 0, 1, and so on. The editor shows samples 1 through 3. To see the other samples, use the horizontal scroll bars.

You enter input and output data for the data set by either:

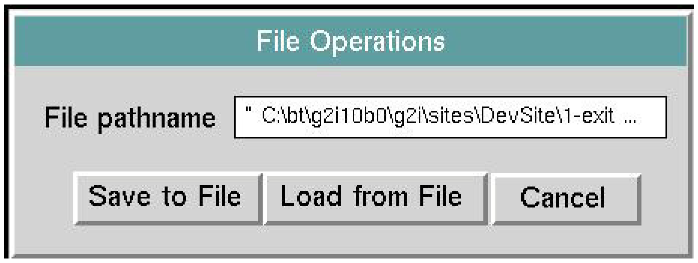
- Editing the spreadsheet cells directly.
- Reading the data from a file.

For more information on how to use the spreadsheet, see the *G2 XL Spreadsheet User's Guide*.

Saving and Loading Data

You can save or load the complete data set to or from a file.

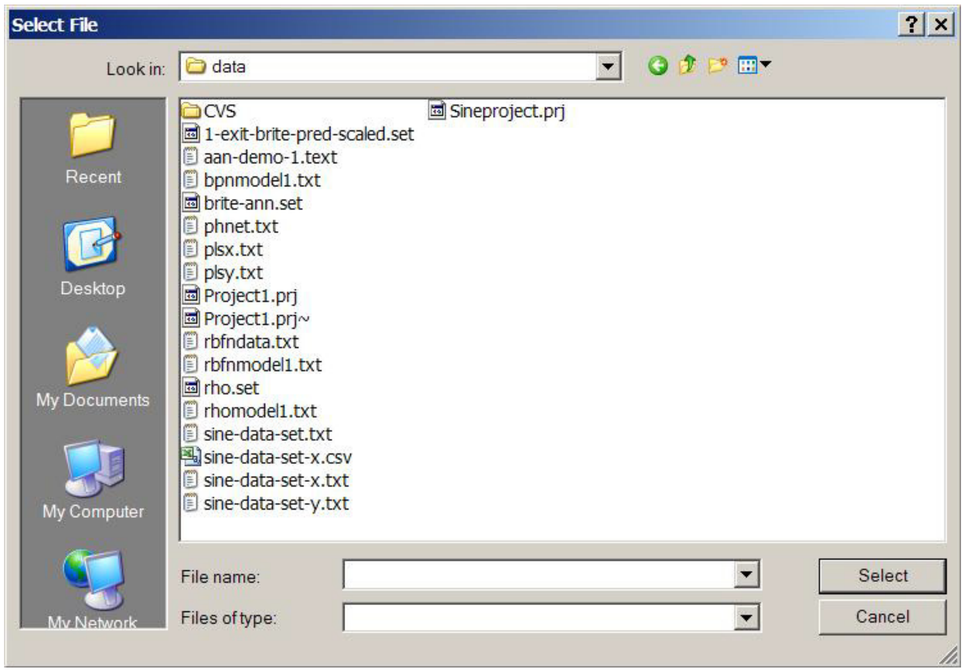
In the G2 server, to load a data set from a file, choose file operations on the Data Set object to display this dialog:



File pathname: " C:\bt\g2i10b0\g2i\sites\DevSite\1-exit ...

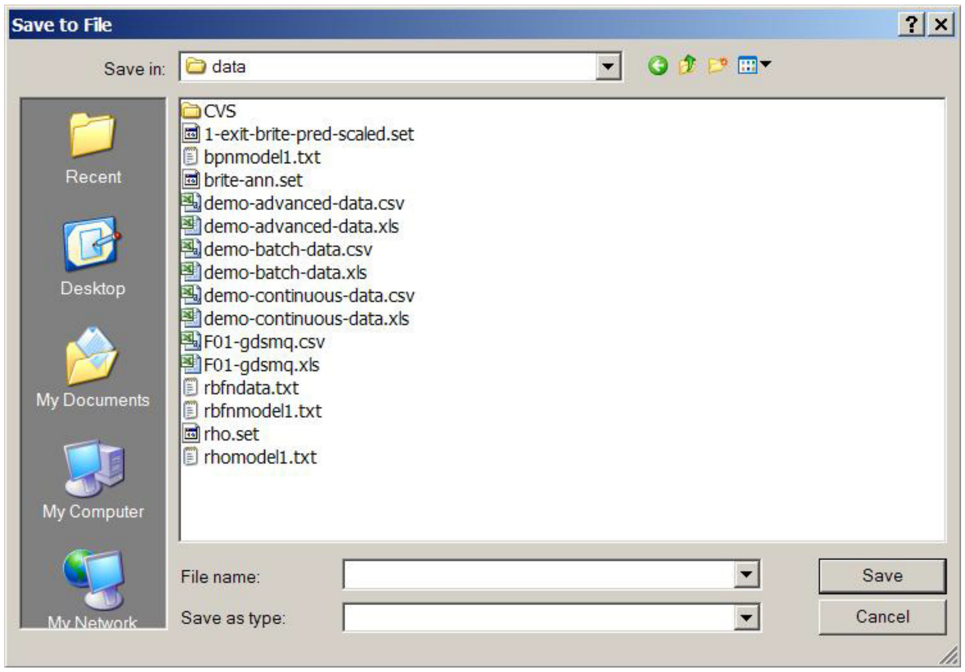
Enter the name of the file from which to load the data and click the Load from File button. To save a data set to a file, select the file operations menu choice, enter the filename, and click the Save to File button.

In Telewindows, to load a data set from a file, choose load from file on the Data Set object to display this file dialog:



CVS
1-exit-brite-pred-scaled.set
aan-demo-1.text
bpnmodel1.txt
brite-ann.set
phnet.txt
plsx.txt
plsy.txt
Project1.prj
Project1.prj~
rbfndata.txt
rbfnmodel1.txt
rho.set
rhomodel1.txt
sine-data-set.txt
sine-data-set-x.csv
sine-data-set-x.txt
sine-data-set-y.txt
Sineproject.prj

To save a data set to a file, choose save into file on the Data Set object to display this file dialog:



CVS
1-exit-brite-pred-scaled.set
bpnmodel1.txt
brite-ann.set
demo-advanced-data.csv
demo-advanced-data.xls
demo-batch-data.csv
demo-batch-data.xls
demo-continuous-data.csv
demo-continuous-data.xls
F01-gdsmq.csv
F01-gdsmq.xls
rbfndata.txt
rbfnmodel1.txt
rho.set
rhomodel1.txt

Text Format for Data Sets

The text format for saving and loading data sets from files consists of the following lines:

- The version number. For this version of NeurOn-Line, it is 1.
- The number of data pairs in the Data Set.
- The number of elements in each input vector.
- The number of elements in each target vector.
- Several lines of data, one line for each data pair in the Data Set. Each line contains the follow items, separated with commas:
  - The number of the data pair, numbered consecutively starting with 0.
  - The timestamp for the data pair. It can be either a float or an integer.
  - The quality of the data pair. It can be *OK*, `manual`, or `no-value`.
  - The input and target values of the data pair, starting with the input values.

Optionally, a line can contain a comment, which begins with a semicolon and continues to the end of the line.

Here is an example of a Data Set stored as text:

```
1; Version of this save/restore protocol for data sets
4 ; Number of samples in this data-set
2 ; Length of each input data vector
1 ; Length of each output data vector
0, 9516, OK, 0.000000000,0.000000000, 0.000000000
1, 9520, OK, 0.000000000,1.000000000, 1.000000000
2, 9524, OK, 1.000000000,0.000000000, 1.000000000
3, 9528, OK, 1.000000000,1.000000000, 0.000000000
```

Customizing the Text Format

By writing your own G2 procedures, you can customize the file format associated with a data set.

For more information, see Application Programmer's Interface.

In the data set object's attribute table, set the attributes file-save-procedure and file-load-procedure to the names of the procedures that read and write using your format. Your file save and load procedures must save and load the following attributes of a data set:

- input-data-set (class gnne-a-matrix)
- output-data-set (class gnne-a-matrix)
- time-stamps (class quantity-array)
- qualities (class symbol-array)

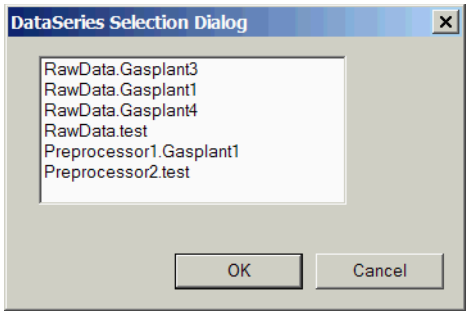
Note Two new procedures gnne-write-data-set-to-stream-with-predictions and gnne-read-data-set-from-stream-with-or-without-predictions have been added into GNNE, to make sure the predication arrays were also saved into data sets, when they were set as the values of attributes file-save-procedure and file-load-procedure.

Use the API procedure nol-configure-data-set to resize the elements of a data set. The procedure g2-get-matrix-dimensions tells you the current dimensions of the input Data Set and output Data Set matrices. These API procedures allow you to save and load parts of data sets:

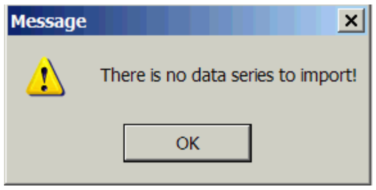
- nol-read-array
- nol-write-array
- nol-read-matrix
- nol-write-matrix

Loading the Data Set From NOL Studio

You can load the complete data set from a connected NOL Studio console. The function is only available through Telewindows. You need to launch a NOL Studio console from that Telewindows. To load a data set after the NOL Studio is launched and connected, choose Import Data From NOL Studio on the Data Set object to display this dialog:

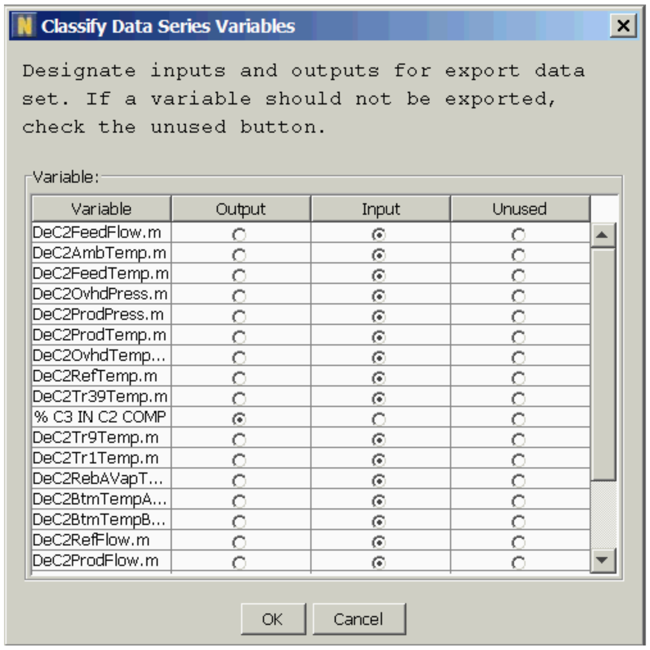


If there is no data series in the NOL Studio, the following dialog appears:



There is no data series to import!

After you select the data series in the data series selection dialog and click OK, the classification dialog shows in the NOL Studio console for specifying the input and output variables for this data set:



Designate inputs and outputs for export data set. If a variable should not be exported, check the unused button.

Variable:

Variable	Output	Input	Unused
DeC2FeedFlow.m	○	●	○
DeC2AmbTemp.m	○	●	○
DeC2FeedTemp.m	○	●	○
DeC2OvhdPress.m	○	●	○
DeC2ProdPress.m	○	●	○
DeC2ProdTemp.m	○	●	○
DeC2OvhdTemp...	○	●	○
DeC2RefTemp.m	○	●	○
DeC2Tr39Temp.m	○	●	○
% C3 IN C2 COMP	●	○	○
DeC2Tr9Temp.m	○	●	○
DeC2Tr1Temp.m	○	●	○
DeC2RebAVapT...	○	●	○
DeC2BtmTempA...	○	●	○
DeC2BtmTempB...	○	●	○
DeC2RefFlow.m	○	●	○
DeC2ProdFlow.m	○	●	○

Click OK to export this data series into the gnne-data-set object in G2.

Clearing the Data Set

You can call gnne-clear-data-set method to clear the data set.

Making Values Permanent

When you choose make permanent from the Data Set's menu, it saves all the Data Set's current values.

Data Path Value

Data Path Value objects are for storing a scaler data value from neural network training and testing method calls.

Property	Description
Data Value	The stored scaler value.
Quality	Determines whether the data value is valid.
Collection Time	A float number giving the G2 time when the new data value is collected.
Expiration Time	A float number giving the G2 time when the new data value is no longer valid.

Vector Path Value

Vector Path Value objects are used for storing vector value from neural network training and testing method calls.

Property	**Description**
Array Length	The length of stored vector.
Initial Values	The initial values for all vector elements when the vector is created.
Quality	Whether the data value is valid.
Collection Time	A float number giving the G2 time when the new data value is collected.
Expiration Time	A float number giving the G2 time when the new data value is no longer valid.

Data Pair

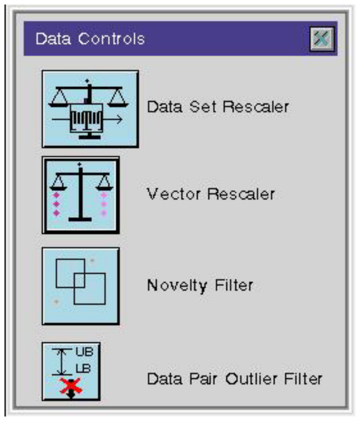
The Data Pair stores a pair of vectors. The left input becomes the X vector and the right input becomes the Y vector. Data Pair objects carry values as input parameters or output parameters for variate neural network method calls.

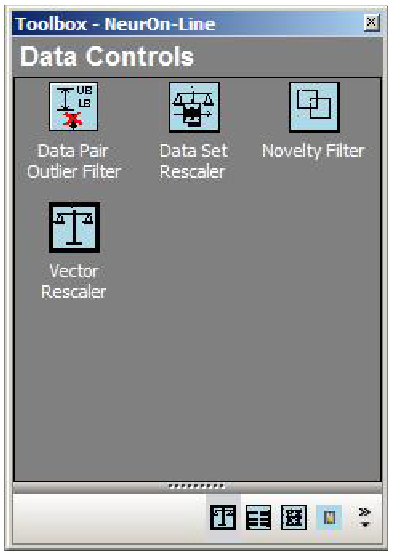
Property	Description
Input Data	The left vector as input vector X.
Target Data	The right vector as output vector X.
Quality	Whether the data value is valid.
Collection Time	A float number giving the G2 time when the new data value is collected.
Expiration Time	A float number giving the G2 time when the new data value is no longer valid.

Data Controls

The blocks in the Data Controls palette are:

- Data Set Rescaler
- Vector Rescaler
- Novelty Filter
- Data Pair Outlier Filter





Data Set Rescaler

The Data Set Rescaler scales the input and target data in a Data Set. You can specify your own scaling factors or let the Data Set Rescaler create them using one of two standard scaling methods: Min-Max scaling or Mean-Standard Deviation scaling. It can also create two Vector Scaling blocks that scale vectors in the same way that the Data Set Rescaler scales a Data Set's input and target data.

If your data has wide variations, you may need to rescale it to train the network best. However, after you train the network with scaled data, you will get invalid results if you apply that network to raw data. Also the network's output data is scaled and is different from the raw target data. To solve these problems, the Data Set Rescaler not only scales the training data, it also creates two Vector Rescalers that undo the scaling so you can apply the network to raw input data and interpret the network's output.

To use the Data Set Rescaler:

1 Create a Data Set Rescaler for one Data Set by cloning it from the palette.

2 Choose configure on the Data Set Rescaler and choose one of the scaling options.

3 Rescale the Data Set.

Calling the rescaling procedure gnne-rescale-data-sets with the original Data Set and scaled Data Set as argument.

4 Create Vector Rescaler blocks.

Call the procedure gnne-rescale-data-sets to make vector rescalers. The procedure returns two Vector Rescalers: the first one contains the scaling factors that the Data Set Rescaler used for input data, and the second one contains the scaling factors the Data Set Rescaler used for target data.

5 Train the neural network with the scaled Data Set.

To make the neural network more accurate and robust, call the training procedure for the network with the scaled Data Set.

6 Use Vector Rescalers for network execution.

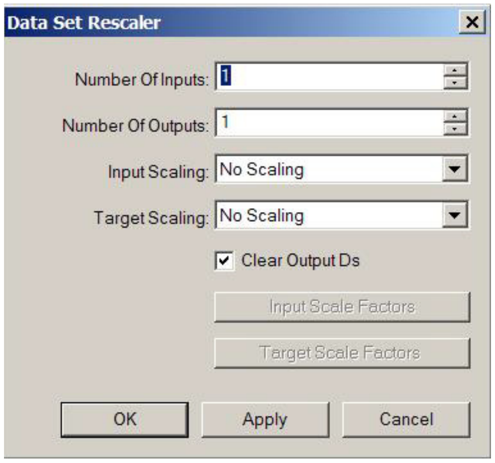
The Vector Rescalers undo the scaling that the Data Set Rescaler applied. You can use raw input data and interpret the original units.

Making Values Permanent

Choose make permanent on the block to save the scale factors.

Configuring

To choose how the Data Set Rescaler performs its scaling, choose properties:



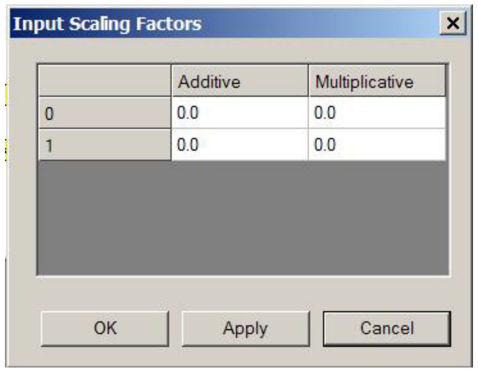
Number Of Inputs: 1
Number Of Outputs: 1
Input Scaling: No Scaling
Target Scaling: No Scaling
[✓] Clear Output Ds

First, enter the number of inputs in the Number of Inputs attribute and the number of targets in the Number of Targets attribute.

Next, choose the scaling options for the input and output data by selecting options for Input Scaling and Target Scaling. The options are:

Option	Description
no scaling	Do not scale the data. For each column, use 0 as the additive scaling factor and 1 as the multiplicative scaling factor.
0-1 min-max	Scale each column so that the maximum value is 1 and the minimum value is 0.
0-1 mean-stdev	Scale each column so that the column's mean is 0 and its standard deviation is 1.
custom scaling	Scale each column using scaling factors that you specify.

If you choose custom scaling, NeurOn-Line activates the Input Scale Factors and/or Target Scale Factors buttons, depending on whether you chose custom scaling for the input or target data. To enter your own scale factors, click the Input Scale Factors button or the Target Scale Factors button, and enter the factors in the spreadsheet that appears. Here is the dialog for entering custom scale factors for a target vector of width 2:



	Additive	Multiplicative
0	0.0	0.0
1	0.0	0.0

Each element in a column Column_i is rescaled according to the following formula, where A_i is the additive factor for that column's elements and M_i is the multiplicative factor for that column's elements:

$$\text{Column}'_i = M_i(\text{Column}_i + A_i)$$

For information on how to use this block through the API, see Application Programmer’s Interface.

Vector Rescaler

The Vector Rescaler block rescales the elements of the input vector by applying additive and multiplicative factors to each element. The block has different factors for each element.

Each element Input_i is rescaled according to this formula, where A_i is the additive factor for that element and M_i is the multiplicative factor for that element:

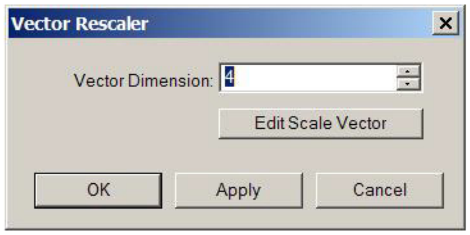
$$\text{Output}_i = M_i(\text{Input}_i + A_i)$$

Making Values Permanent

Choose make permanent on the block to save the current scaling factors.

Configuring

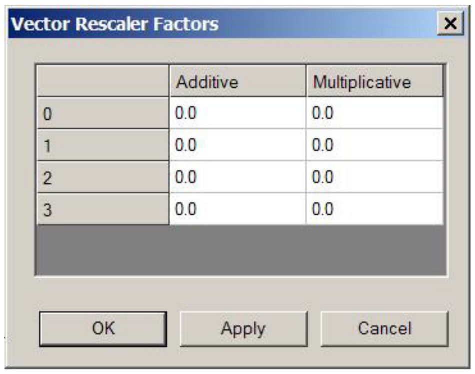
Choose properties to display the following dialog for entering the dimension of the vector to rescale:


Vector Rescaler

Vector Dimension: 4

Enter a number for Vector Dimension and click the Edit Scale Factors button to display the spreadsheet for rescaling the vector.

Here is the spreadsheet for rescaling a vector of length 4:



Vector Rescaler Factors

	Additive	Multiplicative
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0

Enter values in the Additive and/or Multiplicative columns to add values to the current index and/or multiply values by the current index and click OK.

For information on how to use this block through the API, see Application Programmer's Interface.

Novelty Filter

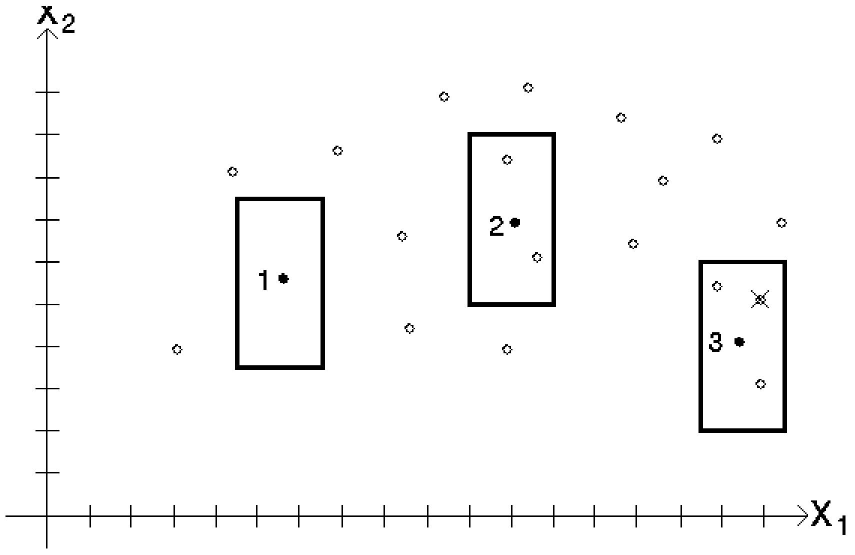
The Novelty Filter is a filter that prevents a Data Set from being filled with redundant data. Whenever you add a data pair to the attached Data Set, the Novelty Filter checks whether there are more than a specified number of data pairs within a specified distance of the new data pair. If there are, the Novelty Filter removes the older data pair.

To filter a Data Set, call the procedure gnne-execute-novelty-filter.

Choosing Which Points to Keep

Whenever the need-filtered Data Set receives a new data pair, the Novelty Filter encloses the input value with a rectangular cell. If that cell contains more than the maximum specified in the Points per Cell attribute, the Novelty Filter removes the oldest data pair. You set the sizes of the cell in the properties dialog. Each input has its own cell size, which is one-half the cell's width.

In the example below, there are three newly added data pairs (the filled circles) and a large number of existing data pairs (the empty circles). Each data pair has two inputs (X1 and X2). In the Novelty Filter's properties dialog, the size for X1 is 1 and the size for X2 is 2. This means that each new point is enclosed by a cell that is 2 by 4 units large. The maximum points per cell is 3.



The Novelty Filter handles the three new points as follows:

- Since the cell contains fewer than the maximum Points per cell, nothing is removed.
- Since the cell contains exactly the maximum Points per cell, nothing is removed.
- Since the cell contains more than the maximum Points per cell, the oldest data pair in the cell is removed. In the example, that data pair has an X through it.

Deciding Whether a Data Pair is Novel

The Novelty Filter passes a control signal when it receives a data pair that it determines is novel. However, a data pair is not novel just because the filter keeps it. An incoming data pair is judged to be novel if either of the following criteria is satisfied:

1 If the input cell contains only the newly received data pair, the data pair is novel.

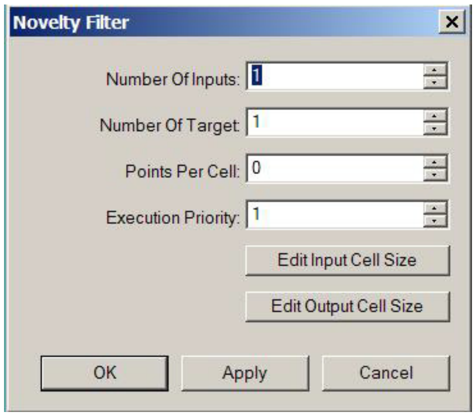
2 If the input cell contains other data pairs, the Novelty Filter averages the target values (or Y values) for those data pairs. Then, it computes the target values for the received data pair and encloses it in a cell. You specify the dimensions for the cell in the filter's configuration panel. These are different dimensions from the ones for the input cell. If the average output values fall outside the cell, the newly received data pair is novel.

Making Values Permanent

Choose make permanent on the block to save the sizes of all the input and output values.

Configuring

Here is the properties dialog for the Novelty Filter:



Number Of Inputs: 1
Number Of Target: 1
Points Per Cell: 0
Execution Priority: 1

Set Number of Inputs to the number of input values in each data pair, and set Number of Targets to the number of output values in each data pair. Set Points per Cell to the maximum number of data pairs you want inside each cell.

Once you have specified these attributes, click Edit Input Cell Sizes and Edit Output Cell Sizes to edit the cell sizes. A dialog appears for entering the size of each input and output value.

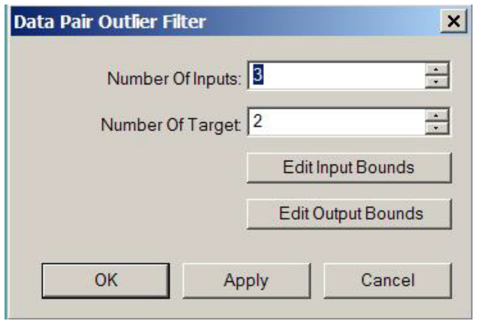
For information on how to use this block through the API, see Application Programmer's Interface.

Data Pair Outlier Filter

The Data Pair Outlier Filter separates any data pairs whose elements do not fall within specified bounds. The data pairs whose elements fall within the bounds are passed through the right output port. The other elements are passed through the bottom output port.

Configuring

To set the upper and lower bounds for each element, choose properties on the block to display this dialog:


Data Pair Outlier Filter

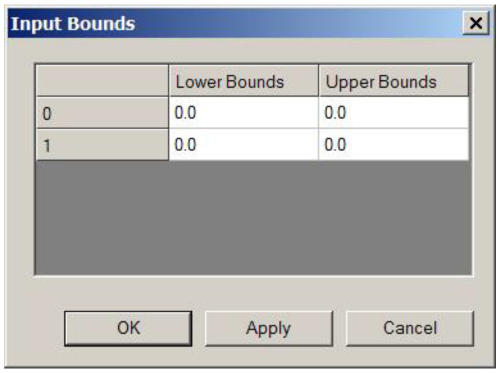
Number Of Inputs: 3

Number Of Target: 2

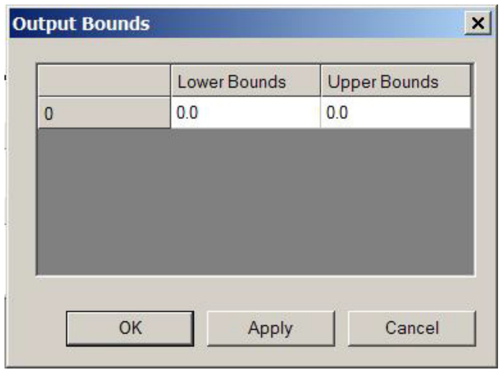
To specify the number of elements of each vector, enter the dimensions for the data pair's x and y vectors in the Number of Inputs and Number of Targets attributes, respectively.

To edit the input vector’s bounds, click the Edit Input Bounds button. To edit the target vector’s bounds, click the Edit Target Bounds button.

Here are the dialogs for editing the x and y bounds of a data pair with two inputs and 1 target:


Input Bounds

	Lower Bounds	Upper Bounds
0	0.0	0.0
1	0.0	0.0


Output Bounds

	Lower Bounds	Upper Bounds
0	0.0	0.0

The editor has two columns: Lower Bounds and Upper bounds. When you enter the bounds, the Upper Bounds must always be greater than the Lower Bounds. Therefore, you must enter the values in a specified order. Click OK when you are finished.

Making Values Permanent

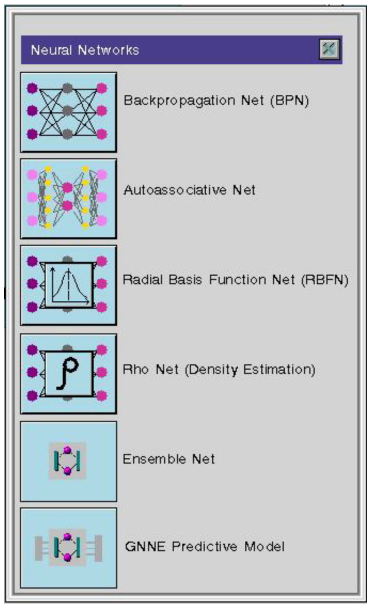
Choose make permanent on the block to save the filter's upper and lower bounds.

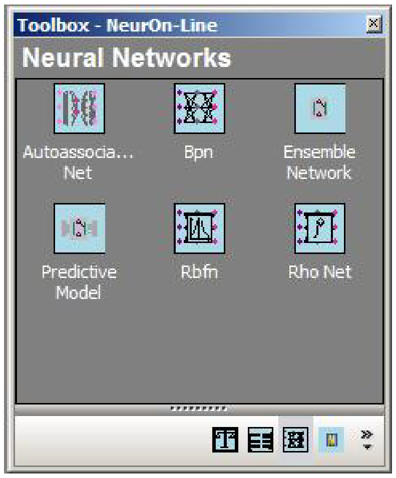
For information on how to use this block through the API, see Application Programmer's Interface.

Neural Networks

The blocks in the Neural Networks palette are:

- Backpropagation Net (BPN)
- Autoassociative Net
- Radial Basis Function Net (RBFN)
- Rho Net
- Ensemble Net (ENN)
- GNNE Predictive Model


Neural Networks
Backpropagation Net (BPN)
Autoassociative Net
Radial Basis Function Net (RBFN)
Rho Net (Density Estimation)
Ensemble Net
GNNE Predictive Model

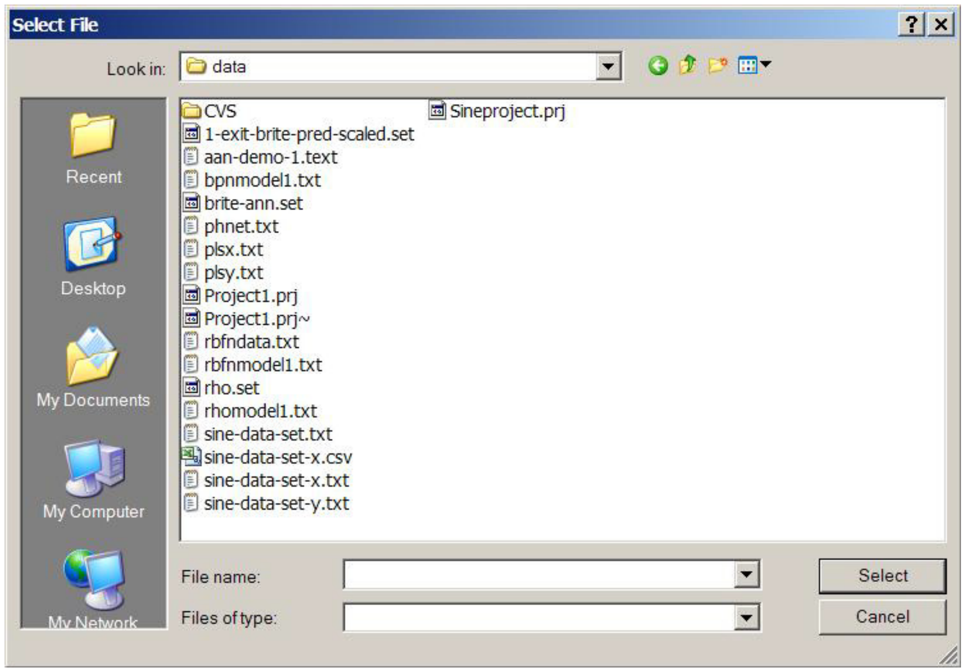


Saving and Loading Network Weights

All Neural Network objects let you save and load what they have learned so far. All Neural Network objects also use text files to load parameters exported from NOL Studio. NOL Studio and GNNE save the information to text files, which you can examine. To load network parameters exported from NOL Studio, you can call corresponding APIs for that particular network to read parameters from text file, or you can choose Load from File to load the parameters interactively.

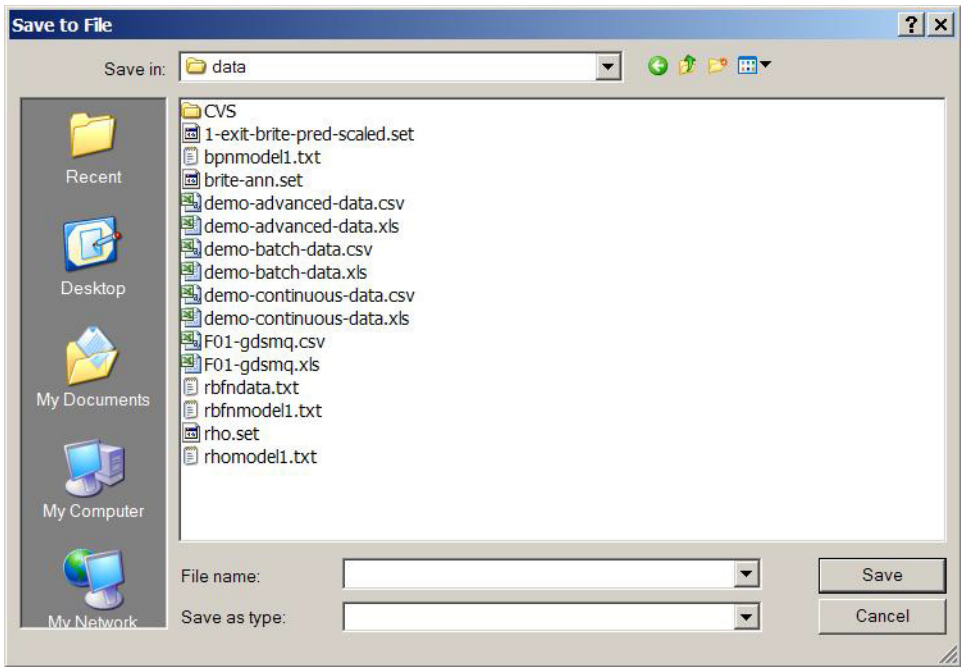
Loading Model Parameters from a Text File

To load parameters of a neural network from a text file through Telewindows, choose Load From File on the neural network object to display this dialog:



CVS
1-exit-brite-pred-scaled.set
aan-demo-1.text
bpnmodel1.txt
brite-ann.set
phnet.txt
plsx.txt
plsy.txt
Project1.prj
Project1.prj~
rbfndata.txt
rbfnmodel1.txt
rho.set
rhomodel1.txt
sine-data-set.txt
sine-data-set-x.csv
sine-data-set-x.txt
sine-data-set-y.txt
Sineproject.prj

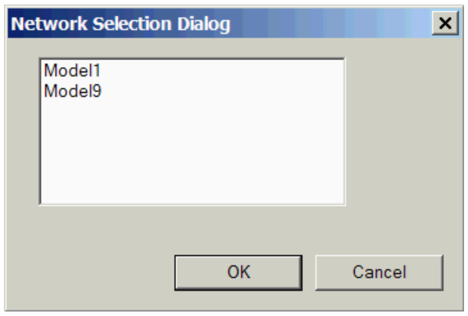
To save the parameters of a neural network to a file, choose Save Into File on the neural network object to display this dialog:



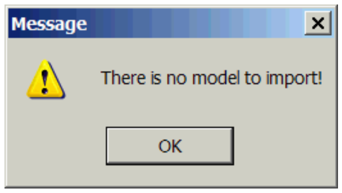
CVS
1-exit-brite-pred-scaled.set
bpnmodel1.txt
brite-ann.set
demo-advanced-data.csv
demo-advanced-data.xls
demo-batch-data.csv
demo-batch-data.xls
demo-continuous-data.csv
demo-continuous-data.xls
F01-gdsmq.csv
F01-gdsmq.xls
rbfndata.txt
rbfnmodel1.txt
rho.set
rhomodel1.txt

Import Model Parameters from NOL Studio

If an existing NOL Studio console is connected to the G2 process, you can import the model parameters directly from that NOL Studio. In Telewindows, to import model parameters of a neural network from a connected NOL Studio owned by that Telewindows, choose Import Parameters From NOL Studio on the neural network object to display this dialog:



The available model names corresponding to the neural network type appear in the selection list. Select the model name and click OK to import the parameters into the neural network object in G2. If there is no available model object in NOL Studio, the following dialog appears:



There is no model to import!

Backpropagation and Autoassociative Network File Format

The text format for saving and loading BPNs from files consists of the following lines.

Note All lists are comma-separated and are in order beginning with the item for the first layer. A comment begins with a semicolon and continues to the end of the line.

1. The version number. For this version of NeurOn-Line, it is 1.
2. The number of layers in the network.
3. A list of the number of nodes in each layer.
4. A list of the transfer functions for each layer. The number 0 stands for linear, and the number 1 stands for sigmoid.
5. The weight of each node in the network. Each weight is on a separate line and is followed by a comment that identifies which nodes it is for. The convention

used to identify the nodes is described below. Note that the bias node is included as an extra node in each layer, except the output layer.

6 The weights are listed in order. The first is for the connection from the first node in the first layer to the first node in the second layer. The last is for the connection from the last node in the second-to-last layer to the last node in the last layer.

Each weight is followed by a comment that identifies which nodes it is for. The comment contains three numbers, as follows.

```
; i j k
```

This is the weight from node i in layer k to node j in layer k+1. For example, the following weight is for the second node in the third layer to the first node in the fourth layer.

```
0.7242780000 ; 2 3 1
```

Here is an example of a file for a BPN, with 3 layers, 2 input layer nodes, 3 hidden layer nodes, and 1 output layer node:

```
; Version of this file save/restore protocol for BPNs
3 ; Number of layers
2, 3, 1 ; Layer sizes of BPN.
0, 1, 0 ; Transfer functions of BPN.
-0.32507146396688 ; 1 1 1
-0.82195669379450 ; 1 2 1
0.19680059179068 ; 1 3 1
-0.97116809332961 ; 2 1 1
0.61150472166297 ; 2 2 1
-0.12016215756566 ; 2 3 1
0.84987859399967 ; 3 1 1
-0.74007775586113 ; 3 2 1
-0.63971444152242 ; 3 3 1
0.84097431197944 ; 1 1 2
-0.79330091884975 ; 2 1 2
-0.01523408110297 ; 3 1 2
0.43417445464837 ; 4 1 2
```

Radial Basis Function and Rho Network File Format

The text format for saving and loading RBFNs from files consists of the following lines.

Note All lists are comma-separated. A comment begins with a semicolon and continues to the end of the line.

1 The version number. For this version of NeurOn-Line, it is 1.

2 A list of the number of nodes in each layer.

3 The unit overlap.

4 Whether the network uses spherical or elliptical units. The number 0 stands for spherical units, and the number 1 stands for elliptical units.

5 The locations of the sphere or ellipse centers. The locations for each row are on a separate line. Each line contains as many numbers as there are elements in the input vector. The number of location lines is the same as the number of hidden units.

6 The shapes of the units. If you are using spherical units, there is one line for each hidden unit, and each line contains the width for the unit. If you are using elliptical units, there are N*H lines, and each line contains N values, where N is the number of input values and H is the number of hidden units. The first N lines represent the inverse covariance matrix of radial unit 1, the next N lines represent the inverse covariance of the second radial unit, etc.

7 The weights for the output layer. There is one line for each node in the hidden layer, and each line contains the weights from the hidden node to the node in the output layer. If this is a RBFN, the weights for the bias node are on an extra line at the end.

Here is an example of a file for an RBFN with spherical units.

```
1; Version of this file save/restore protocol for RBFNs
3, 6, 1 ; Layer sizes of RBFN.
2 ; Unit overlap parameter
0 ; Spherical unit shape
1 ; Bias on
10.8896000000, 5.0603000000, 5.1376100000 ; Unit centers row 1
11.4327000000, 5.5737400000, 5.6030400000 ; Unit centers row 2
7.8846400000, 5.0246100000, 5.0246100000 ; Unit centers row 3
9.4227900000, 5.1608600000, 5.1608600000 ; Unit centers row 4
10.1503000000, 5.5839500000, 5.2830700000 ; Unit centers row 5
6.7058600000, 5.0688600000, 4.9504500000 ; Unit centers row 6
0.8992220000 ; Unit shapes row 1
1.1230300000 ; Unit shapes row 2
1.3784100000 ; Unit shapes row 3
1.2011400000 ; Unit shapes row 4
0.8846430000 ; Unit shapes row 5
2.1013600000 ; Unit shapes row 6
-2.3439900000 ; Second layer weights row 1
-0.3743570000 ; Second layer weights row 2
-0.3669410000 ; Second layer weights row 3
-3.9031200000 ; Second layer weights row 4
0.6760480000 ; Second layer weights row 5
```

```
-0.9269620000 ; Second layer weights row 6
11.1002000000 ; Second layer weights row 7
```

Here is a file for an RBFN that has the same basic architecture as the one above, but that uses elliptical units.

```
1; Version of this file save/restore protocol for RBFNs
3, 6, 1 ; Layer sizes of RBFN.
2 ; Unit overlap parameter
1 ; Elliptical unit shape
1 ; Bias on
6.6357800000, 5.2808800000, 4.9493200000 ; Unit centers row 1
7.2610900000, 5.1641900000, 5.1641900000 ; Unit centers row 2
10.3036000000, 5.3620800000, 5.1868500000 ; Unit centers row 3
8.2783000000, 5.0083000000, 5.0083000000 ; Unit centers row 4
6.6201000000, 4.8137500000, 4.8137500000 ; Unit centers row 5
11.2850000000, 5.3348900000, 5.4306300000 ; Unit centers row 6
5.3577500000, 0.1998100000, -4.0963000000 ; Unit shapes row 1
0.1998100000, 2.8251900000, -1.3811100000 ; Unit shapes row 2
-4.0963000000, -1.3811100000, 7.7281600000 ; Unit shapes row 3
1.0171300000, 0., -1.2055800000 ; Unit shapes row 4
0.7242780000, 9.4219700000, -7.0106700000 ; Unit shapes row 5
-1.2055800000, -7.0106700000, 11.3346000000 ; Unit shapes row 6
0.3653470000, 0.2290360000, -0.7771570000 ; Unit shapes row 7
0.2290360000, 4.0050800000, -3.5144000000 ; Unit shapes row 8
-0.7771570000, -3.5144000000, 7.2356000000 ; Unit shapes row 9
0.3881750000, -0.1173250000, -0.4112650000 ; Unit shapes row 10
-0.1173250000, 17.4052000000, -15.6234000000 ; Unit shapes row 11
-0.4112650000, -15.6234000000, 25.1755000000 ; Unit shapes row 12
4.1952000000, 0.2980240000, -3.6536000000 ; Unit shapes row 13
0.2980240000, 5.1403200000, -4.1436400000 ; Unit shapes row 14
-3.6536000000, -4.1436400000, 11.2164000000 ; Unit shapes row 15
0.2617710000, 0.1799830000, -0.6148770000 ; Unit shapes row 16
0.1799830000, 3.1725600000, -2.8314600000 ; Unit shapes row 17
-0.6148770000, -2.8314600000, 5.4504000000 ; Unit shapes row 18
3.7661200000 ; Second layer weights row 1
0.4220330000 ; Second layer weights row 2
-0.8155410000 ; Second layer weights row 3
0.9443360000 ; Second layer weights row 4
-0.5184290000 ; Second layer weights row 5
1.9278200000 ; Second layer weights row 6
7.8096200000 ; Second layer weights row 7
```

Ensemble Network File Format

The text format for saving and loading ensemble networks from files consists of the following lines.

Note All lists are comma-separated. A comment begins with a semicolon and continues to the end of the line.

---

1. The version number. For this version of NeurOn-Line, it is 1.
2. The number of submodels in the network.
3. The text for all submodels. The format for the submodel is the same as the format for BPNs.

Predictive Model File Format

The Predictive Model in GNNE is called gnne-predictive-model. The Predictive Model is saved into an XML format.

Here is an example of a XML file for a Predictive Model:

```
<?xml version="1.0" encoding="UTF-8"?>
<GNNEPredictiveModel xmlns:xsi="" >
   <name>Model1</name>
   <comment></comment>
   <isTimeBased>true</isTimeBased>
   <modelStatistics>
      <modelRate>0.9643490282921242</modelRate>
      <outputStatistics>
         <variableStatistics>
            <name>% C3 IN C2 COMP</name>
.....
   <ensemble>
      <numberOfSubnet>5</numberOfSubnet>
      <bpn>
         <numberOfLayers>4</numberOfLayers>
         <layer>
            <layerIndex>1</layerIndex>
            <layerSize>20</layerSize>
            <transferFunction>0</transferFunction>
         </layer>
......
```

Backpropagation and Autoassociative Networks

Both the Backpropagation Net (BPN) object and the Autoassociative Net object are feed-forward networks with multiple layers. The Autoassociative Network is a type of Backpropagation Network with a specific architecture, which is especially good for handling certain types of problems, including sensor validation.

Radial Basis Function and Rho Networks

Both the Radial Basis Function Net (RBFN) object and the Rho Net object are 3-layer, feed-forward networks, whose middle layers use a multivariate Gaussian function. Both are especially good at handling classification problems. Their biggest difference is what their output values are. The Radial Basis Function Network can return any type of number. The Rho Network passes a number between 0.0 and 1.0, which represents the probability that the input value is in a particular class.

Ensemble Networks

The Ensemble Net object is an encapsulation object. The Ensemble Network is a set of Backpropagation Net blocks, which have been trained in NOL Studio. It has a specific architecture, which gives it accuracy and robustness.

Predictive Model

The GNNE Predictive Model object is an encapsulation object. The GNNE Predictive Model is a wrapper around an Ensemble Network, which is a set of Backpropagation Net blocks, and is trained in NOL Studio. The GNNE Predictive Model contains the input and output variable information, along with variable delays. The model manages the input and output data for online execution. The model can be saved and transferred in XML format.

Backpropagation Net (BPN)

The Backpropagation Network, or BPN, is a feed-forward, layered network. Each node in a layer is connected to all other nodes in the layer before it and the layer after it. It is especially useful for modeling multivariate functions.

The first layer and the input vector must be the same size. The last layer and the output vector must have the same size. The hidden or intermediate layers (layers between the first and last layers) can be any size. You can have up to three hidden layers, for a total of up to five layers. In general, a network has one hidden layer. The number of nodes depends on the complexity of the function that the network has to model. The more complex the function, the more nodes needed.

You can choose whether a layer uses the sigmoidal or linear function for its nodes. In general, the input and output layers use the linear function, and at least one of the hidden layers use a sigmoidal function.

Before you can pass data through a network, you must train the network. The number of data pairs in the training data set should be greater than the number of weights over the number of outputs. For example, if a network has 5 inputs, 10 hidden nodes and 3 outputs, its training data set should have at least this many data pairs.

$$\frac{10\text{hidden}(5\text{inputs} + 1\text{bias}) + 3\text{outputs}(10\text{hidden} + 1\text{bias})}{3\text{outputs}} = 31$$

In practice, several times this number is recommended.

When you pass a vector to a network, it calculates the value for its output vector by passing the input vector's data through the layers of its network. Passing data through a network does not change the values of its weights.

Configuring

To set the number of layers, number of nodes, and the transfer functions, you configure the BPN object by calling gnne-configure-bpn.

You can specify the number of layers between 2 to 5 and set the number of nodes and transfer function for each input layer. If the network contains fewer than five layers, some of the fields will be inactive. You specify the nodes and transfer functions for the output layers. See the example for gnne-configure-bpn in Neural Networks.

Caution If you change the architecture for a trained network by reducing the size of any layer, you must retrain the network.

Adjusting Weights

When you first clone a BPN from the palette, all its weights are set to zero. However, the network needs to contain small random weights to train properly. To fill the network with weights appropriate for training, you can call the gnne-randomize-bpn-weights method for BPN. GNNE overwrites the net's current weights with new random weights. You can specify an absolute amount to randomize by, a percentage to randomize by, or both. This is the formula that the object uses to jiggle the weights, where P is the percentage you entered, A is the absolute amount you entered, and R1 and R2 are random numbers from -1.0 to 1.0.

$$\text{NewWeight} = (1 + R_1 \cdot P/100) \cdot \text{OldWeight} + R_2 \cdot A$$

When you are training a network and it seems to be stuck in a local minimum, slightly changing the weights can help push the network back onto the right track.

Saving and Loading Weights

You can save the network's weights to a text file so you can load them later. The file format for saved weights is described in Saving and Loading Network Weights.

You use gnne-write-bpn-to-file and gnne-read-bpn-from-file to save or load network data. To load weights, overwriting the weights that are currently in the network.

Making Values Permanent

Choose make permanent on the neural network object to save the network's internal configuration and weight so that resetting G2 has no effect on their values.

For the information about how to use this block through API, see Application Programmer's Interface.

Autoassociative Net

The Autoassociative Network is a type of Backpropagation network that uses autoassociative mappings. It is a feed-forward, layered network. Each node in a layer is connected to all other nodes in the layer before it and the layer after it. In general, the input and output vectors are the same size and the network contains three hidden layers. It is especially useful for sensor validation problems.

An Autoassociative Network contains 5 layers. The first and last layers must be the same size as the input and output vectors. The hidden or intermediate layers (layers between the first and last layers) can be any size. Usually, an Autoassociative Network has 3 hidden layers. The middle layer, or bottleneck layer, must have fewer nodes than any other.

You can choose whether a layer uses the sigmoidal or linear function for its nodes. In general, the input, output, and bottleneck layers use the linear function, and the rest use the sigmoid function.

Before you can pass data through a network, you must train the network. When you pass a vector to a network, it calculates the value for its output vector by passing the input vector's data through the layers of its network. Passing data through a network does not change the values of its weights.

If you run an Autoassociative network when it is configured to correct gross errors, the Remote Process generates output like the following on the background window:

```
For no fault, f = 45.1403
For sensor 1, f = 6.33029, estimated bias = -8.995572
For sensor 2, f = 45.1061, estimated bias = 0.410265
For sensor 3, f = 45.0235, estimated bias = 0.482471
For sensor 4, f = 44.8587, estimated bias = 0.760695
For sensor 5, f = 44.4518, estimated bias = -1.263943
```

The f value is the input/output residual. To correct gross errors from various sensors, GNNE estimates the possible error or "estimated bias," based on the calculation of an expected value.

The higher the bias (a value away from 0), the more the Autoassociative network is correcting the sensor value. The sensor with the least input/output residual, like sensor 5 in the example output, is replaced by the Autoassociative network with its corrected value.

Configuring

To configure the Network Architecture, choose the number of layers, specify the number of nodes, and specify the transfer functions for each layer. You configure the Autoassociative Net object by calling gnne-configure-autoassociative-net described in Neural Networks.

You can specify the number of layers between 2 to 5 and set the number of nodes and transfer function for each layer. If the network contains fewer than five layers, some of the fields will be inactive. You specify the nodes and transfer functions for the output layers. See the example for gnne-configure-bpn in Neural Networks.

Caution If you reduce the number of nodes in any layer or reduce the number of layers for a trained network, the network's current weights will be meaningless and you will need to retrain the network.

We recommend that the first and last layer of an Autoassociative Network have the same number of nodes. If the network contains fewer than five layers, some of the fields will be inactive. You specify the number of nodes and transfer functions for the output layers.

By default, the values for the transfer functions alternate in the manner that is recommended for an Autoassociative Network: linear, sigmoid, linear, sigmoid, linear, for the input, first hidden, bottleneck, second hidden, and output layers, respectively.

Choosing the Run Mode

The configure method lets you choose whether the network replaces faulty input values. If you choose run mode as filter noise only, the network does not perform the replacement. When you run the network, it performs a single forward pass, which filters random errors from the inputs but not systematic errors (or biases).

If you choose the correct gross errors option, the network does perform the replacement. When you run the network, it performs N+1 passes, where N is the number of elements in the input vector.

The first pass is the same as the pass used for the noise filter only option. In the rest of the passes, one of the input values is ignored and the network computes the best replacement value. Using the standard deviation for the input value, the network computes how far off the input value is from its replacement value. The network then replaces the input value that is furthest from its replacement value.

Caution Correct gross errors mode requires several times more computational work than the filter noise only option.

When you choose the correct gross errors option, you need to provide an array of Noise Standard Deviations the network uses.

Adjusting Weights

You can overwrite the current weights with random weights and adjust the current weights by a random amount. For more information on adjusting weights, see Adjusting Weights for the BPN object.

Saving and Loading Weights

You can save the network's weights to a text file so you can load them later. For information on how to do this, see Saving and Loading Weights for the BPN object.

Making Values Permanent

Choose make permanent on the block to save the network's internal configuration and weights.

For information on how to use this block through the API, see Application Programmer's Interface

Radial Basis Function Net (RBFN)

The Radial Basis Function Network (or RBFN) is a 3-layer, feed-forward network, whose middle layer uses a multi-variate Gaussian function. It is especially useful for classification problems. The RBFN is best for choosing which class out of many classes an item belongs to. In general, RBFNs take less time to train but more time to execute than BPNs and Autoassociative Networks.

An RBFN contains exactly 3 layers. The first layer and the input vector must be the same size. The last layer and the output vector must be the same size. The middle or hidden layer can be any size.

Each node in a layer is connected to all other nodes in the layers before it and after it. The connections between the input and hidden layers are unweighted. However, RBFNs weight the connections between the hidden layer and output layer normally, like a BPN or an Autoassociative network.

The transfer function of the input and output layer is linear. You can choose whether the transfer functions of the hidden layer are spherical or elliptical Gaussians.

An RBFN can take a vector of any length as an input value, and it passes an output vector and an output scalar. The vector is the same size as the last layer in the network. The scalar is the maximum hidden node activation, which indicates how well the hidden layer covers the input vector. This number is between 0.0 and 1.0. If it is close to zero, for example less than 0.2, the hidden layer does not cover the input well, indicating that the network possibly predicted inaccurately due to extrapolation.

Configuring

To configure the Network Architecture, specify the number of nodes in the input, hidden, and output layers, the overlap between the nodes, and the shapes of the hidden nodes. You configure the RBFN Net object by calling gnne-configure-rbfn described in Neural Networks.

Caution If you change the architecture for a trained network by reducing the size of any layer, you will need to retrain the network.

To set the unit overlap, choose whether the overlap is automatic or fixed. If the overlap is automatic, the network chooses the best unit overlap for you automatically. Generally, you will use an automatic overlap. If the overlap is fixed, you need to provide a positive value for the unit overlap. The unit overlap affects how smoothly the trainer fits the function to the data. A larger unit overlap creates a smooth, slowly changing fit. A smaller unit overlap allows rapid changes in the fit.

To choose the function shape for the hidden layer, select spherical or elliptical. When data is sparse or the input values are not correlated to each other, spherical units may perform better. When more data is available or the input values are correlated to each other, elliptical units may perform better. If the input dimension is 1, there is no difference between spherical and elliptical nodes, and the network selects Spherical by default.

Saving and Loading Weights

You can save the network's weights to a text file so you can load them later. For information on how to do this, see Saving and Loading Weights for the BPN object.

Making Values Permanent

Choose make permanent on the block to save the network's internal configuration and weights.

For information on how to use this block through the API, see Application Programmer's Interface

Rho Net

The Rho Network is a 3-layer, feed-forward network, whose middle layer uses a multi-variate Guassian function. It is especially useful for classification problems. The Rho Net is best for deciding whether an item belongs to one particular class or not. In general, Rho Networks take less time to train but more time to execute than BPNs and Autoassociative Networks.

A Rho Network contains exactly 3 layers. The first layer and the input vector must be the same size. The last layer and the output vector must be the same size. The middle or hidden layer can be any size.

Each node in a layer is connected to all other nodes in the layers before it and after it. The connections between the input and hidden layers are unweighted. However, a Rho Net weights the connections between the hidden layer and output layer by using a probability function, which returns the likelihood that an input belongs to a particular class.

The transfer functions of the input and output layers are linear. You can choose whether the shape for the transfer functions of the hidden layer are spherical or elliptical.

A Rho Net can take a vector of any length as an input value, and it passes a vector. The contents of the vector depends on how you trained the network. When you call *gnne-train-rho-net* to train a Rho Network, you choose whether the Rho Network treats the training data as data from a single class or from multiple classes. If you choose single class, the output vector contains one element, which is the probability that the input element is in that class. If you choose multiple classes, the output vector contains an element for each class, and the value of each element is the probability that the output belongs to that element's class.

Configuring

To configure the Network Architecture, specify the number of nodes in the input, hidden, and output layers, the overlap between the nodes, and the shapes of the hidden nodes. You configure the a Rho Net object by calling gnne-configure-rbfn described in Neural Networks.

Caution If you change the architecture for a trained network by reducing the size of any layer, you will need to retrain the network.

To set the unit overlap, choose whether the overlap is automatic or fixed. If the overlap is automatic, the network chooses the best unit overlap for you automatically in the course of training. Generally, you will use an automatic overlap.

If the overlap is fixed, you need to provide a positive value in the configuration method call. The overlap parameter is a multiplicative factor applied to a basic hidden unit width, which is the nearest neighbor distance between the radial units. If the Unit Overlap is 1.0, each hidden unit's width is the distance to the nearest hidden unit. If the overlap parameter is 2.0, for example, the unit's width is twice the nearest neighbor distance. The unit overlap affects how smoothly the trainer fits the function to the data. A larger unit overlap creates a smooth, slowly changing fit. A smaller unit overlap allows rapid changes in the fit. The Unit Overlap should usually be between 0.5 and 5.0.

To choose the function shape for the hidden layer, you can select spherical or elliptical. When data is sparse or the input values are not correlated to each other, spherical units may perform better. When more data is available or the input values are correlated to each other, elliptical units may perform better. If the input dimension is 1, there is no difference between spherical and elliptical nodes, and the network selects Spherical by default.

Saving and Loading Weights

You can save the network's weights to a text file so you can load them later. For information on how to do this, see Saving and Loading Weights for the BPN object.

Making Values Permanent

Choose make permanent on the block to save the network's internal configuration and weights.

For information on how to use this block through the API, see Application Programmer's Interface

Ensemble Net (ENN)

The Ensemble Network, or ENN, is a feed-forward network. It has a specific architecture, which includes a set of submodels of Backpropagation Nets, or BPNs, and an output median calculator. The ENN provides accuracy and robustness, and especially useful for modeling multivariate functions.

The first layers of all BPNs and the input vector must be the same size. The last layers of all BPNs and the output vector must also be the same size. Before you can use the Ensemble Net object, you must train the network. The Ensemble Net must be trained initially, using NOL Studio, which is an offline, neural network modeling tool. For more information, see the *NeurOn-Line Studio User's Guide*.

Before you can pass data through an Ensemble Net the first time, you must initialize the ENN object by loading its settings and weights through file operation. When you pass a vector to an Ensemble Net, it passes the vector through all its submodels, and it uses the Median calculator to calculate the median values of the output from all submodels. The output vector of the Ensemble Net is composed of these median values.

Adjusting Weights

When you first create an ENN object, all its weights are set to zero, and its architecture is set to a null initial state. You must load its architecture settings and its weights from a text file exported from NOL Studio. You cannot change the architecture of an Ensemble Net; however, if you detect the network has large prediction errors, you can adjust the weights by retraining the network. You can call gnne-train-ensemble method to retrain the network.

Saving and Loading Weights

You can save the network's weights to a text file so you can load them later. For information on how to do this, see Saving and Loading Weights for the BPN object.

Making Values Permanent

Choose make permanent on the neural network object to save the network's internal configuration and weight so that resetting G2 has no effect on their values.

For information on how to use this block through the API, see Application Programmer's Interface

GNNE Predictive Model

The GNNE Predictive Model is a feed-forward network. It is a combination of an Ensemble Network and an input-output manager. The GNNE Predictive Model provides the easy management of the input and output data, and the Ensemble Network's accuracy and robustness.

The GNNE Predictive Model object cloned from the palette is an empty model without variable information or model parameters. Before you can pass data through the Predictive Model the first time, you must initialize the model object by loading its variable information and model parameters through file operation. You can also transfer that information through an online bridge when NOL Studio is connected to G2.

Adjusting Weights

When you first create a GNNE Predictive Model object, all its weights are set to zero, and its architecture is set to a null initial state. You must load its architecture settings and its weights from an XML file exported from NOL Studio. You cannot change the architecture of a GNNE Predictive Model; however, if you detect the network has large prediction errors, you can adjust the weights by retraining the network. See the *NeurOn-Line Studio User's Guide* for information on how to export predictive models in XML format.

Saving and Loading Weights

After you retrain the GNNE Predictive Model, you can save the model information to an XML file so you can load it later. For information on how to do this, see Saving and Loading Weights for the BPN object.

Making Values Permanent

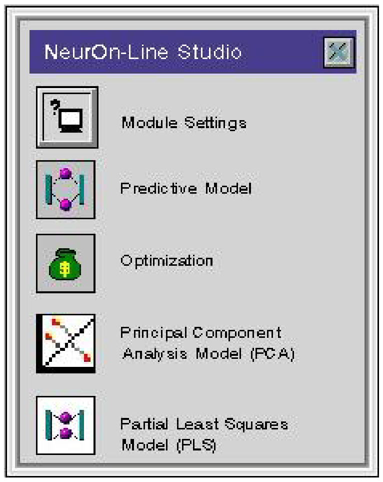
Choose make permanent on the neural network object to save the network's internal configuration and weight so that resetting G2 has no effect on their values.

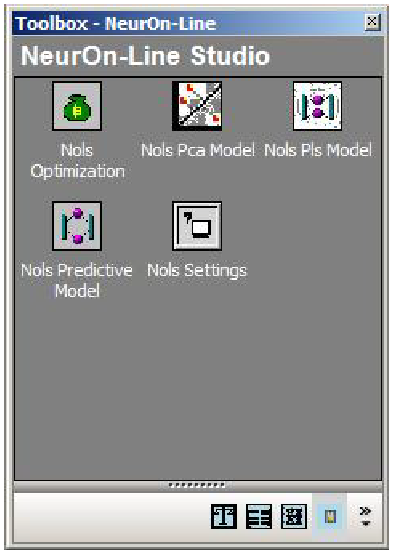
For information on how to use this block through the API, see Application Programmer's Interface.

NOL Studio Objects

The blocks in the NOL Studio palette are:

- Module Setting
- Predictive Model
- Optimization Mode
- Partial Least Square Model
- Principal Component Analysis Model


NeurOn-Line Studio
Module Settings
Predictive Model
Optimization
Principal Component Analysis Model (PCA)
Partial Least Squares Model (PLS)



Module Setting

The Module Settings for NOL Studio is an instance of nols-setting. The attributes of the nols-settings object are:

Attribute	Description
nols-studio-home-directory	The directory in which NOL Studio is installed, such as "C:\Program Files\Gensym\G2-2015\nolstudio".
nols-remote-process-listener-port	The port that the NolG2Gateway class uses, which is 22044, by default.
nols-connection-timeout	A connection timeout, which is 10 seconds, by default.
nols-interface-object-name	The name of the nols-studio-gateway object that provides communication, which is a subclass of gsi-interface. This object is created transiently by the launch procedures and referenced by the initialization procedures.
nols-execution-command	The name of execution file to launch the interface.
nols-host	The name of G2 host machine.
nols-remote-process-id	The ID return by the remote process.
others	Not used in this version.

Predictive Model

See “The Predictive Model and its API” in Chapter 13 Model Deployment of *NeurOn-Line Studio User’s Guide*.

Optimization Mode

See “The Optimization Model and its API” in Chapter 14 Optimization Deployment of *NeurOn-Line Studio User’s Guide*.

Partial Least Square Model

Partial Least Squares (PLS) is a statistical technique for building a linear regression model. The linear PLS model is simple and robust for correlating input variables.

To configure a PLS model:

1. Make sure G2 is running, then clone the PLS model onto a workspace in your application.

   Make sure the workspace does not belong to the nolstudio module.

2. Write procedures that use the PLS model API to make the model prediction.

Saving and Loading Parameters

You can load PLS parameters from a text file exported from NOL Studio, and you can import the parameters directly from a connected NOL Studio console. The steps for loading PLS model parameters are the same as for loading a neural network model. For information on how to do this, see Saving and Loading Weights for the neural network object.

Methods for PLS Model

Here are the methods you use for building a PLS model and running data through the model to get model predictions.

nols-load
(*model*: class nols-pls-model, *stream*: class g2-stream)

Loads the PLS parameter exported from NolStudio predictive model, which is trained as a linear model only.

nols-learn
(*model*: class nols-pls-model, *x*: class item-array, *y*: class item-array, *nfactor*: integer)

Builds the PLS model from data matrix *x* and *y* with a specified number of internal factors. The data matrix should be an item array of float arrays.

nols-rescaler-input-vector
(*model*: class nols-pls-model, *input-vector*: class float-array, *output-vector*: class float-array)

Scales the input data before feeding it into the PLS model. The PLS model stores the scale weights internally.

nols-rescaler-output-vector
(*model*: class nols-pls-model, *input-vector*: class float-array, *output-vector*: class float-array)

Scales the output data back to their normal range after PLS model execution. The PLS model stores the scale weights internally.

nols-execute
(*model*: class nols-pls-model, *x*: class float-array)
-> *value:* class float-array

Executes a PLS model with the given input data *x* and returns the output.

nols-execute
(*model*: class nols-pls-model, *x*: class item-array, *y*: class item-array)

Executes a PLS model with the given input data matrix *x*, where *y* provides the resulting matrix. The data matrix should be an item array of float arrays.

For details about the API, see Application Programmer's Interface

Principal Component Analysis Model

Principal components analysis (PCA) is a statistical technique applied to a set of variables to discover which sets of variables form coherent subsets that are relatively independent of one another. These subsets, principal components, are thought to be representative of the underlying processes that have created the correlations among variables. For this reason, PCA models are useful for analyzing data offline, as well as for online process monitoring.

To configure a PCA model:

1. Make sure G2 is running, then clone the PCA model onto a workspace in your application.

   Make sure the workspace does not belong to the nolstudio module.

2. Write procedures that use the PCA model API to do the PCA calculation.

Saving and Loading Parameters

You can load PCA parameters from a text file exported from NOL Studio, and you can also import the parameters directly from a connnected NOL Studio console. The steps for loading PCA model parameters are the same as for loading a neural network model. For information on how to do this, see Saving and Loading Weights for the neural network object.

Loading PCA Data

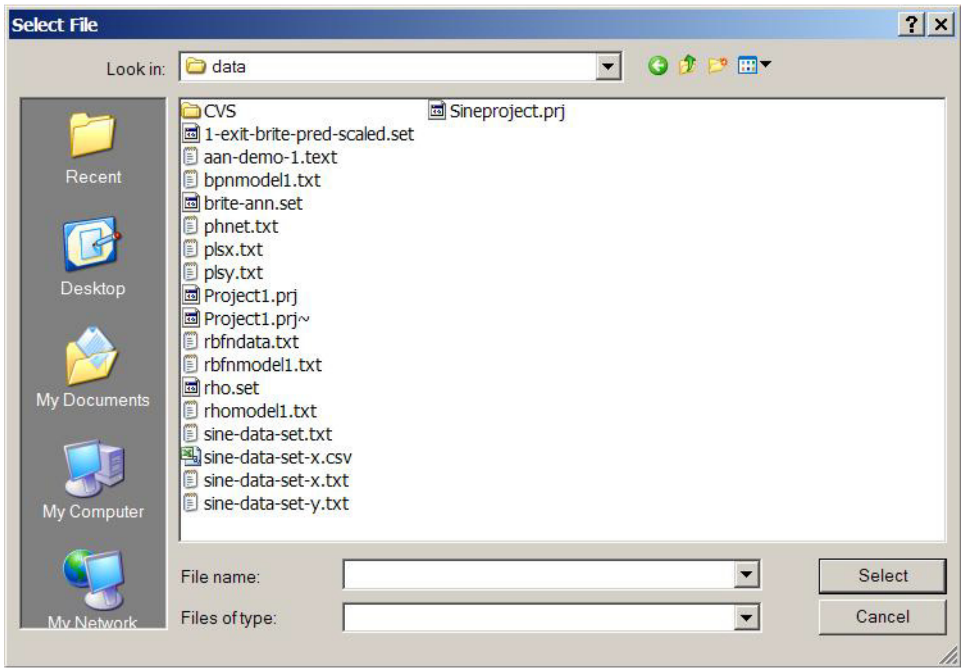
Principal component analysis models are a representation of the statistical characteristics of a data set. The principal components calculated from the model are useful to show these statistical properties, which can be used to monitor the process. Here the model data include the principal components and square prediction errors. These data are calculated in NOL Studio and are available in G2 through text file exchange or directly online data passing.

The squared prediction errors are the sum of errors between the real values, which are normally scaled, and the reconstructed values from PCA model. Mathematically, the error can be quantified by a matrix E (n x m), calculated as follows:

$$E = X - TP^t = X(I - PP^t)$$

For more information, see “What is PCA” in Chapter 3 “Visualizing Data” in the *NeurOn-Line Studio User’s Guide*.

To load PCA data from a file, in Telewindows, choose Load Data From File on the PCA object to display this dialog:

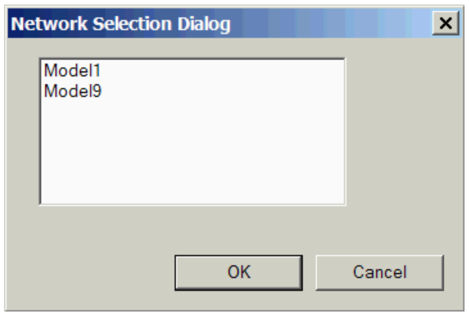


CVS
1-exit-brite-pred-scaled.set
aan-demo-1.text
bpnmodel1.txt
brite-ann.set
phnet.txt
plsx.txt
plsy.txt
Project1.prj
Project1.prj~
rbfndata.txt
rbfnmodel1.txt
rho.set
rhomodel1.txt
sine-data-set.txt
sine-data-set-x.csv
sine-data-set-x.txt
sine-data-set-y.txt
Sineproject.prj

Select the exported score file from the file system, then click Select to load.

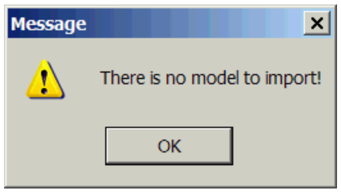
Importing Model Data from NOL Studio

In Telewindows, to import PCA data from a connected NOL Studio owned by that Telewindows, choose Import Data From NOL Studio on the PCA object to display this dialog:



Model1
Model9

The available model names corresponding to the PCA models in NOL Studio appear in the selection list. Select the model name and click OK to import the data into the PCA object in G2. If there is no available PCA model object in NOL Studio, the following dialog appears:


Message

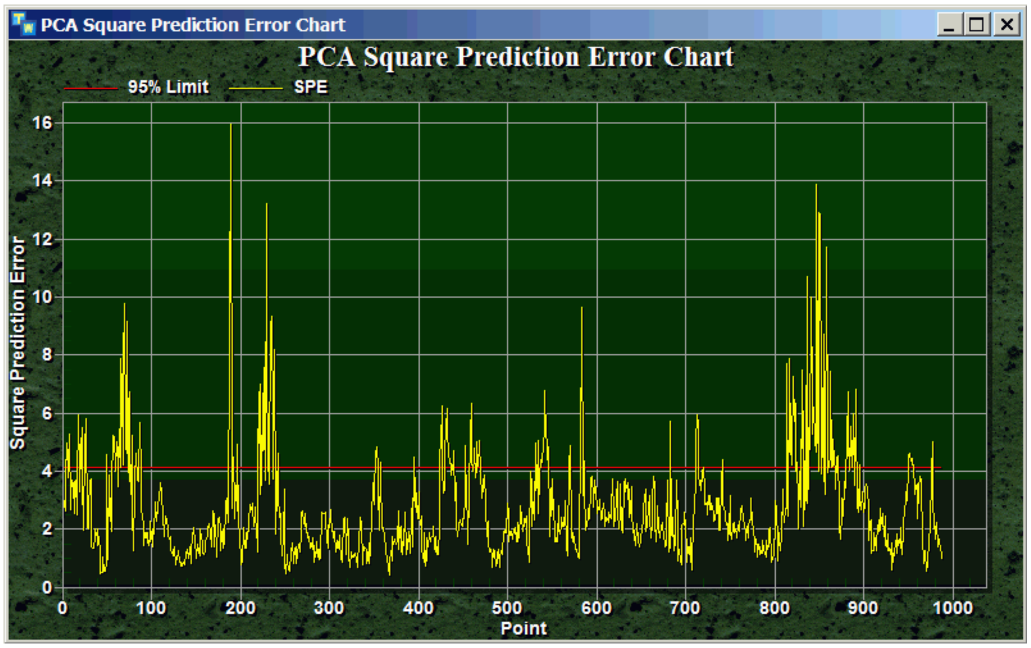
There is no model to import!

Displaying Statistical PCA Charts in G2

Statistical PCA charts are useful multivariate statistical control tools for monitoring the underling process. In Telewindows, you can create PCA monitoring charts based on variant control limits for the PCA model data. If the model data are imported into the PCA object in G2, you can view the behavior of the training data in these control charts. Statistical PCA charts include principal component (PC) charts and square prediction error (SPE) charts. The PC chart can be shown in single line chart, 2D chart, and 3D chart formats. Some chart examples are show below.

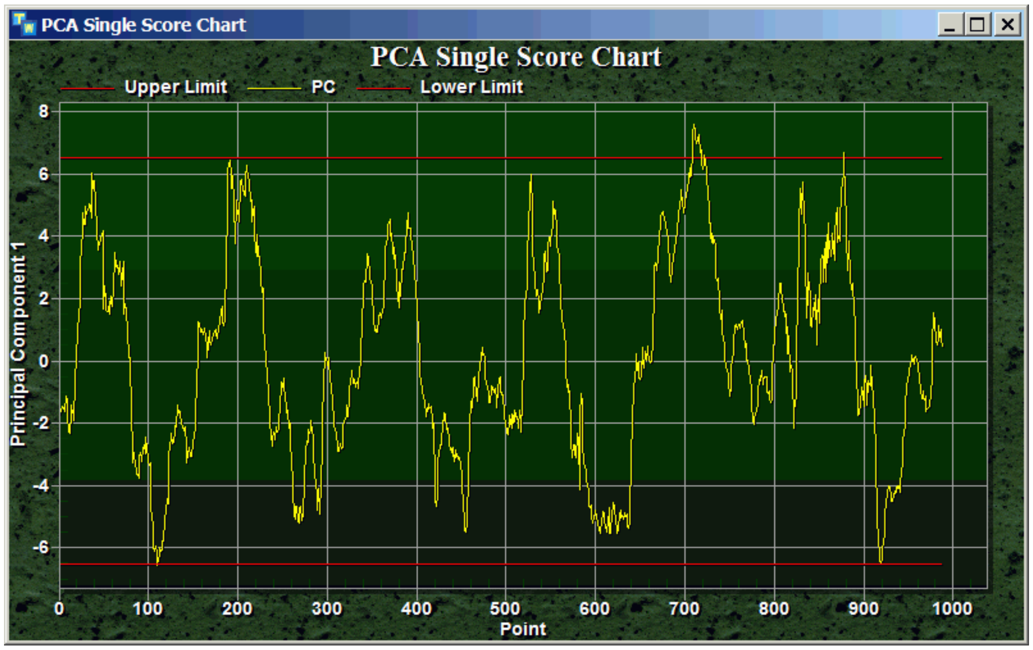
You can launch all types of charts through APIs for the PCA model. For details, see Application Programmer's Interface

Square Prediction Error Chart



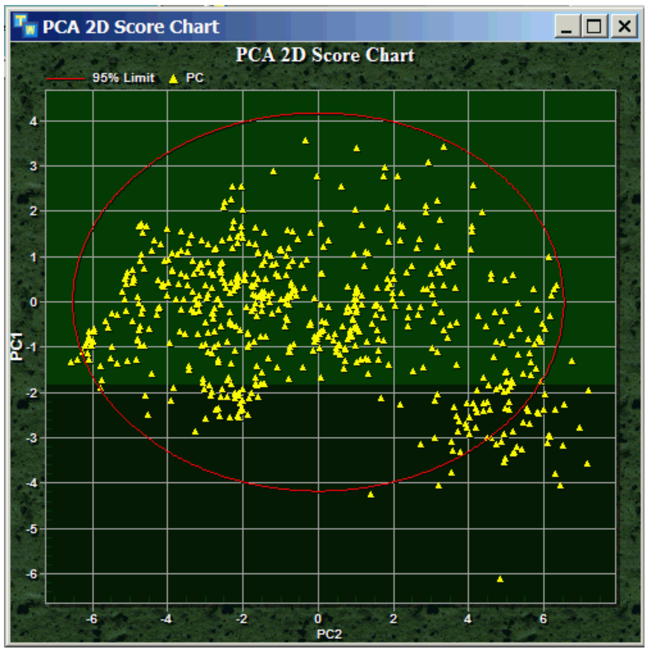
The yellow represents the error calculated from the training data, and red line is the 95% control limit, which is used to monitor the overall behavior of the process.

Single Principal Component Chart



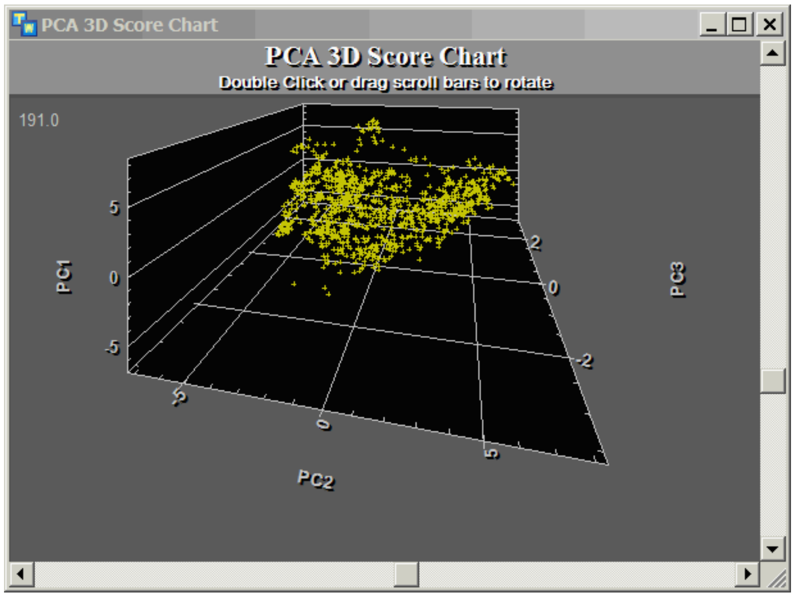
The single PC chart shows a single principal component variable changes over the sampling time defined in the training data. Because the input variables are all scaled around their centers, the PC variable varies around it zero mean. The red lines represent the 95% control limit for this PC and is useful for process monitoring.

Two Dimensional Principal Component Chart



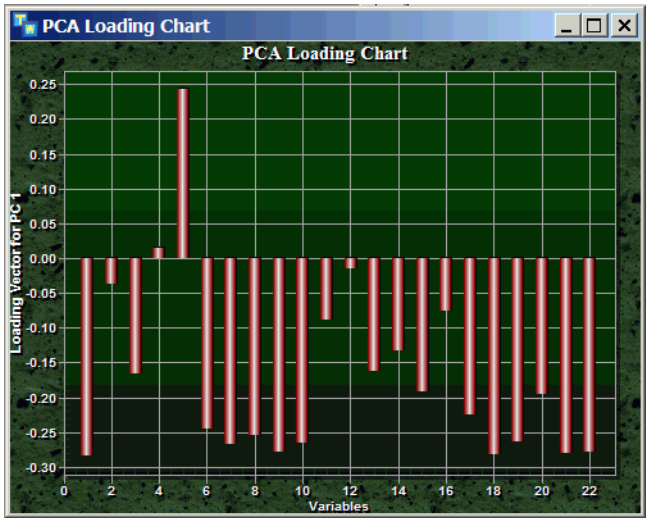
The 2D PC chart shows how two principal component variables from their training data vary against each other. The red line represent the 95% control limit for this surface. The 2D chart provides more information about the process.

Three Dimensional Principal Component Chart



The 3D PC chart shows how three principal component variables from their training data vary against each other. The 3D chart provides more information than the 2D chart.

PCA Loading Vector Chart



The loading vector chart shows the loading vector for a given principal component. The value on each variable shows the contribution of that variable to the given principal component.

Methods for PCA Model

These are the methods you use to build a PCA model and run data through the model.

nols-load

(*model*: class nols-pca-model, *stream*:class g2-stream)

Loads the PCA parameter exported from the NOLStudio projection chart.

nols-learn

(*model*: class nols-pca-model, *x*: class item-array)

Builds the PCA model from data matrix *x*, which is an item array of float arrays.

nols-rescaler-input-vector

(*model*: class nols-pca-model, *input-vector*: class float-array, *output-vector*: class float-array)

Scales the input data before feeding it into the PCA model. The PCA model stores the scale weights internally.

nols-execute

(*model*: class nols-pca-model, *x*: class float-array, *pcs*: class float-array)

Runs the scaled input data through the PCA model. *Pcs* provides the results of the calculation.

nols-execute

(*model*: class nols-pca-model, *x*: class float-array, *pcs*: class float-array, *nfactor*: integer)

Calculates first *nfactor* principal components for the scaled input. *Pcs* provides the results of the calculation.

For details, see Application Programmer's Interface

Application Programmer’s Interface

Describes GNNE’s application programmer’s interface (API).

Introduction

This chapter describes how to use the G2 Neural Network Engine Application Programmer’s Interface (API) and serves as a quick reference guide for application developers.

You can normally control GNNE objects from within a G2 procedure or function with GNNE’s application programmer’s interface (API). The API includes methods and procedures that perform all available actions for every GNNE object.

GNNE Objects

The APIs are organized by the target object.

Data Sets

gnne-append-data-set
(*blk*: class gnne-data-set , *input-dp*: class gnne-data-pair,
output-data: class gnne-data-path-value)

Inserts the data contained in the input data pair into a new row at the end of the Data Set object. The input data pair is not added to the data structure of the data set. The data is copied from the *input-dp* structure to the data set. The calling procedure must manage the data structure by deleting the input data pair after the call to the procedure has been made; otherwise, the procedure can leak items.

Parameter	Description
blk	The Data Set object to which data is to be added.
input-dp	The input data pair that contains the row of data to add to the data set.
output-data	An object that contains the number of rows in the data set.

gnne-make-data-set-permanent
(*blk*: class gnne-data-set)

Make the data set permanent.

Parameter	Description
blk	The Data Set object to make permanent.

gnne-remove-pair-from-data-set
(*blk*: class gnne-data-set, *index*: integer)

Removes a data pair from the data set, given the row index.

Parameter	Description
blk	The Data Set object from which to remove a data pair.
index	The row number of the data pair to remove.

gnne-read-data-set-from-file
(*blk*: class gnne-data-set, *file-name*: text)

Populates the contents of a Data Set object with the contents of a file. If the procedure named by the File Load Procedure attribute of the Data Set object exists, then that procedure is used to load the file. Otherwise, the file must be in the standard format created when the procedure saves a file.

Parameter	**Description**
blk	The Data Set object to populate.
file-name	The name of the file containing the data values, including a path appropriate for the file system type.

gnne-write-data-set-to-file
(*blk*: class gnne-data-set, *file-name*: text)

Stores the contents of a Data Set object to a file. The format of the output file can be customized by providing the name of a user-defined procedure in the File Load Procedure of the Data Set object. Otherwise, the procedure uses the standard file format.

Parameter	**Description**
blk	The Data Set object that contains the data to write.
file-name	The name of the file in which to save the data, including a path appropriate for the file system type.

gnne-configure-data-set
(*blk*: class gnne-data-set, *number-of-samples*: integer, *width-of-input*: integer, *width-of-output*: integer)

Configures a Data Set object to a given specification.

Parameter	**Description**
blk	The Data Set object to be configured.
number-of-samples	The number of rows for the data set.
width-of-input	The number of input variables for the data set.
width-of-output	The number of output variables for the data set.

gnne-clear-data-set

(*blk*: class gnne-data-set)

Clears transient values of a data set without affecting the permanent values.

Parameter	Description
blk	The Data Set object to be cleared.

gnne-read-data-set

(*input-ds*: class gnne-data-set, *line-pointer*: integer,
output-dp: class gnne-data-pair, *output-data*: class gnne-data-path-value)

Reads the contents of a data set and places the contents into an output data pair. The procedure maintains the line pointer during and after each procedure call.

Parameter	Description
input-ds	The Data Set object to read.
line-pointer	The index of the data set row for the procedure to read from.
output-dp	The output data pair into which the reader reads the data.
output-data	The line pointer after the procedure call.

gnne-random-divide-data-set

(*obj-list*: class item-list, *fract*: float, *clear-output*: symbol,
ds1: class gnne-data-set, *ds2*: class gnne-data-set)

Randomly copies all the data pairs from one or more data sets to two output data sets. This procedure is especially useful when you need to split data into two sets: one for training a neural network and the other for testing a neural network.

Parameter	Description
obj-list	The list of Data Set objects to read.
fract	The fract value for dividing the data sets, which is a float from 0 to 1.
clear-output	The flag for clearing the output at the end of the procedure call, which is the symbol yes or no.

Parameter	Description
ds1	The first Data Set object into which the data is copied.
ds2	The second Data Set object into which the data is copied.

gnne-copy-data-sets
(*obj-list1*: class item-list , *obj-list2*: class item-list, *clear-output*: symbo)

Copies all of the data sets in the first object list to each of the data sets in the second object list.

Parameter	Description
obj-list1	The list of Data Set objects to read.
obj-list2	The list of Data Set objects into which the data is copied.
clear-output	Whether to clear the output at the end of the procedure call, which is the symbol yes or no.

gnne-rescale-data-sets
(*blk*: class gnne-data-set-rescaler,
obj-list1: class item-list, *obj-list2*: class item-list)

Scales the data from all of the data sets in the first object list into each of the data sets in the second object list, using a Data Set Rescaler object.

Parameter	Description
blk	The Data Set Rescaler object used to read data from the data sets.
obj-list1	The list of Data Set objects containing the data to be scaled.
obj-list2	The list of Data Set objects into which the scaled data is copied.

gnne-execute-maximum-age-filter
(*max-age*: quantity, *ds*: class gnne-data-set, *obj-list*: class item-list,
output-data: class gnne-data-path-value)

Removes any data pair whose age is greater than a specified limit, and archives the removed data pairs to a list of data sets.

Parameter	Description
max-age	The maximum age limit for each sample to be filtered.
ds	The Data Set object to be filtered.
obj-list	The list of Data Set objects used to archive the filtered data pairs.
output-data	The data set size after the procedure call.

gnne-execute-size-limitation-filter
(*max-size*: integer, *ds*: class gnne-data-set, *obj-list*: class item-list, *output-data*: class gnne-data-path-value)

Limits the number of data pairs stored in the data set. If the data set contains more data pairs than the maximum specified, the filter removes enough data pairs from the top of the data set to keep the size at the maximum, and it archives the removed data pairs to the data sets in the given list.

Parameter	Description
max-size	The maximum size limit for each sample to be filtered.
ds	The Data Set object to be filtered.
obj-list	The list of Data Set objects used to archive the filtered data pairs.
output-data	The data set size after the procedure call.

gnne-execute-novelty-filter
(*filter*: class gnne-novelty-filter, *ds*: class gnne-data-set, *obj-list*: class item-list, *output-data*: class gnne-data-path-value)
-> *truth-value*

Filters the data set, using a Novelty Filter object. You configure the filtering algorithm in the object. The calling procedure must manage the data structures by deleting the object list and output data after the call to the procedure has been made; otherwise, the procedure can leak items.

Parameter	Description
filter	The Novelty Filter object that filters the data.
ds	The primary Data Set object to be filtered. In a diagram, this would be the data set to which the capability link on the Novelty Filter object would attach.

Parameter	Description
obj-list	The list of Data Set objects in which the removed data points are stored.
output-data	The new number of rows in the data set.

Return Value	Description
truth-value	Returns true if the new data point is none.

Vectors and Matrices

gnne-read-array

(*stream*: class g2-stream, *number-of-values*: integer, *destination-array*: class g2-array, *delimiter*: text)

Reads a line of values, separated by delimiters. This procedure can read integer, float, quantity, symbol, text, and truth-value arrays.

Parameter	Description
stream	The G2 file stream to read.
number-of-values	The number of values to read.
destination-array	The G2 array in which to store the values.
delimiter	The delimiter string.

gnne-write-array

(*stream*: class g2-stream, *array*: class g2-array, *delimiter*: text, *comment*: text)

Writes a line of values, separated by delimiters. This procedure can write integer, float, quantity, symbol, text, and truth-value arrays.

Parameter	Description
stream	The G2 file stream to write.
array	The G2 array from which to write the values.
delimiter	The delimiter string.
comment	Comment text to write with the file stream.

gnne-read-matrix

(*stream*: class g2-stream, *x*: class gnne-a-matrix, *rows*: integer, *cols*: integer, *delimiter*: text)

Reads two-dimensional arrays of values, separated by delimiters, into a matrix. The matrix is redimensioned, if necessary.

Parameter	Description
stream	The G2 file stream to read.
x	The matrix in which to store the values.
rows	The number of rows for the two-dimensional array.
cols	The number of columns for the two-dimensional array.
delimiter	The delimiter string.

gnne-write-matrix

(*stream*: class g2-stream, *x*: class gnne-a-matrix, *delimiter*: text, *comment*: text)

Writes a matrix into a G2 file stream, separated by delimiters.

Parameter	Description
stream	The G2 file stream to write.
x	The matrix from which to write the values.
delimiter	The delimiter string.
comment	Comment text to write with the matrix.

gnne-execute-vector-rescaler

(*blk*: class gnne-vector-rescaler, *input-vector*: class gnne-vector-path-value, *output-vector*: class gnne-vector-path-value)

Rescales the elements of the input vector by applying additive and multiplicative factors to each element, using a Vector Rescaler object.

Parameter	Description
blk	The Vector Rescaler object used to process the vectors. The object contains scale factors for each element.
input-vector	The input vector path value passed into the object.
output-vector	The output vector path value produced by the object.

gnne-convert-vector-to-data-pair

(*input-vector1*: class gnne-vector-path-value,
input-vector2: class gnne-vector-path-value,
output-dp: class gnne-data-pair, *check-concurrency*: truth-value)

Converts two vectors into a data pair. The first vector is the X in the data pair, and the second vector is the Y. The *check-concurrency* flag indicates whether the input vectors have a later timestamp than the data pair. If this flag is true, the data pair is only valid when both vectors have later timestamps.

Parameter	**Description**
input-vector1	The input vector path value passed into the data pair as X.
input-vector2	The input vector path value passed into the data pair as Y.
output-dp	The output data pair.
check-concurrency	Whether to check the timestamps of both vectors and the data pair.

Neural Networks

gnne-execute-bpn

(*net*: class bpn, *x*: class gnne-vector-path-value,
y: class gnne-vector-path-value)

Executes a Backpropagation Network object with an input X vector, and pushes the output values into an output Y vector.

Parameter	**Description**
net	The Backpropagation Network object to be executed.
x	The input vector X for the network.
y	The output vector Y from the network.

gnne-configure-bpn

(*net*: class bpn, *layer-sizes*: class integer-array, *transfer-functions*: class integer-array)

Configures a Backpropagation Network object with a given specification.

Parameter	Description
net	The Backpropagation Network object to be configured.
layer-sizes	The integer array that contains the size parameter of each layer.
transfer-functions	The integer array that contains the flag of transfer functions for each layer.

gnne-clear-bpn

(*net*: class bpn)

Clears a Backpropagation Network object of its weights.

Parameter	Description
net	The Backpropagation Network object to be cleared.

gnne-write-bpn-to-file

(*net*: class bpn, *stream*: class g2-stream)

Saves parameters of a Backpropagation Network object to a file stream.

Parameter	Description
net	The Backpropagation Network object whose parameters are to be saved.
stream	The G2 file stream into which the parameters are saved.

gnne-read-bpn-from-file

(*net*: class bpn, *stream*: class g2-stream)

Reads parameters of a Backpropagation Network object from a file stream.

Parameter	Description
net	The Backpropagation Network object whose parameters are to be read.
stream	The G2 file stream from which the parameters are read.

gnne-execute-rbfn

(*net*: class rbfn, *x*: class gnne-vector-path-value, *y*: class gnne-vector-path-value, *maximum-activation-path*: class gnne-data-path-value)

Executes a Radial Basis Function Network object with an input X vector, and pushes the output values into an output Y vector. The Maximum Activation attribute of the object indicates the performance of the inner layer.

Parameter	**Description**
net	The Radial Basis Function Network object to be executed.
x	The input vector X for the network.
y	The output vector Y from the network.
maximum-activation-path	The maximum hidden node activation value.

gnne-configure-rbfn

(*net*: class rbfn, *inputs*: integer, *hidden*: integer, *outputs*: integer, *overlap*: float, *unit-type*: symbol)

Configures a Radial Basis Function Network object with a given specification.

Parameter	**Description**
net	The Radial Basis Function Network object to be configured.
inputs	The number of inputs.
hidden	The size of the hidden layer.
outputs	The number of outputs.
overlap	The type of overlap.
unit-type	The unit type.

gnne-write-rbfn-to-file
(*net*: class rbfn, *stream*: class g2-stream)

Saves the specification and weights of a Radial Basis Function Network object into a file stream.

Parameter	Description
net	The Radial Basis Function Network object to be saved.
stream	The G2 file stream into which the data is saved.

gnne-read-rbfn-from-file
(*net*: class rbfn, *stream*: class g2-stream)

Reads the specification and weights of a Radial Basis Function Network object from a file stream.

Parameter	Description
net	The Radial Basis Function Network object whose data is to be read.
stream	The G2 file stream to read.

gnne-execute-rho-net
(*net*: class rho-net, *x*: class gnne-vector-path-value,
y: class gnne-vector-path-value)

Executes a Rho Network object with an input X vector, and pushes the output values into an output vector Y.

Parameter	Description
net	The Rho Network object to be executed.
x	The input vector X for the network.
y	The output vector Y from the network.

gnne-execute-ensemble-model
(*net*: class ensemble-network , *x*: class gnne-vector-path-value,
y: class gnne-vector-path-value)

Executes an Ensemble Network object with an input X vector, and pushes the output values into an output vector Y.

Parameter	**Description**
net	The Ensemble Network object to be executed.
x	The input vector X for the network.
y	The output vector Y from the network.

gnne-read-ensemble-model-from-file
(*net*: class ensemble-network, *stream*: class g2-stream)

Reads the specification of an Ensemble Network object from a file stream.

Parameter	**Description**
net	The Ensemble Network object whose data is to be read.
stream	The G2 file stream to read.

gnne-write-ensemble-model-to-file
(*net*: class ensemble-network, *stream*: class g2-stream)

Saves the specification of an Ensemble Network object into a file stream.

Parameter	**Description**
net	The Ensemble Network object to be saved.
stream	The G2 file stream into which the data is saved.

gnne-execute-autoassociative-net
(*net*: class autoassociative-net, *input-vector*: class gnne-vector-path-value,
output-vector: class gnne-vector-path-value)

Executes an Autoassociative Network object with an input X vector, and pushes the output values into an output vector Y.

Parameter	**Description**
net	The Autoassociative Network object to be executed.

Parameter	Description
input-vector	The input vector X for the network.
output-vector	The output vector Y from the network.

gnne-configure-autoassociative-net
(*net*: class autoassociative-net, *layer-sizes*: class integer-array, *transfer-functions*: class integer-array, *run*: symbol)

Configures a Autoassociative Network object with a given specification.

Parameter	Description
net	The Autoassociative Network object to be configured.
layer-sizes	The integer array that contains the size parameter of each layer.
transfer-functions	The integer array that contains the flag of transfer functions for each layer.
run	A symbol that indicates the running mode for the Autoassociative Network. It can be either filter-only or correct-gross-errors.

gnne-write-autoassociative-net-to-file
(*net*: class autoassociative-net, *stream*: class g2-stream)

Saves the specification of an Autoassociative Network object into a file stream.

Parameter	Description
net	The Autoassociative Network object to be saved.
stream	The G2 file stream into which the data is saved.

gnne-read-autoassociative-net-from-file
(*net*: class autoassociative-net, *stream*: class g2-stream)

Reads the specification of an Autoassociative Network object from a file stream.

Parameter	Description
net	The Ensemble Network object whose data is to be read.
stream	The G2 file stream.

Network Training

gnne-train-neural-network
(*blk*: class gnne-trainer, *net*: class neural-network,
ds: class gnne-data-set, *output-data*: class gnne-data-path-value)

Uses a Trainer object to train a neural network, using a data set.

Parameter	Description
blk	The Trainer object to be executed.
net	The network object to be trained.
ds	The data set to use for training.
output-data	The RMSE error value after the network has been trained.

gnne-execute-fit-tester
(*fit-metric-text*: text, class gnne-fit-tester, *net*: class neural-network,
ds: class gnne-data-set, *output-data*: class gnne-data-path-value)

Tests a neural network, using a data set.

Parameter	Description
fit-metric-test	A text that determines the type of test. The options are: “rmse”, “class”, and “prob”.
net	The network object to be tested.
ds	A data set containing the inputs and associated output data against which the neural net fitness is to be tested. Predictions are placed in the prediction columns of this data set.
output-data	A data path value into which the result of the fitness test is placed.

gnne-execute-sensitivity-tester

(*net*: class neural-network, *ds*: class gnne-data-set, *sensitivity-matrix*: class gnne-a-matrix)

Calculates the sensitivity of a neural network, given a data set.

Parameter	Description
net	The network object to be tested.
ds	The data set to be tested.
sensitivity-matrix	A matrix that holds the sensitivity values.

Statistical Models

nols-load

(*model*: class nols-pls-model, *stream*: class g2-stream)

Loads the PLS parameter exported from text file exported from NOL Studio.

Parameter	Description
model	The PLS model whose parameters are loaded.
stream	The g2-stream from the text file.

nols-learn

(*model*: class nols-pls-model, *x*: class item-array, *y*: class item-array, *nfactor*: integer)

Builds the PLS model from data matrix *x* and *y* with a specified number of internal factors. The data matrix should be an item array of float arrays. This method requires the NOL Studio remote process and statistical calculator.

Parameter	Description
model	The PLS model whose parameters are loaded.
x	The input matrix. The class of the element in the item array is float-array.
y	The input matrix. The class of the element in the item array is float-array. The x and y should have the same array-length
nfactor	The number of factor on hidden layer.

nols-rescaler-input-vector
(*model*: class nols-pls-model, *input-vector*: class float-array, *output-vector*: class float-array)

Scales the input data before feeding it into the PLS model. The PLS model stores the scale weights internally. The method is called before model evaluation.

Parameter	Description
model	The PLS model which provide the scale weights.
input-vector	The raw vector.
output-vector	The scaled vector.

nols-rescaler-output-vector
(*model*: class nols-pls-model, *input-vector*: class float-array, *output-vector*: class float-array)

Scales the output data back to their normal range after PLS model execution. The PLS model stores the scale weights internally.

Parameter	Description
model	The PLS model which provide the scale weights.
input-vector	The raw vector.
output-vector	The scaled vector.

nols-execute
(*model*: class nols-pls-model, *x*: class float-array)
-> *value:* class float-array

Executes a PLS model with the given input data *x* and returns the output.

Parameter	Description
model	The PLS model to be evaluated.
x	The scaled input vector.

Return Value	Description
value	Returns output vector. The value need to be rescaled back to their normal range.

nols-execute

(*model*: class nols-pls-model, *x*: class item-array, *y*: class item-array)

Executes a PLS model with the given input data matrix *x*, where *y* provides the resulting matrix. The data matrix should be an item array of float arrays.

Parameter	Description
model	The PLS model to be evaluated.
x	The scaled input matrix.
y	The scaled output matrix.

nols-load

(*model*: class nols-pca-model, *stream*: class g2-stream)

Loads the PCA parameter exported from the NOL Studio projection chart.

Parameter	Description
model	The PCA model whose parameters are loaded.
stream	The g2-stream from the text file.

nols-rescaler-input-vector

(*model*: class nols-pca-model, *input-vector*: class float-array, *output-vector*: class float-array)

Scales the input data before feeding it into the PCA model. The PCA model stores the scale weights internally. The method is called before model evaluation.

Parameter	Description
model	The PCA model which provide the scale weights.
input-vector	The raw vector.
output-vector	The scaled vector.

nols-execute
(*model*: class nols-pca-model, *x*: class float-array, *pcs*: class float-array)

Runs the scaled input data through the PCA model. *Pcs* provides the results of the calculation.

Parameter	Description
model	The PCA model to be evaluated.
x	The scaled input vector.
pcs	The calculated principal components. The number of components depends on the default number of component value stored in PCA model.

nols-execute
(*model*: class nols-pca-model, *x*: class float-array, *pcs*: class float-array, *nfactor*: integer)

Calculates first *nfactor* principal components for the scaled input. *Pcs* provides the results of the calculation.

Parameter	Description
model	The PCA model to be evaluated.
x	The scaled input vector.
pcs	The calculated principal components.
nfactor	The number of components.

nols-show-pca-spe-chart
(*model*: class nols-pca-model, *win*: class g2-window)
-> *handle*: integer

Shows the squared prediction error chart in given window. This procedure can only be called if the model data is loaded in this model.

Parameter	Description
model	The PCA model.
win	The window object in which the chart will show

Return Value	**Description**
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-show-pca-single-pc-chart
(*model*: class nols-pca-model, *pcIndex*: integer, *win*: class g2-window)
-> *handle*: integer

Shows the single principal component in the given window. This procedure can only be called if the model data is loaded in this model.

Parameter	**Description**
model	The PCA model.
pcIndex	The index of the principal component.
win	The window object in which the chart will show

Return Value	**Description**
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-show-pca-2d-pc-chart
(*model*: class nols-pca-model, *pc1Index*: integer, *pc2Index*: integer,
win: class g2-window)
-> *handle*: integer

Shows the two dimensional principal component chart in the given window. This procedure can only be called if the model data is loaded in this model.

Parameter	**Description**
model	The PCA model.
pc1Index	The index of the first principal component.
pc2Index	The index of the second principal component.
win	The window object in which the chart will show

Return Value	**Description**
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-show-pca-3d-pc-chart
(*model*: class nols-pca-model, *pc1Index*: integer, *pc2Index*: integer, *pc3Index*: integer, *win*: class g2-window)
-> *handle*: integer

Shows the three dimensional principal component chart in the given window. This procedure can only be called if the model data is loaded in this model.

Parameter	**Description**
model	The PCA model.
pc1Index	The index of the first principal component.
pc2Index	The index of the second principal component.
pc3Index	The index of the third principal component.
win	The window object in which the chart will show

Return Value	**Description**
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-show-pca-loading-chart
(*model*: class nols-pca-model, *pcIndex*: integer, *win*: class g2-window)
-> *handle*: integer

Shows the loading vector for given principal component in the given window. This procedure can only be called if the model data is loaded in this model.

Parameter	**Description**
model	The PCA model.
pcIndex	The index of the principal component, whose loading vector will be shown.
win	The window object in which the chart will show

Return Value	**Description**
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-create-pca-spe-monitoring-chart
(*model*: class nols-pca-model, *nshow*: integer, *win*: class g2-window)
-> *handle*: integer

Creates the squared prediction error monitoring chart based on the SPE statistics in given window. This procedure can be called without the model data loaded in this model.

Parameter	**Description**
model	The PCA model.
nshow	The number of data point shown in the chart.
win	The window object in which the chart will show

Return Value	**Description**
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-create-pca-pc-monitoring-chart
(*model*: class nols-pca-model, *pcIndex*: integer, *nshow*: integer, *win*: class g2-window)
-> *handle*: integer

Creates the single principal component chart based on the PC statistics in the given window. This procedure can be called without the model data loaded in this model.

Parameter	**Description**
model	The PCA model.
pcIndex	The index of the principal component.
nshow	The number of data point shown in the chart.
win	The window object in which the chart will show

Return Value	**Description**
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-create-pca-2d-pc-monitoring-chart
(*model*: class nols-pca-model, *pc1Index*: integer, *pc2Index*: integer, *nshow*: integer, *win*: class g2-window)
-> *handle*: integer

Creates the two dimensional principal component chart based on the statistics of these two PCs in the given window. This procedure can be called without the model data loaded in this model.

Parameter	Description
model	The PCA model.
pc1Index	The index of the first principal component.
pc2Index	The index of the second principal component.
nshow	The number of data point shown in the chart.
win	The window object in which the chart will show

Return Value	Description
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-create-pca-3d-pc-monitoring-chart
(*model*: class nols-pca-model, *pc1Index*: integer, *pc2Index*: integer, *pc3Index*: integer, *nshow*: integer, *win*: class g2-window)
-> *handle*: integer

Creates the three dimensional principal component chart based on the statistics of these three PCs in the given window. This procedure can be called without the model data loaded in this model.

Parameter	Description
model	The PCA model.
pc1Index	The index of the first principal component.
pc2Index	The index of the second principal component.
pc3Index	The index of the third principal component.
nshow	The number of data point shown in the chart.
win	The window object in which the chart will show

Return Value	Description
handle	Returns the chart handle. This handle can be used to refer to the chart.

nols-show-pca-pc-point-in-chart
(*model*: class nols-pca-model, *chart*: integer, *n*: integer, *input-vector*: class float-array, *pcIndex*: integer, *win*: class g2-window)

Shows a single point in the single PC monitoring chart. The PC point is calculated from the given input vector. The chart handle points to the chart where the point will be added. This procedure is called after the monitoring chart has been created.

Parameter	Description
model	The PCA model.
chart	The chart handle point to the chart where the point will be added.
n	The index of the data point in the chart.
input-vector	The input vector used to calculate the PC point. This should be unscaled real data.
pcIndex	The index of the principal component.
win	The window object in which the chart shows

nols-show-pca-pc-point-in-2d-chart
(*model*: class nols-pca-model, *chart*: integer, *n*: integer, *input-vector*: class float-array, *pc1Index*: integer, *pc2Index*: integer, *win*: class g2-window)

Shows a single point in the two dimensional PC monitoring chart. The PC point is calculated from the given input vector. The chart handle points to the chart where the point will be added. This procedure is called after the monitoring chart has been created.

Parameter	Description
model	The PCA model.
chart	The chart handle point to the chart where the point will be added.
n	The index of the data point in the chart.

Parameter	Description
input-vector	The input vector used to calculate the PC point. This should be unscaled real data.
pc1Index	The index of the first principal component.
pc2Index	The index of the second principal component.
win	The window object in which the chart shows

nols-show-pca-pc-point-in-3d-chart
(*model*: class nols-pca-model, *chart*: integer, *n*: integer, *input-vector*: class float-array, *pc1Index*: integer, *pc2Index*: integer, *pc3Index*: integer,*win*: class g2-window)

Shows a single point in the three dimensional PC monitoring chart. The PC point is calculated from the given input vector. The chart handle points to the chart where the point will be added. This procedure is called after the monitoring chart has been created.

Parameter	Description
model	The PCA model.
chart	The chart handle point to the chart where the point will be added.
n	The index of the data point in the chart.
input-vector	The input vector used to calculate the PC point. This should be unscaled real data.
pc1Index	The index of the first principal component.
pc2Index	The index of the second principal component.
pc3Index	The index of the third principal component.
win	The window object in which the chart shows

Interaction with NOL Studio

Remote Process Management

nols-launch-remote-process
( )
-> <u>*id*</u>: float

Uses the default nols-settings to launch the remote process for calculating NOL Studio models.

Return Value	Description
<u>*id*</u>	The process ID for the started remote process.

nols-launch-remote-process-by-settings
(*settings*: class nols-setting)
-> <u>*id*</u>: float

Uses the specified nols-settings object to launch the remote process for calculating NOL Studio models.

Parameter	Description
setting	The nols-settings object that provides information such as host name and listening port for launching remote process.

Return Value	Description
<u>*id*</u>	The process ID for the started remote process.

nols-launch-remote-process-with-message
(*settings*: class nols-setting, *client*: class ui-client-item)
-> <u>*id*</u>: float

Uses the specified nols-settings to launch the remote process for calculating NOL Studio models. If it is called within Telewindows, a message dialog shows the launching status.

Parameter	Description
setting	The nols-settings object that provides information such as host name and listening port for launching remote process.
client	The window object used for launching message dialog.

Return Value	Description
<u>*id*</u>	The process ID for the started remote process.

nols-kill-remote-process
( )

Kills the remote process used for calculating NOL Studio models.

nols-launch-nolstudio-by-setting
(settings: class nols-setting, client:class ui-client-item)
-> <u>*id*</u>: float

Uses the specified nols-settings to launch the remote process for calculating NOL Studio models. If it is called within Telewindows, a message dialog shows the launching status.

Parameter	Description
setting	The nols-settings object that provides information such as host name and listening port for launching remote process.
client	The window object provided as an owner for this NOL Studio console and used for launching message dialog.

Return Value	Description
<u>*id*</u>	The process ID for the started remote process.

nols-kill-remote-studio
(*client*: class ui-client-item)
-> *id*: float

Kills the remote NOL Studio console owned by the specified window object.

Parameter	Description
client	The window object provided as an owner for this NOL Studio console and used for launching message dialog.

Return Value	Description
id	The process ID for the NOL Studio process.

Data Management

gnne-data-set

gnne-data-set::gnne-export-to-studio
(*ds*: class gnne-data-set, *win*: class g2-window)

Exports the data set object into NOL Studio owned by the window object as a raw data series. The variable name is created automatically for each variable inside NOL Studio.

Parameter	Description
ds	The data matrix to be passed into NOL Studio. The input and output of the data set are merged together with output appended at the end of input variables.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

NOL Studio Data Objects

nols-load-project

(*pjDir*: text, *pjName*: text, *win*: class g2-window)

Loads a project into NOL Studio owned by the window object. The project is specified by the project name and path.

Parameter	Description
pjDir	The directory name where the project file is located as a text string.
pjName	The file name of the project file.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-export-dataseries

(*dsName*: text, *X*: class item-array, *win*: class g2-window)

Exports a matrix into NOL Studio owned by the window object as a raw data series. The variable name is created automatically for each variable inside NOL Studio.

Parameter	Description
dsName	The name of the data series in the NOL Studio console as a text string.
X	An item-array with each item as an equal length float-array, This item-array is served as the data matrix, where each float-array is a row vector.
win	Window object provided as an owner for this NOL Studio console, and used for launching any message dialog.

nols-delete-dataseries
(*dsName*: text, *win*: class g2-window)

Deletes a raw data series in NOL Studio owned by the window object. The process fails if the raw data has been preprocessed.

Parameter	Description
dsName	The name of the data series in NOL Studio console as a text string.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-get-raw-dataseries-names
(*win*: class g2-window)
-> *data-series*: sequence

Returns a sequence of text strings, which are the names of raw data series in the NOL Studio owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
data-series	A sequence of text strings, which are names of raw data series in the NOL Studio console.

nols-get-processed-dataseries-names
(*processName*: symbol, *win*: class g2-window)
-> *data-series*: sequence

Returns a sequence of text strings, which are the names of data series preprocessed by the preprocessor with given name in the NOL Studio owned by the window object.

Parameter	Description
processName	Provides the text string as the name of the preprocessor in NOL Studio console.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
data-series	A sequence of text strings, which are names of preprocessed data series in the NOL Studio console. The data series are preprocessed by the preprocessor with given name.

nols-show-data-in-line-chart
(*X*: class item-array, *win*: class g2-window)

Passes the data matrix into NOL Studio as a raw data series and shows all variables in that data series in a line chart. The NOL Studio is owned by the window object.

Parameter	Description
X	An item-array with each item as an equal length float-array, This item-array is served as the data matrix, where each float-array is a row vector.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-show-single-line-chart
(*X*: class float-array, *win*: class g2-window)

Passes the data array into NOL Studio as a single variable raw data series and show this variable in a line chart. The NOL Studio is owned by the window object.

Parameter	Description
X	A float-array is served as a single column of a data matrix. When passed into NOL Studio, the float-array is a single variable data series.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-show-data-in-projection-chart
(*X*: class item-array, *win*: class g2-window)

Passes the data matrix into NOL Studio as a raw data series and shows all variables in a projection chart. The NOL Studio is owned by the window object.

Parameter	Description
X	An item-array with each item as an equal length float-array, This item-array serves as the data matrix, where each float-array is a row vector.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-show-data-in-xy-chart
(*X*: class float-array, *Y*: class float-array, *win*: class g2-window)

Passes the two data arrays into NOL Studio, merges them as a raw data series, and shows one variable against the other in a X-Y chart. The NOL Studio is owned by the window object.

Parameter	Description
X	A float-array is served as a single column of a data matrix. When passed into NOL Studio, the float-array is the first variable of a two-variable data series.
Y	A float-array is served as a single column of a data matrix. When passed into NOL Studio, the float-array is the second variable of a two-variable data series.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

gnne-show-inputs-in-line-chart
(*ds*: class gnne-data-set, *win*: class g2-window)

Passes the data matrix of the input-data-set of a gnne-data-set into NOL Studio as a raw data series and shows all variables in that data series in a line chart. The NOL Studio is owned by the window object.

Parameter	Description
ds	The gnne-data-set whose input data set will be exported into NOL Studio and shown in a line chart.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

gnne-show-outputs-in-line-chart
(*ds*: class gnne-data-set, *win*:class g2-window)

Passes the data matrix of the output data set of a gnne-data-set into NOL Studio as a raw data series and show all variables in that data series in a line chart. The NOL Studio is owned by the window object.

Parameter	Description
ds	The gnne-data-set whose output data set will be exported into NOL Studio and shown in a line chart.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

gnne-show-input-variable-in-line-chart
(*ds*: class gnne-data-set, *index*: integer, *win*: class g2-window)

Passes one of the variables in the input data set of a gnne-data-set into NOL Studio as a single variable raw data series and shows that variable in a line chart. The *index* of the variable in the input data set is given, and the NOL Studio is owned by the window object.

Parameter	Description
ds	The gnne-data-set where one of the variables in input data set will be exported into NOL Studio and shown in a line chart.
index	The index of the variable in the input data set.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

gnne-show-output-variable-in-line-chart
(*ds*: class gnne-data-set, *index*: integer, *win*: class g2-window)

Passes one of the variables in the output data set of a gnne-data-set into NOL Studio as a single variable raw data series and shows that variable in a line chart. The *index* of the variable in the input data set is given, and the NOL Studio is owned by the window object.

Parameter	Description
ds	The gnne-data-set where one of the variables in output data set will be exported into NOL Studio and shown in a line chart.
index	The index of the variable in the output data set.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

gnne-show-inputs-in-projection-chart
(*ds*: class gnne-data-set, *win*: class g2-window)

Passes the data matrix of the input data set of a gnne-data-set into NOL Studio as a raw data series and shows all variables in a projection chart. The NOL Studio is owned by the window object.

Parameter	Description
ds	The gnne-data-set whose input data set will be exported into NOL Studio and shown in a projection chart.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Neural Networks

Predictive Model

nols-get-predictive-model-names
(*win*: class g2-window)
-> *models*: sequence

Returns a sequence that contains the name of predictive models in NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
models	A sequence of text strings, which are names of predictive models in the NOL Studio console.

nols-get-predictive-model
(*model*: class nols-predictive-model, *modelName*: text, *win*: class g2-window)

Loads the model parameters from a predictive model in NOL Studio into the nols-predictive-model object. The NOL Studio is owned by the window object.

Parameter	Description
model	The model object that gets its parameters from the mode in NOL Studio.
modelName	The model name for the model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-show-predictive-model-prediction
(*modelName*: text, *win*: class g2-window)

Shows the predicted vs. actual chart for the predictive model with the specified name in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
modelName	The model name for the model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-validate-predictive-model
(*modelName*: text, *X*: class item-array, *Y*: class item-array,
win: class g2-window)

Uses the given input and output data to validate the predictive model with the specified name in NOL Studio. NOL Studio is owned by the window object. *X* and *Y* are item-arrays of float-arrays. One float-array is a row of data, and every float-array in one item-array should have the same array-length. You don't need to supply the variable name. The order of columns should match the input and output of the predictive model. *X* and *Y* should have the same array length.

Parameter	Description
modelName	The model name for the model in NOL Studio.
X	An item-array of float-arrays. This is the input matrix for validation. The float array length should be the same as the length of input variables.
Y	An item-array of float-arrays. This is the output matrix for validation. The float array length should be the same as the length of output variables.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Optimization

nols-get-optimization-names
(*win*: class g2-window)
-> <u>*models*</u>: sequence

Returns a sequence that contains the name of optimizations in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
<u>*models*</u>	A sequence of text strings, which are names of optimizations in the NOL Studio console.

nols-get-optimization
(*model*: class nols-optimization, *modelName*: text, *win*: class g2-window)

Loads the parameters from a optimization in the NOL Studio into the nols-optimization object. The optimization is specified by the model name as a input. The NOL Studio is owned by the window object.

Parameter	Description
model	The optimization model object that gets its parameters from the optimization in NOL Studio.
modelName	The model name for the optimization in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Backpropagation Net Model

nols-get-bpn-model-names
(*win*: class g2-window)
-> *models*: sequence

Returns a sequence that contains the name of backpropagation net models in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
models	A sequence of text strings, which are names of backpropagation net models in the NOL Studio console.

gnne-import-parameters-from-studio
(*net*: class bpn, *model-name*: text, *win*: class g2-window)
-> *status*: integer

Loads the model parameters from a backpropagation net model in the NOL Studio into the bpn object. The model is specified by the given model name. The NOL Studio is owned by the window object.

Parameter	Description
model	The bpn object that gets its parameters from the backpropagation net model in NOL Studio.
modelName	The model name for the backpropagation net model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
status	A status flag. If model parameters are successfully loaded into the bpn object, returns 1, otherwise, returns -1.

nols-show-bpn-model-prediction
(*modelName*: text, *win*: class g2-window)

Shows the predicted vs. actual chart for the backpropagation net model with specified name in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
modelName	The model name for the backpropagation net model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-validate-bpn-model
(*modelName*: text, *X*: class item-array, *Y*: class item-array,
win: class g2-window)

Uses the given input and output data to validate the backpropagation net model with specified name in the NOL Studio. The NOL Studio is owned by the window object. *X* and *Y* are item-arrays of float-arrays. One float-array is a row of data, and every float-array in one item-array should have the same array-length. You don't need to supply the variable name. The order of columns should match the input and output of the backpropagation net model. *X* and *Y* should have the same array length.

Parameter	Description
modelName	The model name for the backpropagation net model in NOL Studio.
X	An item-array of float-arrays. This is the input matrix for validation. The float array length should be the same as the length of input variables.
Y	An item-array of float-arrays. This is the output matrix for validation. The float array length should be the same as the length of output variables.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Radial Basis Function Net Model

nols-get-rbfn-model-names
(*win*: class g2-window)
-> *models*: sequence

Returns a sequence that contains the name of radial basis net models in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
<u>*models*</u>	A sequence of text strings, which are names of radial basis net models in the NOL Studio console.

gnne-import-parameters-from-studio
(*net*: class rbfn, *model-name*: text, *win*: class g2-window)
-> <u>*status*</u>: integer

Loads the model parameters from a radial basis net model in the NOL Studio into the rbfn object. The model is specified by the given model name. The NOL Studio is owned by the window object.

Parameter	Description
model	The rbfn object that gets its parameters from the radial basis net in NOL Studio.
modelName	The model name for the radial basis net in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
<u>*status*</u>	A status flag. If model parameters are successfully loaded into the rbfn object, returns 1, otherwise, returns -1.

nols-show-rbfn-model-prediction
(*modelName*: text, *win*: class g2-window)

Shows the predicted vs. actual chart for the radial basis net model with specified name in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
modelName	The model name for the radial basis net model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-validate-rbfn-model
(*modelName*: text, *X*: class item-array, *Y*: class item-array, *win*: class g2-window)

Uses the given input and output data to validate the radial basis net model with the specified name in the NOL Studio. The NOL Studio is owned by the window object. *X* and *Y* are item-arrays of float-arrays. One float-array is a row of data, and every float-array in one item-array should have the same array-length. You don't need to supply the variable name. The order of columns should match the input and output of the radial basis net model. *X* and *Y* should have the same array length.

Parameter	Description
modelName	The model name for the radial basis net model in NOL Studio.
X	An item-array of float-arrays. This is the input matrix for validation. The float array length should be the same as the length of input variables.
Y	An item-array of float-arrays. This is the output matrix for validation. The float array length should be the same as the length of output variables.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Autoassociative Net Model

nols-get-aan-model-names
(*win*: class g2-window)
-> *models*: sequence

Returns a sequence that contains the name of autoassociative net models in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
models	A sequence of text strings, which are names of autoassociative net models in the NOL Studio console.

gnne-import-parameters-from-studio
(*net*: class autoassociative-net , *model-name*: text, *win*: class g2-window)
-> *status*: integer

Loads the model parameters from a autoassociative net model in the NOL Studio into the autoassociative-net object. The model is specified by the given model name. The NOL Studio is owned by the window object.

Parameter	Description
model	The autoassociative-net object that gets its parameters from the autoassociative net in NOL Studio.
modelName	The model name for the autoassociative net in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	**Description**
status	A status flag. If model parameters are successfully loaded into the autoassociative-net object, returns 1, otherwise, returns -1.

nols-show-aan-model-prediction
(*modelName*: text, *win*: class g2-window)

Shows the predicted vs. actual chart for the autoassociative net model with the specified name in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	**Description**
modelName	The model name for the autoassociative net model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-validate-aan-model
(*modelName*: text, *X*: class item-array, *win*: class g2-window)

Uses the given data matrix to validate the autoassociative net model with specified name in the NOL Studio. The NOL Studio is owned by the window object. *X* is an item-array of float-arrays. One float-array is a row of data, and every float-array in one item-array should have the same array-length. You don't need to supply the variable name. The order of columns should match the variables used in the autoassociative net model.

Parameter	**Description**
modelName	The model name for the autoassociative net model in NOL Studio.
X	An item-array of float-arrays. This is matrix as the input data for validation. The float array length should be the same as the length of input variables.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Rho Net Model

nols-get-rho-model-names
(*win*: class g2-window)
-> *models*: sequence

Returns a sequence that contains the name of rho net models in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
models	Provides the sequence of text strings, which are names of rho net models in the NOL Studio console.

gnne-import-parameters-from-studio
(*net*: class rho-net , *model-name*: text, *win*: class g2-window)
-> *status*: integer

Loads the model parameters from a rho net model in the NOL Studio into the rho-net object. The model is specified by the given model name. The NOL Studio is owned by the window object.

Parameter	Description
model	The rho-net object that get its parameters from the rho net in NOL Studio.
modelName	The model name for the rho net in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
status	A status flag. If model parameters are successfully loaded into the rho-net object, returns 1, otherwise, returns -1.

nols-show-rho-model-prediction
(*modelName*: text, *win*: class g2-window)

Shows the predicted vs. actual chart for the rho net model with specified name in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
modelName	The model name for the rho net model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Ensemble Network

gnne-import-parameters-from-studio
(*net*: class ensemble-network, *model-name*: text, *win*: class g2-window)
-> *status*: integer

Loads the model parameters from a predictive model in the NOL Studio into the ensemble-network object. The model is specified by the given model name. The NOL Studio is owned by the window object.

Parameter	Description
model	The ensemble-network object that get its parameters from the predictive model in NOL Studio.
modelName	The model name for the predictive model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
status	A status flag. If model parameters are successfully loaded into the ensemble-network object, returns 1, otherwise, returns -1.

Partial Least Square Model

nols-get-pls-model-names
(*win*: class g2-window)
-> *models*: sequence

Returns a sequence that contains the name of partial least square models in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
models	Provides the sequence of text strings, which are names of partial least square models in the NOL Studio console.

gnne-import-parameters-from-studio
(*model*: class nols-pls-model, *model-name*: text, *win*: class g2-window)
-> *status*: integer

Loads the model parameters from a partial least square model in the NOL Studio into the nols-pls-model object. The model is specified by the given model name. The NOL Studio is owned by the window object.

Parameter	Description
model	The nols-pls-model object that get its parameters from the partial least square model in NOL Studio.
modelName	The model name for the partial least square model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
<u>*status*</u>	A status flag. If model parameters are successfully loaded into the nols-pls-model object, returns 1, otherwise, returns -1.

nols-show-pls-model-prediction
(*modelName*: text, *win*: class g2-window)

Shows the predicted vs. actual chart for the partial least square model with specified name in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
modelName	The model name for the partial least square model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

nols-validate-pls-model
(*modelName*: text, *X*: class item-array, *Y*: class item-array,
win: class g2-window)

Uses the given input and output data to validate the partial least square model with specified name in the NOL Studio. The NOL Studio is owned by the window object. *X* and *Y* are item-arrays of float-arrays. One float-array is a row of data, and every float-array in one item-array should have the same array-length. You don't need to supply the variable name. The order of columns should match the input and output of the partial least square model. *X* and *Y* should have the same array length.

Parameter	Description
modelName	The model name for the partial least square model in NOL Studio.
X	An item-array of float-arrays. This is the input matrix for validation. The float array length should be the same as the length of input variables.

Parameter	Description
Y	An item-array of float-arrays. This is the output matrix for validation. The float array length should be the same as the length of output variables.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Principal Component Analysis Model

nols-get-pca-model-names
(*win*: class g2-window)
-> *models*: sequence

Returns a sequence that contains the name of principal component analysis models in the NOL Studio. The NOL Studio is owned by the window object.

Parameter	Description
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
models	A sequence of text strings, which are names of principal component analysis models in the NOL Studio console.

gnne-import-parameters-from-studio

(*model*: class nols-pca-model, *model-name*: text, *win*: class g2-window)
-> *status*: integer

Loads the model parameters from a principal component analysis model in the NOL Studio into the nols-pls-model object. The model is specified by the given model name. The NOL Studio is owned by the window object.

Parameter	Description
model	The nols-pca-model object that get its parameters from the principal component analysis model in NOL Studio.
modelName	The model name for the principal component analysis model in NOL Studio.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

Return Value	Description
status	A status flag. If model parameters are successfully loaded into the nols-pca-model object, returns 1, otherwise, returns -1.

nols-validate-pca-model

(*modelName*: text, *X*: class item-array, *win*: class g2-window)

Uses the given data matrix to validate the principal component analysis model with specified name in the NOL Studio. The NOL Studio is owned by the window object. *X* is an item-array of float-arrays. One float-array is a row of data, and every float-array in one item-array should have the same array-length. You don't need to supply the variable name. The order of columns should match the variables used in the principal component analysis model.

Parameter	Description
modelName	The model name for the principal component analysis model in NOL Studio.

Parameter	Description
X	An item-array of float-arrays. This is matrix as the input data for validation. The float array length should be the same as the length of input variables.
win	The window object provided as an owner for this NOL Studio console and used for launching any message dialog.

GNNE Predictive Model

The APIs are organized by the functionality.

Model Import and Export

gnne-load-predictive-model
(*PredictiveModel*: class gnne-predictive-model)
-> *status*: truth-value

Loads the model variable information and network parameters from an XML file. Before calling this method, you need to set the file name and directory path to the attribute of the model object.

Parameter	Description
PredictiveModel	The model object into which the variable information and network parameters should be imported.

Return Value	Description
status	A status flag. If model parameters are successfully loaded into the model object, returns true, otherwise, returns false.

gnne-save-predictive-model
(*Model*: class gnne-predictive-model, *Stream*: class g2-stream)
-> *status*: truth-value

Saves the model variable information and network parameters into an XML file. You specify the file name and directory path as attributes of the model object.

Parameter	Description
Model	The model object into which the variable information and network parameters should be exported.
Stream	A G2 stream into which to write the model information.

Return Value	Description
status	A status flag. Returns true if model parameters are successfully exported from the model object; otherwise, returns false.

Model Properties

gnne-reset
(*Model*: class gnne-predictive-model)

Resets all model parameters to their default state, just as if it had been cloned from the palette.

Parameter	Description
Model	The model object to reset.

gnne-get-name
(*Model*: class gnne-predictive-model)
-> *name*: text

Returns the name of the model object.

Parameter	Description
Model	The Model object.

Return Value	Description
name	The name of the model object.

gnne-get-comment
(*Model*: class gnne-predictive-model)
-> *comment*: text

Returns the comments of the model object.

Parameter	Description
Model	The model object.

Return Value	Description
<u>*comment*</u>	The comment string of the model object.

gnne-get-variables

(*Model*: class gnne-predictive-model)
-> <u>*variables*</u>: sequence

Returns a sequence of model variables, which include inputs and outputs.

Parameter	Description
Model	The Model object whose variables to get.

Return Value	Description
<u>*variables*</u>	The sequence of model variables.

gnne-set-external-variables

(*Model*: class gnne-predictive-model, *Variables*: sequence)

Sets the inputs and outputs of the predictive models to the variables and parameters in G2. This method throws an error when:

- Not all inputs and outputs of the predictive model are assigned to a variable or a parameter.
- The name or tag of one of the variable structures does not match any input or output.
- The variable or parameter class of one of the variable structures is not a quantitative variable or parameter.

Parameter	Description
Model	The model object whose variables to set.
Variables	A sequence of structures, where each structure represents one variable. The syntax for each structure is: structure (variable-name: text, variable-tag: text, classification: text, ("input", "output", "actual") variable-or-parameter: class variable-or-parameter) The structure must contain variable-name or tag or both.

gnne-get-model-statistics
(*Model*: class gnne-predictive-model)
-> *model statistics*: structure

Returns a structure of model statistics.

Parameter	Description
Model	The model object whose statistics to get.

Return Value	Description
model statistics	A structure of model statistics. For example:

```
structure (output-statistics: sequence
  (structure (variable-name: "Output1",
              training: structure (rmse: 0.086,
                                   corrcoef: 0.982),
              testing: structure (rmse: 0.091,
                                  corrcoef: 0.964))))
```

gnne-has-time-stamps
(*Model*: class gnne-predictive-model)
-> *flag*: true-value

Determines whether a model has been trained based on a data series with time stamps.

Parameter	Description
Model	The Model object to test.

Return Value	Description
flag	Returns true when the model has been trained based on data series with time stamps.

gnne-get-output-variable-statistics
(*VarTable*: class gnne-variable-table, *VariableName*: text, *VariableTag*: text))
-> *success*: truth-value, *statistics*: structure

Returns the output statistics for online retraining. This method can be used to decide whether the retraining is valid.

Parameter	Description
VarTable	The variable table on the subworkspace of the GNNE Predictive Model.
VariableName	The name of the variable for which to get statistics.
VariableTag	The tag of the variable for which to get statistics.

Return Value	Description
success	Returns true if the variable and its output statistics are valid.
statistics	The output statistics, which is the result of online retraining. For example: structure (rmse: rmse, corrcoef: correlation)

Model Execution

gnne-set-variable-value-by-name
(*Model*: class gnne-predictive-model, *VariableName*: text, *Value*: float,
G2Time: quantity)

Sets the variable value for a model. The value is set to a data buffer, which is included in a variable table. After enough input data are set to the data buffer, you can calculate the output.

Parameter	Description
Model	The Model object whose variable value to set.
VariableName	The input variable name.
Value	The value to be set to the data buffer.
G2Time	The time stamp for the data value. If the original data series is row-based, use next method.

gnne-set-variable-value-by-name
(*Model*: class gnne-predictive-model, *VariableName*: text, *Value*: float)

Sets the variable value for a model. The value is set to a data buffer, which is included in a variable table. After enough input data are set to the data buffer, you can calculate the output.

Parameter	Description
Model	The Model object whose variable value to set.
VariableName	The input variable name.
Value	The value to be set to the data buffer.

gnne-set-variable-value-by-tag
(*Model*: class gnne-predictive-model, *VariableTag*: text, *Value*: float, *G2Time*: quantity)

Sets the variable value for a model. The value is set to a data buffer, which is included in a variable table. After enough input data are set to the data buffer, you can calculate the output.

Parameter	Description
Model	The Model object whose variable value to set.
VariableName	The input variable name.
Value	The value to be set to the data buffer.
G2Time	The time stamp for the data value. If the original data series is row-based, use next method.

gnne-set-variable-value-by-tag
(*Model*: class gnne-predictive-model, *VariableTag*: text, *Value*: float)

Sets the variable value for a model. The value is set to a data buffer, which is included in a variable table. After enough input data are set to the data buffer, you can calculate the output.

Parameter	Description
Model	The Model object whose variable value to set.
VariableName	The input variable tag.
Value	The value to be set to the data buffer.

gnne-calculate-outputs-for-row
(*Model*: class gnne-predictive-model)
-> *output values*: sequence

Calculates the outputs of a row-based model. The output values are returned in a sequence of structures. If the output variables are mapped to G2 variables and parameters, these variables are updated as well.

This method throws an error if:

- The model is not initialized.
- The required input data is not available.
- There is a calculation error.

Parameter	Description
Model	The Model object whose outputs to calculate.

Return Value	Description
output values	A sequence of structures, where each structure represents an output variable. The syntax for each structure is: structure (variable-name: *text*, variable-tag: *text*, unit: *text*, variable-value: *quantity*) The structure includes the variable-name, variable-tag, and unit only if they are available.

gnne-calculate-outputs-for-time
(*Model*: class gnne-predictive-model, *G2Time*: quantity)
-> <u>*output values*</u>: sequence

Calculates the outputs for a time-based model. The output values are returned as a sequence of structures. If the output variables are mapped to G2 variables and parameters, the variables are updated as well.

For a time-based model, you can request a time that does not necessarily correspond to a specific input. In this case, the output values are interpolated between the closest time steps.

This method throws an error if:

- The model is not initialized.
- The required input data, including all delayed values, is not available.
- There is a calculation error.

Parameter	Description
Model	The model object whose outputs to calculate.

Return Value	Description
<u>*output values*</u>	A sequence of structures, where each structure represents one output variable. The syntax for each structure is: structure (variable-name: *text*, variable-tag: *text*, unit: *text*, g2-time: *quantity*, variable-value: *quantity*) The structure includes the variable-name, variable-tag, and unit only if they are available.

gnne-calculate-online-model-statistics
(*Model*: class gnne-predictive-model)
-> <u>*model statistics*</u>: structure

Calculates the online model statistics for the model output variables.

Parameter	Description
Model	The model object whose statistics to calculate.

Return Value	Description

model statistics	A structure of model statistics. For example:
	structure
	(output-statistics: sequence (structure
	  (variable-name: "Output1",
	   training: structure (rmse: 0.086,
	                  corrcoef: 0.982),
	   testing: structure (rmse: 0.091,
	                  corrcoef: 0.964))))

Model Retraining

gnne-train-predictive-model
(*Model*: class gnne-predictive-model, *Xmatrix* : sequence, *Ymatrix*: sequence, *Time*: float, *Autostop*: truth-value, *InitialTraining*: truth-value, *display*: truth-value)

Trains the model online with pre-formatted input and output matrices. This method initially sets the attribute gnne-complete-training of the model to false, then spawns the training process through the nols-gateway. When the training finishes, gnne-complete-training is set to true. If during the training an error occurs, the gnne-has-error attribute of the model is set to true and the error message is set to gnne-error-message.

Parameter	Description
Model	The Model object to be trained.
Xmatrix	A sequence of float-arrays as the input matrix. This matrix must be formatted based on the input variables and their delays. The dimension of this matrix should be fit for the ensemble net structure in the GNNE Predictive Model.
Ymatrix	A sequence of float-array as the output matrix.
Time	The number of minutes for training the model.
Autostop	The flag indicates whether the training process automatically stops based on a converge criterion. When false, the training process continues to the time set by the *time* argument.

Parameter	Description
InitialTraining	When true, the model weights are initialized to a set of random number. When false, the training process starts with the existing model weights.
Display	When true, a training console with error information is displayed during training.

gnne-train-predictive-model

(*Model*: class gnne-predictive-model, *DataFiles*: sequence, *Time*: float, *Autostop*: truth-value, *InitialTraining*: truth-value, *Display*: truth-value)

Trains the model online with data series in `.ds` formatted files. This method initially sets the attribute gnne-complete-training of the model to false, then spawns the training process through the nols-gateway. When the training finishes, gnne-complete-training is set to true. If during the training an error occurs, the gnne-has-error attribute of the model is set to true and the error message is set to gnne-error-message.

Parameter	Description
Model	The Model object to be trained.
DataFiles	A text sequence, where the text string gives the file names for the data series. The data files has to be in a predefined `.ds` format.
Time	This argument set how many minutes you want to spend to train this model.
Autostop	The flag indicates whether the training process automatically stops based on a converge criterion. When false, the training process continues to the time set by the *time* argument.
InitialTraining	When true, the model weights are initialized to a set of random number. When false, the training process starts with the existing model weights.
Display	When true, a training console with error information is displayed during training.

Index