

NeurOn-Line

Reference Manual

Version 5.1 Rev. 0



NeurOn-Line Reference Manual, Version 5.1 Rev. 0

June 2016

The information in this publication is subject to change without notice and does not represent a commitment by Gensym Corporation.

Although this software has been extensively tested, Gensym cannot guarantee error-free performance in all applications. Accordingly, use of the software is at the customer's sole risk.

Copyright (c) 1985-2016 Gensym Corporation

All rights reserved. No part of this document may be reproduced, stored in a retrieval system, translated, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Gensym Corporation.

Gensym®, G2®, Optegrity®, and ReThink® are registered trademarks of Gensym Corporation.

NeurOn-Line™, Dynamic Scheduling™, G2 Real-Time Expert System™, G2 ActiveXLink™, G2 BeanBuilder™, G2 CORBALink™, G2 Diagnostic Assistant™, G2 Gateway™, G2 GUIDE™, G2GL™, G2 JavaLink™, G2 ProTools™, GDA™, GFI™, GSI™, ICP™, Integrity™, and SymCure™ are trademarks of Gensym Corporation.

Telewindows is a trademark or registered trademark of Microsoft Corporation in the United States and/or other countries. Telewindows is used by Gensym Corporation under license from owner.

This software is based in part on the work of the Independent JPEG Group.

Copyright (c) 1998-2002 Daniel Veillard. All Rights Reserved.

SCOR® is a registered trademark of PRTM.

License for Scintilla and SciTE, Copyright 1998-2003 by Neil Hodgson, All Rights Reserved.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations, and Gensym Corporation disclaims any responsibility for specifying which marks are owned by which companies or organizations.

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

Part Number: DOC126-510

Contents Summary

	Preface	xix
Part I	Entry and Paths	1
Chapter 1	Entry Points	3
Chapter 2	Path Displays	41
Chapter 3	Connections	47
Part II	Data Processing	59
Chapter 4	Scalar Blocks	61
Chapter 5	Vector Blocks	115
Chapter 6	Data Set Blocks	159
Part III	Neural Networks	213
Chapter 7	Neural Network Blocks	215
Chapter 8	Training Blocks	241
Part IV	Action and Other	267
Chapter 9	Action Utilities	269
Chapter 10	Inference Blocks	299
Chapter 11	Capabilities	321

Part V Application Programmer's Interface 337

Chapter 12 Application Programmer's Interface 339

Glossary 371

Index 379

Contents

Preface xix

About this Guide xix

Audience xxi

Conventions xxii

Related Documentation xxiii

Customer Support Services xxvi

Part I **Entry and Paths** 1

Chapter 1 **Entry Points** 3

Introduction 3

Using Entry Points 4

 Enabling Data Input 4

 Reading the Output Value 5

 Specifying How Often to Generate Values 6

Entry Points 7

 Using Entry Points to Obtain Data from a G2 Variable 8

 Choosing Between Embedded and External Data Sources 13

 Using a G2 Variable Directly 15

 Viewing the Variable 15

 Obtaining Values from External Datapoints 16

 Specifying the Embedded Value for an Entry Point 18

 Using Vector Entry Points 28

 Making Vector Values Permanent 28

 Configuring 29

 See Also 32

White Noise 34

 Configuring 34

 Example 35

 See Also 35

Sine Wave 37

 Specifying the Shape 37

 Specifying a Phase 37

Resetting 38
Configuring 38
See Also 39

Chapter 2 Path Displays 41

Introduction 41

Data Path Display 42

Determining Which Path Attribute to Display 42

Configuring 42

Example 42

See Also 43

Vector Path Display 44

Configuring 45

See Also 46

Chapter 3 Connections 47

Introduction 47

Connection Posts 49

Highlighting 49

Configuring 50

Examples 50

See Also 52

Connectors 53

Configuring 53

Example 54

See Also 54

Circuit Breakers 55

Configuring 55

Example 57

See Also 57

Part II Data Processing 59

Chapter 4 Scalar Blocks 61

Introduction 62

Performing Arithmetic Operations 63

Adding and Filtering Noise 64

Averaging Values 64

Stopping and Pausing Data 64

Outputting Data 64

Computing Statistical Properties 64

Defining Your Own Function	65
Summation	66
Configuring	66
Example	66
See Also	66
Difference	68
Configuring	68
Example	68
See Also	68
Change Sign	69
Configuring	69
Example	69
See Also	69
Bias	70
Configuring	70
Example	70
See Also	71
Multiplication	72
Configuring	72
Example	72
See Also	73
Quotient	74
Configuring	74
Example	74
See Also	74
Inverse	75
Configuring	75
Example	75
See Also	75
Gain	76
Configuring	76
Example	76
See Also	76
Additive Noise	78
Configuring	78
Example	78
See Also	79
Outlier Filter	80
Specifying a Range	80
Specifying How to Round Output Values	80
Configuring	81

Examples	82
See Also	83
First-Order Exponential Filter	84
Filtering	84
Specifying How to Round Output Values	85
Configuring	85
Example	86
See Also	86
Sample Median	87
Configuring	87
Example	88
See Also	88
Average Input Value	89
Configuring	89
Example	89
See Also	90
Median Input Value	91
Configuring	91
Example	91
See Also	92
Data Delay	93
Handling Multiple Signals	93
Resetting	93
Configuring	93
See Also	94
Data Inhibit	95
Resetting	95
Configuring	95
Example	96
See Also	96
Data Output	97
Configuring	97
See Also	98
Set Attribute	99
Configuring	99
Example	100
See Also	100
Data Shift	101
Specifying How to Delay Values	101
Configuring	101
Example	102
See Also	102

- Variance **103**
 - Configuring **103**
 - See Also **104**
- Moving Average **105**
 - Configuring **105**
 - Example **106**
 - See Also **106**
- Arithmetic Function **107**
 - Built-in G2 Function **107**
 - User-Defined Function **108**
 - Procedure **108**
 - Tabular Function **109**
 - Configuring **110**
 - See Also **110**
- Arithmetic Function of Two Arguments **111**
 - Using Built-in G2 Function **111**
 - Using a User-Defined Function **111**
 - Using a Procedure **112**
 - Configuring **113**
 - See Also **113**

Chapter 5 Vector Blocks 115

- Introduction **116**
 - Choosing When to Evaluate **116**
 - Creating Vectors and Scalars **117**
 - Using Vectors with Classifiers **117**
 - Manipulating Vectors **118**
 - Inhibiting Vectors **118**
 - Operating on Vector Elements **118**
- Vectorizer **119**
 - Configuring **119**
 - Example **120**
 - See Also **120**
- Scalarizer **122**
 - Configuring **122**
 - Example **123**
 - See Also **123**
- Windower **124**
 - Configuring **124**
 - Example **125**
 - See Also **125**
- Classifier Input Converter **126**

- Configuring **126**
- Example **127**
- See Also **127**
- Classifier Output Converter **128**
 - Configuring **128**
 - Example **128**
 - See Also **128**
- Vector Combiner **129**
 - Configuring **129**
 - Example **130**
 - See Also **130**
- Vector Splitter **131**
 - Configuring **131**
 - Example **132**
 - See Also **132**
- Vector Order Swapper **133**
 - Making Values Permanent **133**
 - Configuring **133**
 - Example **134**
 - See Also **135**
- Vector Inhibit **136**
 - Configuring **136**
 - See Also **137**
- Vector Rescaler **138**
 - Making Values Permanent **138**
 - Configuring **138**
 - Example **139**
 - See Also **140**
- Vector Sum **141**
 - Configuring **141**
 - Example **142**
 - See Also **142**
- Vector Difference **143**
 - Configuring **143**
 - Example **144**
 - See Also **144**
- Vector Product **145**
 - Configuring **145**
 - Example **146**
 - See Also **146**
- Vector Quotient **147**

Configuring **147**

Example **148**

See Also **148**

Vector Function **149**

Using a Built-in G2 Function **149**

Using a User-Defined Function **150**

Using a Procedure **150**

Using a Tabular Function **151**

Configuring **152**

See Also **153**

Vector Function of Two Arguments **154**

Using a Built-in G2 Function **154**

Using a User-Defined Function **155**

Using a Procedure **155**

See Also **157**

Chapter 6 Data Set Blocks 159

Introduction **160**

Creating Data Pairs **161**

Filtering Data **161**

Reading Data **162**

Copying Data **162**

Scaling Data **162**

Data Pair Buffer **163**

Specifying Whether Values are Concurrent **163**

Resetting **164**

Clearing the Data Pair Buffer **164**

Configuring **164**

Example **164**

See Also **165**

Data Pair Converter **166**

Configuring **166**

Example **166**

See Also **167**

Data Pair Divider **168**

Configuring **168**

Example **168**

See Also **168**

Data Pair Random Gate **169**

Configuring **169**

Example **170**

See Also **170**

- Data Pair Outlier Filter 171
 - Configuring 171
 - Making Values Permanent 172
 - Example 173
 - See Also 174
- Data Pair Quality Filter 175
 - Configuring 175
 - Example 175
 - See Also 176
- Data Set 177
 - Editing the Data Set 177
 - Entering and Viewing Data 179
 - Saving and Loading Data 180
 - Plotting Data 181
 - Text Format for Data Sets 181
 - Customizing the Text Format 182
 - Clearing the Data Set 183
 - Making Values Permanent 183
 - Configuring 183
 - See Also 183
- Maximum Age Filter 185
 - Configuring 185
 - Example 186
 - See Also 186
- Size Limitation Filter 187
 - Configuring 187
 - Example 188
 - See Also 188
- Data Set Reader 189
 - Resetting 189
 - Configuring 189
 - Example 190
 - See Also 190
- Random Divider 191
 - Configuring 191
 - Example 192
 - See Also 192
- S-Fold Divider 193
 - Configuring 193
 - Example 194
 - See Also 194
- Data Set Copier 195

- Configuring 195
- Example 196
- See Also 196
- Data Set Rescaler 197
 - Making Values Permanent 198
 - Configuring 199
 - See Also 200
- Data Set Plot 201
 - Configuring 201
 - Choosing What to Display 201
 - Choosing How to Display the Data 203
 - Choosing Where to Display the Data 206
 - Creating and Deleting Data Series 206
 - Making Values Permanent 207
 - Example 207
 - See Also 208
- Novelty Filter 209
 - Choosing Which Points to Keep 209
 - Deciding Whether a Data Pair is Novel 210
 - Making Values Permanent 211
 - Configuring 211
 - Example 212
 - See Also 212

Part III Neural Networks 213

Chapter 7 Neural Network Blocks 215

- Introduction 216
 - Saving and Loading Network Weights 216
 - Backpropagation and Autoassociative Networks 220
 - Radial Basis Function and Rho Networks 220
 - Ensemble Networks 220
- Backpropagation Net (BPN) 221
 - Configuring 222
 - Adjusting Weights 222
 - Saving and Loading Weights 224
 - Making Values Permanent 224
 - Examples 225
 - See Also 226
- Autoassociative Net 227
 - Configuring 228
 - Choosing the Run Mode 229

- Adjusting Weights **230**
- Saving and Loading Weights **230**
- Making Values Permanent **230**
- See Also **230**

- Radial Basis Function Net (RBFN) **231**
 - Configuring **232**
 - Saving and Loading Weights **233**
 - Making Values Permanent **233**
 - See Also **233**

- Rho Net **234**
 - Configuring **235**
 - Saving and Loading Weights **236**
 - Making Values Permanent **236**
 - See Also **236**

- Ensemble Net (ENN) **238**
 - Adjusting Weights **239**
 - Saving and Loading Weights **239**
 - Making Values Permanent **240**
 - Examples **240**
 - See Also **240**

Chapter 8 Training Blocks 241

- Introduction **241**
 - Basic Training and Testing **243**
 - Finding the Best Network Configuration **243**
 - Finding Which Inputs are Significant **243**

- Trainer **244**
 - Watching the Training Happen **244**
 - Configuring the Trainer for a Backpropagation, Autoassociative, or Ensemble Network **246**
 - Choosing the Maximum Number of Training Iterations **246**
 - Choosing the Training Method **246**
 - Choosing Whether to Accelerate Training **247**
 - Configuring the Trainer for a Radial Basis Function Network **247**
 - Configuring the Trainer for a Rho Network **249**
 - Example **249**
 - See Also **250**

- Fit Tester **251**
 - Configuring **251**
 - Example **252**
 - See Also **253**

- Train and Test **255**
 - Configuring **256**

The Train and Test Block's Subworkspace 257
Example 258
See Also 258

Five Fold CV 260
Configuring 261
Example 261
See Also 262

Sensitivity Tester 264
Making Values Permanent 264
Configuring 264
See Also 265

Part IV Action and Other 267

Chapter 9 Action Utilities 269

Introduction 270
Looping 271
Stopping Paths 271
Outputting Data 271
Branching 271
Performing Actions on Blocks 271
Invoking a Rule 272

Control Path Loop 273
Resetting 273
Configuring 273
Example 274

Control Path Circuit Breaker 275

N-to-1 Sieve 276
Resetting 276
Configuring 276
Example 276
See Also 277

Control Counter 278
Resetting 278
Configuring 278
Example 278
See Also 279

Control Inhibit 280
Resetting 280
Configuring 280
Example 281

See Also 281

Inference Output 282

Configuring 282

Example 283

See Also 284

Control Switch 285

Configuring 285

Example 285

See Also 285

Reset 286

Configuring 286

Example 286

See Also 287

Evaluate 288

Configuring 288

See Also 288

Clear 289

Configuring 289

Example 289

See Also 290

Make Permanent 291

Configuring 291

Example 291

See Also 292

Restore Permanent Values 293

Configuring 293

See Also 293

Attribute Transfer 294

Configuring 294

Example 294

See Also 295

Rule Action 296

Configuring 296

Example 296

See Also 297

Chapter 10 Inference Blocks 299

Introduction 299

Observations 300

Performing Logical Operations 301

Pausing Paths 301

High Value Observation, Low Value Observation	302
Specifying a Threshold	302
Configuring	303
See Also	304
Equality Observation	305
Configuring	306
See Also	307
Conclusion	308
Configuring	308
Example	309
See Also	310
AND Gate	311
How the Block Handles no-value Quality Inputs	311
Configuring	312
Example	313
See Also	313
OR Gate	314
Configuring	314
Example	315
See Also	316
NOT Gate	317
Configuring	317
Example	318
See Also	318
Inference Inhibit	319
Configuring	319
Example	320
See Also	320

Chapter 11 Capabilities 321

Introduction	321
Charting and Graphing Attributes	322
Forcing a Block to Evaluate	322
Starting a Control Signal	322
Chart Capability	323
Setting Up a Chart	323
Configuring a Chart	325
Choosing How a Block's Data is Displayed	327
Going to a Chart	330
Resetting	330
Configuring	331
Examples	332

See Also 333

Clock 334

Configuring 334

Example 335

See Also 335

Control Initiation Capability 336

Configuring 336

See Also 336

Part V Application Programmer's Interface 337

Chapter 12 Application Programmer's Interface 339

Introduction 339

Accessing the NOL API Procedures 340

Path Displays 341

Vector Blocks 342

Data Set Blocks 351

Neural Networks 360

Action Utilities 367

File Operations 369

Glossary 371

Index 379

Preface

Describes this manual and the conventions that it uses.

About this Guide **xxi**

Audience **xxiii**

Conventions **xxiv**

Related Documentation **xxv**

Customer Support Services **xxviii**



About this Guide

NeurOn-Line is a graphical, object-oriented software product for building neural network applications. Typical applications include quality assurance, sensor validation, diagnosis, and process modeling. Users of NeurOn-Line can model dynamic, nonlinear phenomena that are difficult to describe by analytical models.

This guide documents each block that can appear in these diagrams.

This guide is strictly a reference and does not teach you how to use NeurOn-Line. It assumes you are familiar with G2. See the *NeurOn-Line User's Guide* for basic information on how to use NOL, and for information on how to customize the NOL environment.

This guide is divided into these parts and chapters:

This part/chapter...	Describes...
Part I, Entry and Paths	The palettes in the Entry & Paths submenu. These palettes contain blocks that let you enter data into your application and manipulate paths.
Chapter 1, Entry Points	Blocks that let you enter data into your NeurOn-Line application.
Chapter 2, Path Displays	Blocks that display the data that a path is carrying.
Chapter 3, Connections	Blocks that let paths cross workspaces, join paths together, and create loops.
Part II, Data Processing	The palettes in the Data Processing submenu. These palettes contain blocks that let you create and manipulate scalar values, vectors, data pairs, and Data Sets
Chapter 4, Scalar Blocks	Blocks that perform various operations on scalar data, including arithmetic, filtering, and statistical.
Chapter 5, Vector Blocks	Blocks that create, manipulate, and perform operations on vectors.
Chapter 6, Data Set Blocks	Blocks that create, manipulate, and filter the data pairs and Data Sets that you use to train and test a Neural Network.
Part III, Neural Networks	The palettes in the Neural Networks submenu. These palettes contain blocks that implement, train, and test Neural Networks.
Chapter 7, Neural Network Blocks	Blocks that implement Backpropagation, Autoassociative, Radial Basis Function, and Rho Neural Networks.
Chapter 8, Training Blocks	Blocks that train and test Neural Networks.
Part IV, Action and Other	The palettes in the Action & Other submenu. These palettes contain blocks that execute actions, allow you to graph data, and test inference values.

Chapter 9, Action Utilities	Blocks that perform a variety of actions on objects in the G2 environment, including other NeurOn-Line blocks.
Chapter 10, Inference Blocks	Blocks that add features to other NeurOn-Line blocks, such as graphing.
Chapter 11, Capabilities	Blocks that create and manipulate inference values.

Each block in NeurOn-Line has a section that contains these components:

This section...	Contains this information...
Name	The name of the block that appears on the palette.
Icon	A full-size icon of the block. All stubs and attribute displays are labeled with their names.
Description	A description of what the block does and how to configure it.
Configuring	A picture of the configuration panel for the block, and a description of each attribute label in the panel.
Example	An example of how the block works. It could show how to use the block in a diagram or show what output values it passes given certain input values. If the block's description contains many examples, this section may be missing.
See Also	References to related sections in this manual. If some of a block's features are not discussed in the description, this section contains a reference to where that feature is described. If other blocks are similar to this block, this section contains references to them.

Audience

This book is intended to be used primarily by programmers using NOL to develop end-user applications. You should be familiar with G2.

This reference manual provides a menu-by-menu, palette-by-palette description of all the blocks in NeurOn-Line. Each menu has its own part, each palette in a menu has its own chapter, and each block as its own section in that chapter.

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
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
<i>new-argument</i>	User-specified values in syntax descriptions
<u>text-string</u>	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

Convention Examples	Description
<i>c:\Program Files\Gensym\</i>	Windows pathnames
<i>/usr/gensym/g2/kbs</i>	UNIX pathnames
<i>spreadsh.kb</i>	File names
<i>g2 -kb top.kb</i>	Operating system commands
<i>public void main() gsi_start</i>	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 underlined. 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/UII 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

Entry and Paths

Chapter 1: Entry Points

Describes the blocks you use to get data into a diagram by using sensor data from outside the application.

Chapter 2: Path Displays

Describes the blocks that display the value on paths

Chapter 3: Connections

Describes the blocks you use to get data into a diagram by using sensor data from outside the application.

Entry Points

Describes the blocks you use to get data into a diagram by using sensor data from outside the application.

Introduction	3
Using Entry Points	4
Entry Points	7
White Noise	34
Sine Wave	37

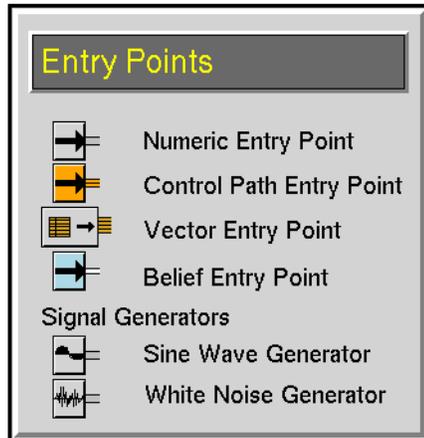


Introduction

The blocks on the Entry Points palette let you enter data into your application. The Entry Points enable you to get data externally from a G2 procedure, variable, parameter, from G2 Gateway (GSI), or from an embedded variable in the table for the block. The Signal Generators let you simulate values and test your application.

The blocks are: Numeric Entry Point, Control Entry Point, Vector Entry Point, Belief Entry Point, Sine Wave Generator, and White Noise Generator.

You can find these blocks on the Entry Points palette under the Entry & Path submenu of the Palettes menu:



Using Entry Points

This section describes general information about using entry points.

Enabling Data Input

Entry points and signal generators do not pass data until you explicitly enable data to flow. After you enable data input, the signal generators immediately update their output values, then update their value regularly.

To enable data to flow through a diagram:

→ Select Enable Data Input from the Controls menu.

A check mark appears next to the entry, indicating that data is enabled.

To stop data from flowing through a diagram:

→ Select Enable Data Input from the Controls menu when the menu choice is already selected.

You might want to turn data input off if you want G2 to run quicker as you modify a diagram.

Note Choosing the **override** menu choice from a block always passes a value, even when you disable data input.

Reading the Output Value

Entry points hold their current values in an attribute that defines a G2 variable or in the attribute **sensor-value**, depending on the current data source. Unlike all other blocks attributes, these attributes are displayed in the table for the block. Signal generators hold their current values in an attribute of the block named **output-value**.

This table shows the attribute that defines a G2 variable for each type of entry point:

For this entry point...	The attribute of the block that stores the output value is...
Numeric Entry Point	dp-out
Belief Entry Point	ip-out
Control Entry Point	cp-out
Vector Entry Points	vpv-out

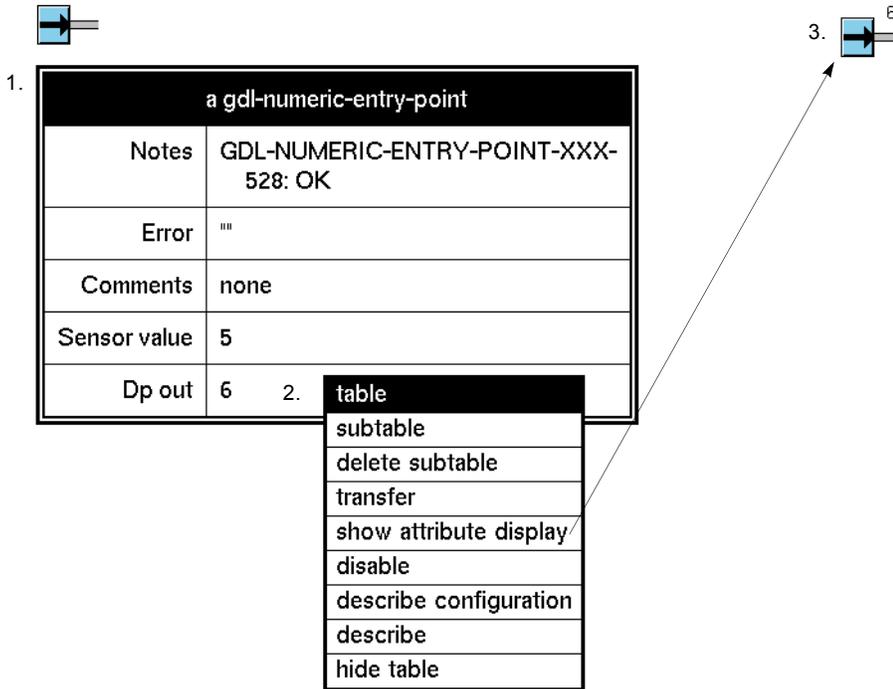
You can display the output value of an entry point or signal generator by showing the attribute display from the table.

Note You cannot display the output value of a Vector Entry Point because the value is contained in a list.

To see the output value beside the block:

- 1 Display the block's attribute table.
- 2 Click on the attribute's value (but not directly on the text) to display the menu.
- 3 Select show attribute display to display the current value.

This figure shows how to show the attribute display for a Numeric Entry Point:



You change an entry point's output value by changing the value of the block's G2 variable. For information on how to change a block's output value, see [Specifying the Embedded Value for an Entry Point](#). You change a signal generator's output value, you configure or override the block.

Note If you manually override the value of an entry point, the output value in the table for the block does not show the manually overridden value.

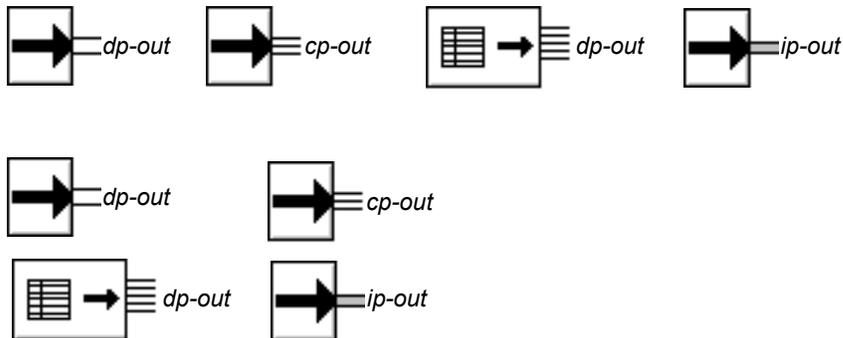
Specifying How Often to Generate Values

The field Sample Period specifies how often a signal generator passes a new value. If this value is large, the block produces a coarser signal and your application runs faster. By default, the value is 5 seconds.

For example, if a Real Time Clock block has a Sample Period of 1 second, it might send out these values over 10 seconds: 4001, 4002, 4003, 4004, 4005, 4006, 4007, 4008, 4009, and 4010. If the block has a Sample Period of 2 seconds, it would send out the following values over the same period of time: 4002, 4004, 4006, 4008, and 4010.

If you change a block's Sample Period while your application is running, NeurOn-Line resets the block and starts passing values according to the new period.

Entry Points



The five blocks are from left to right: Numeric Entry Point, Text Entry Point, Symbolic Entry Point, Belief Entry Point, and Control Entry Point.

Any Entry Point except a Vector Entry Point enters data into a diagram from a G2 procedure, variable, parameter, or GSI. Any Entry Point other than a Vector Entry Point can get its current value from one of two sources:

- From an **embedded data source**, which is a G2 variable that is an attribute of the block. You use an embedded data source when you want to obtain data from a procedure, formula, or action button, or rule. You can conclude a value directly into the G2 variable.
- From an **external data source**, which provides data directly from a G2 sensor. You use an external data source when you want to obtain data from a variable, parameter, or sensor.

The name of the attribute that contains the G2 variable that is the embedded data source depends on the type of Entry Point, as described in [Reading the Output Value](#). For example, for a Numeric Entry Point, the name of this attribute is `dp-out`.

You configure the name of the G2 variable in the Name of Sensor attribute of the Entry Point. You configure how long the internal data is valid in the Validity Interval attribute of the entry point. You configure the data source, the formula, and the update interval of the embedded data source in the variable's subtable

You can toggle between the embedded and external data source while running your diagram to toggle between simulated and real-time data.

Using Entry Points to Obtain Data from a G2 Variable

You use Entry Points to obtain data from G2 variables. For example, you do this when you want to place all sensors for a diagram on a single workspace.

Entry Points obtain output data from one of two locations, depending on whether you are using an embedded or external data source:

- When you are using an embedded data source, the block obtains its output value from an attribute of the block, which is itself a variable. The name of this attribute depends on the type of entry point. For example, a Numeric Entry Point defines the attribute `dp-out`, which is an embedded variable.
- When you are using an external data source, the block obtains its output data directly from the external sensor. The Entry Point stores the current value of the external variable in an attribute of the block named `sensor-value`.

You can switch between these two data sources by configuring the Data Source attribute of the block. The output value of the block depends on the data source the block is using.

When you configure the attributes of the block through the configuration panel, you are configuring the *embedded* data source, namely, the attribute of the block that contains a variable, for example, `dp-out`. Configuring the block has no effect on the external data source.

To configure the block to use an external or embedded data source:

- 1 Create, name, and specify a G2 variable that supplies data to the Entry Point.
Typically, the data server for this variable is an external data source, such as a GSI data server or G2. This variable is the external data source. You must specify the `Formula` and `Default-update-interval` attributes. You typically also specify the `Validity-interval` and `Data-server` attributes.
- 2 Click on the embedded variable, for example, `dp-out`, select the `subtable` menu choice, and specify the attributes the embedded variable.
This variable is the embedded data source. You must specify the `Formula` and `Default-update-interval` attributes. You do not need to specify the `Validity-interval` attribute because you configure this attribute for the block. Also, you do not typically specify the `Data-server` attribute because the variable typically simulates real-time data in its formula.
- 3 Configure the `Name of Sensor` attribute of the Entry Point to specify the G2 variable that supplies data to the Entry Point.

If the Data Source attribute is **external**, the named sensor must exist before you enter its name in the Name of Sensor. If the Data Source attribute is **embedded**, the named sensor does not have to exist before you enter it.

When you configure the Name of Sensor, you are configuring the Name-of-sensor attribute in the subtable of the embedded G2 variable, for example, dp-out.

You can specify the Name of Sensor as an expression that evaluates to a G2 variable. For more information, see [Evaluating Expressions in Attributes](#) in the *NeurOn-Line User's Guide*.

- 4 Configure the Data Source to be either **embedded** or **external**.
 - **Embedded** means the Entry Point will use the G2 variable stored in the attribute of the block, for example, dp-out.
 - **External** means the Entry Point will use the external sensor whose value is stored in the **Sensor-value** attribute of the block.

- 5 Configure the Validity Interval of the Entry Point to how long the value of the embedded data source remains valid.

When you configure the Validity Interval, you are configuring the Validity-interval attribute in the subtable of the embedded G2 variable, for example, dp-out.

The default Validity Interval is **supplied**, which means the Entry Point uses the Validity-interval supplied by the specified data source.

Specify the Validity Interval as either a time interval, for example, **5 seconds**, or **indefinite**, in which case the data value never expires.

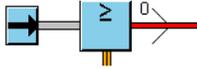
You can also override the Validity Interval given by an Entry Point by creating a rule that concludes a value directly into the dp-out output path attribute of the Entry Point, using the **with expiration** syntax, e.g., **conclude that the dp-out of EP-1 = the current time with expiration (the current time + 5)**.

- 6 Specify Value on Initialization to supply a default value when the Entry Point is reset.

Note Initial values never expire, even if you specify Value on Initialization and Validity Interval for a block. Also, manual values that you provide by overriding the block never expire.

This next figure shows a Numeric Entry Point whose Name of Sensor is pointing to a variable named **float-var-1**, which is the external data source.

The configuration panel specifies a Validity Interval of **5 seconds**, which determines the validity of the embedded data source. The default Data Source is **embedded**, which means the block uses the value generated by the dp-out embedded variable.



Numeric Entry Point	
Name of Sensor	float-var-1
Data Source	<input checked="" type="radio"/> embedded <input type="radio"/> external
Validity Interval	5 seconds
Value on Initialization	NONE
<input type="button" value="OK"/> <input type="button" value="Apply"/> <input type="button" value="Cancel"/>	

This next figure shows the external data source, the variable named `float-var-1`, and its associated table. The variable generates random numbers between 1 and 10 once every 20 seconds, and the data is valid for 10 seconds.



10.0, expires 20 Aug 96 2:18:12 p.m.

FLOAT-VAR-1

FLOAT-VAR-1, a float-variable	
Options	do not forward chain, breadth first backward chain
Notes	OK
Item configuration	none
Names	FLOAT-VAR-1
Tracing and breakpoints	default
Data type	float
Initial value	none
Last recorded value	10.0, expires 20 Aug 96 2:18:12 p.m.
History keeping spec	do not keep history
Validity interval	10 seconds
Formula	random (1,10)
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	20 seconds

The following figure shows the table for the Numeric Entry Point, which contains the `dp-out` attribute and the `sensor-value` attribute, and the subtable for the `dp-out` attribute.

The `dp-out` attribute defines an embedded variable, which is the embedded data source for the Entry Point. The embedded variable generates random numbers between 11 and 20 once every 10 seconds, as the subtable shows.

Notice that the `Validity-interval` of the embedded variable corresponds to the `Validity Interval` attribute in the configuration panel for the block. The `Sensor-value` attribute shows the current value of the external data source, the variable named `float-var-1`.

a gdl-numeric-entry-point	
Notes	GDL-NUMERIC-ENTRY-POINT-XXX-350: OK
Error	""
Comments	none
Sensor value	4.0
Dp out	16

a gdl-quantitative-entry-point-variable, the dp out of some gdl-numeric-entry-point	
Options	do not forward chain, breadth first backward chain
Notes	OK
Item configuration	none
Names	none
Tracing and breakpoints	default
Data type	quantity
Initial value	none
Last recorded value	16, expires 20 Aug 96 2:24:00 p.m.
History keeping spec	do not keep history
Validity interval	5 seconds
Formula	random(11,20)
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	10 seconds
Name of sensor	float-var-1
Quality	ok



Choosing Between Embedded and External Data Sources

By default, all Entry Points obtain their output values from the embedded data source, for example, the variable `dp-out`. You can cause the Entry Point to obtain its data from the external data source by reconfiguring the entry point. In this way, you can switch between simulated data that the embedded data source generates, and live data that the external data source generates.

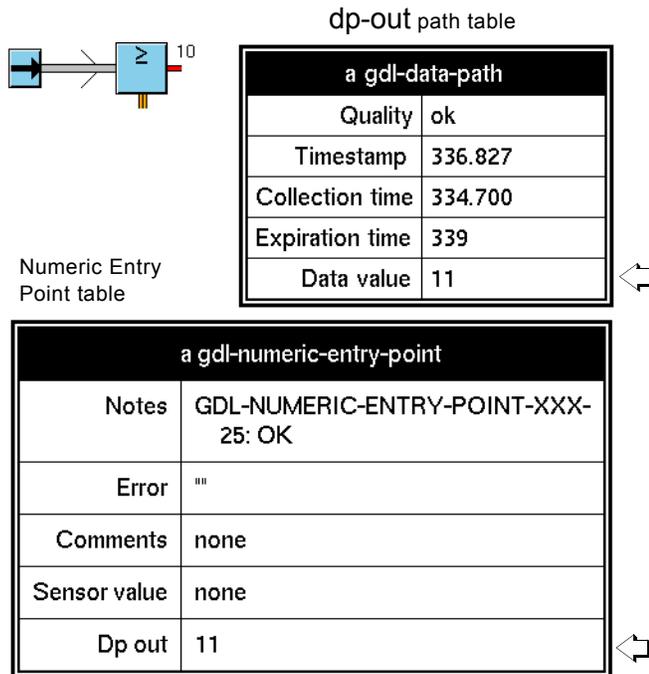
To choose between embedded and external data sources:

→ Configure the Data Source attribute to be **external** or **embedded**.

When the block is obtaining its data from the embedded data source, for example, the `dp-out` variable, the arrow in the Entry Point's icon is black. When you configure the Data Source attribute to be **external**, the arrow in the Entry Point's icon changes to the active color of the block, whose default is cyan. In this way, you can determine the current data source.

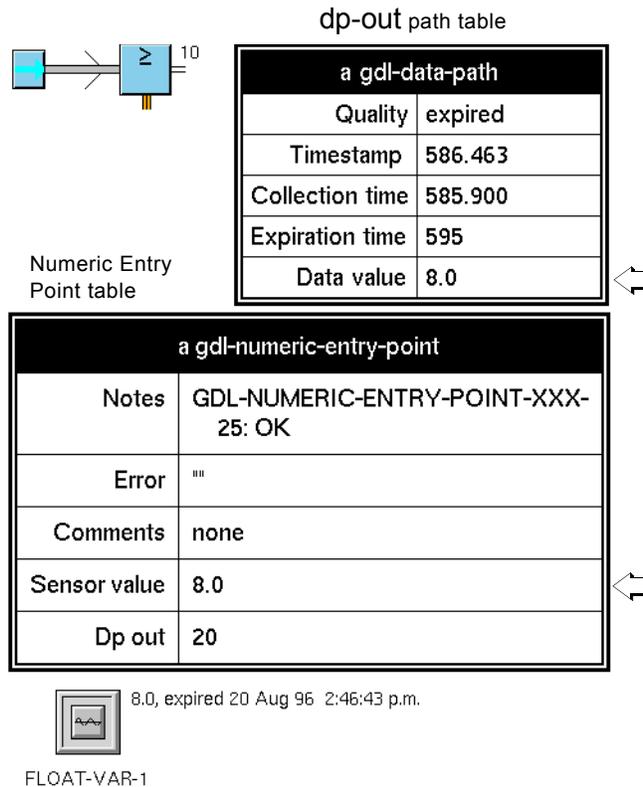
Note Even when you are using the external data source, the value of the embedded variable that is the embedded data source, for example, the `dp-out`, continues to update. To verify that the block is using the correct data source, display the table for the output path of the block.

The following figure shows the table for the output data path for a Numeric Entry Point when Data Source is **embedded**. Notice that the **Data-value** on the output data path of the Numeric Entry Point corresponds to the value of the `dp-out` embedded variable in the block's table. The output value is greater than 10, thus the inference output path of the High Value observation is true.



The following figure shows the result of configuring the Data Source attribute to be external.

Notice that the arrow on the icon of the Numeric Entry Point is now cyan, indicating it is using the external data source. The **Data-value** on the output data path of the Numeric Entry Point corresponds to the value of the **Sensor-value** attribute in the block's table, which is the current value of the **float-var-1** external variable. The output value is less than 10, thus the inference output path of the High Value observation is false.



Using a G2 Variable Directly

You do not need to use an Entry Point to obtain data from a G2 variable in a diagram. Instead, you can connect variables directly to a path simply by dragging a path into the variable and making the connection. For more information on how to do this, see [Using Variables and Parameters](#) in the *NeurOn-Line User's Guide*.

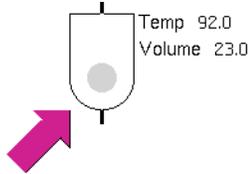
Viewing the Variable

You can view the variable that is the sensor for a particular Entry Point, using a menu choice. This menu choice is especially useful if the Entry Point and the variable are on different workspaces.

To view the variable for an Entry Point:

- ➔ Choose go to sensor from the Entry Point's menu.

NOL shows the workspace of the variable and places an arrow near the variable for a number of seconds. If the variable is embedded in another G2 object, the arrow points to that object, as the following figure shows.



Obtaining Values from External Datapoints

You can use an external datapoint to set the output value for an Entry Point to obtain data from external systems, such as Distributed Control Systems (DCS). You create and configure external datapoints by using an External Datapoint Configuration block, which creates external datapoints from a `.csv` file. This block also requires a Network Interface to provide communication with the external system.

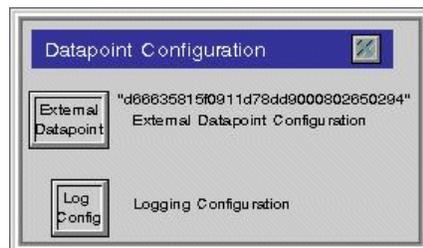
Once you create the external datapoints, you can then create Entry Points automatically from some or all of the external datapoints.

This feature is available in the Gensym Data Point Management System (GDPM) module, which is available as part of NeurOn-Line. This module also provides datapoint logging and data replay capabilities.

For more information, see the *Optegrity User's Guide*.

To create entry points from external datapoints:

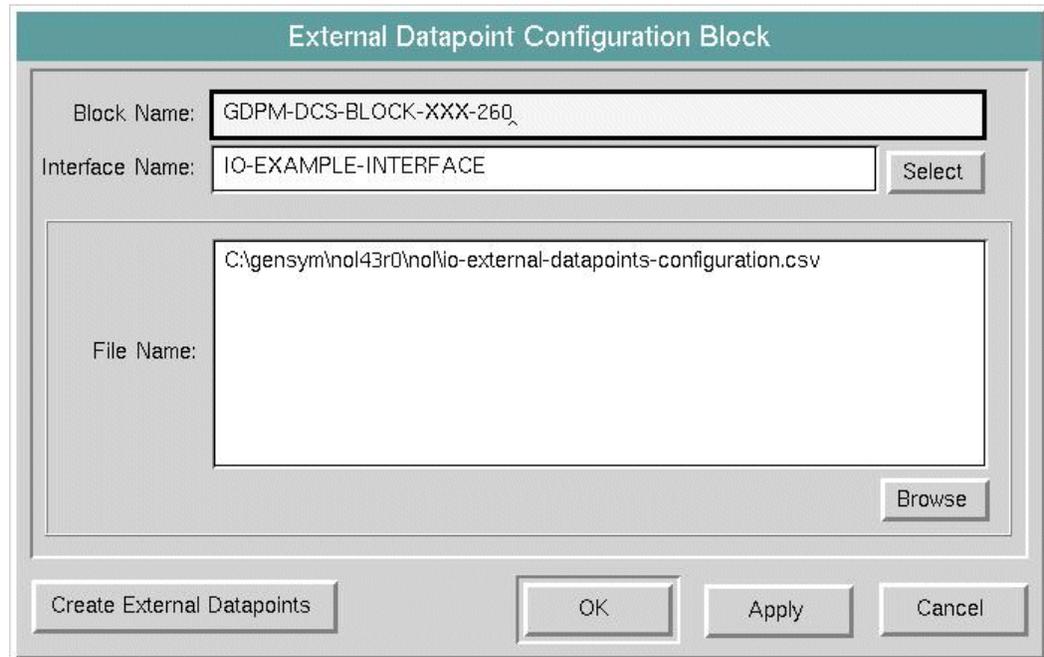
- 1 Choose Palettes > Datapoint Configuration > Data Points to display this palette:



- 2 Create and configure an External Datapoint Configuration block.

Choose Properties on the block and specify a unique name for the Block Name. Select the Interface Name to use for communicating with the external system. Specify the name of the `.csv` file that contains configuration information for the external datapoints.

Here is the properties dialog for an External Datapoint Configuration block:



The screenshot shows a dialog box titled "External Datapoint Configuration Block". It contains the following fields and controls:

- Block Name:** A text box containing "GDPM-DCS-BLOCK-XXX-260".
- Interface Name:** A text box containing "IO-EXAMPLE-INTERFACE" with a "Select" button to its right.
- File Name:** A text box containing "C:\gensym\nol43r0\nol\io-external-datapoints-configuration.csv" with a "Browse" button to its right.
- Buttons:** At the bottom, there are four buttons: "Create External Datapoints", "OK", "Apply", and "Cancel".

For detailed information on configuring an External Datapoint Configuration block, see Chapter 8 “Configuring External Datapoints” in the *Optegrity User’s Guide*.

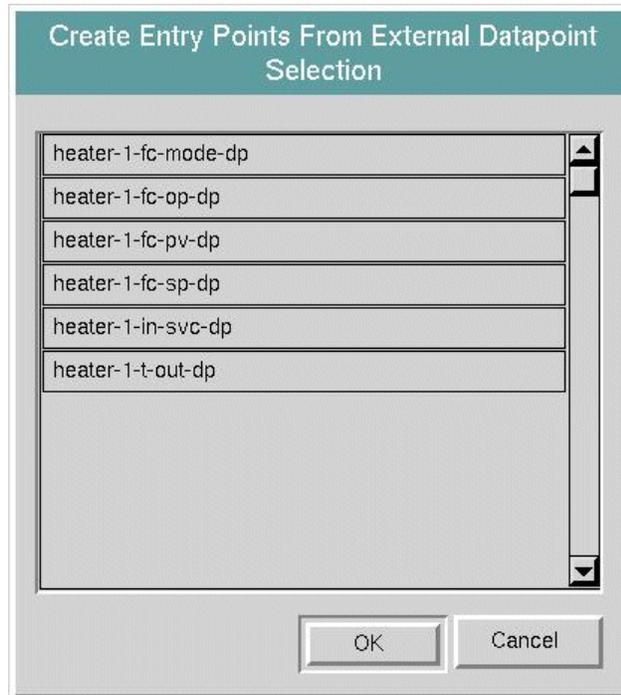
3 Create the external datapoints.

You create the external datapoints by clicking the Create External Datapoints button in the properties dialog or by choose Create External Datapoints on the block.

The external datapoints are created from the configuration information in the specified file and appear on the detail of the block.

- 4 Choose Create Entry Point from the popup menu of the workspace.

A dialog appears for choosing the external datapoints from which to create Entry Points:



- 5 Click OK.

The Entry Points appear on the same workspace as the External Datapoint Configuration block.

You can now build your diagram from these Entry Points to obtain values from external systems through these external datapoints.

Specifying the Embedded Value for an Entry Point

In addition to using a variable to set the output value for an Entry Point as described in [Using Entry Points to Obtain Data from a G2 Variable](#), you can set the output value of the embedded variable by using:

- **A button.** Any G2 button, such as sliders and action buttons, can set the value for an Entry Point.
- **A rule or procedure.** A G2 rule or procedure can conclude a value for an Entry Point.

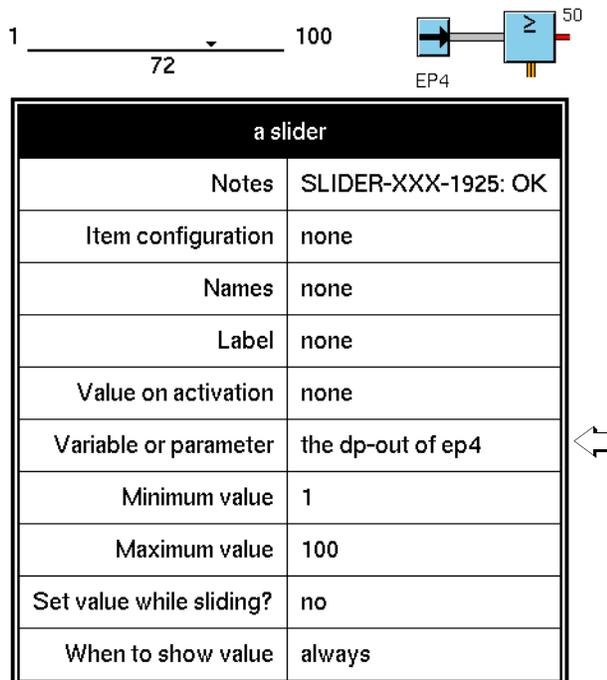
- **A formula.** In the subtable for the Entry Point's variable, you can specify a formula that G2 evaluates when it needs the Entry Point's value.
- **Your own variable or parameter.** You can replace the Entry Point's variable with a variable or parameter created from your own object definition.

Note The Data Output block performs the opposite action of the Entry Points; it passes information *from* a diagram *to* a G2 variable or parameter.

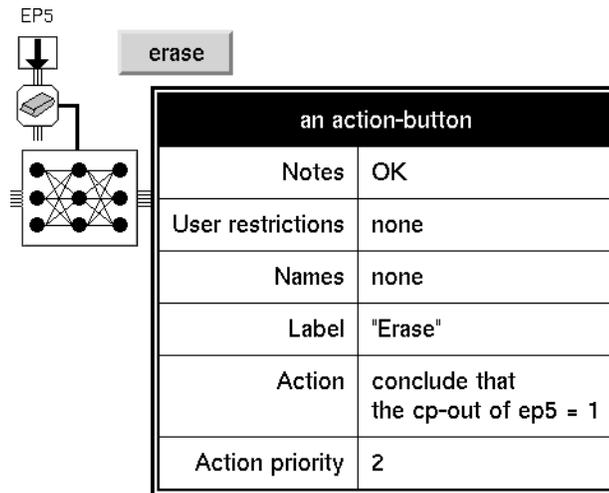
Using a Button

Buttons are especially useful when you are testing an application or creating a demo. They let the operator choose the value for an Entry Point. If you are using a slider, radio button, or check box, set the attribute **Variable-or-parameter** to the Entry Point's variable. If you are using an action button, use a **conclude** statement to set the Entry Point's variable.

For example, the following figure shows a slider that sets the output value for a Numeric Entry Point:



In this figure, an action button starts a sequence of action blocks.



Using a Rule or Procedure

A G2 rule or procedure can set the value for an Entry Point. Whenever the rule or procedure executes, it sets the value of the Entry Point, which then passes it. To set the value of the Entry Point, use a **conclude** statement.

Note You cannot set the value of a Vector Entry Point with a rule or procedure.

This figure shows a Belief Entry Point that gets a value from a rule. Whenever any tank has a temperature over 100, the Entry Point passes along the status value.



EP1

<p>whenever the temperature T of any tank receives a value and when T > 100 then conclude that the ip-out of ep1 = 1.0</p>
--

The next figure shows a Control Entry Point that gets a value from the procedure `process-bottles`. When you call the procedure, the Entry Point passes along one control signal for each bottle.



EP2



```
process-bottles(num-bottles: integer)
begin
  conclude that the cp-out of ep2 = num-
  bottles
end
```

This figure shows a Numeric Entry Point that gets a value from the procedure `adjust-speed`. The procedure decrements the value of the Entry Point by an amount you specify.



EP3



```
adjust-speed(delta:float)
speed:float;
begin
  speed = gdl-get-data-path-value(ep3, the
  dp-out of ep3);
  conclude that the dp-out of ep3 = speed -
  delta;
end
```

Using a Formula

You can give a formula to an Entry Point's variable. G2 evaluates the formula at the interval specified in the variable's `Default-update-interval` attribute.

To specify the formula:

- 1 In the attribute table for the Entry Point, click on the attribute `dp-out`, `ip-out`, or `cp-out`, and select **subtable** from the menu.

NOL displays the attribute's subtable.

- 2 Edit the attribute `Formula` in the subtable.
- 3 Set the attribute `Default-update-interval` to the interval at which you want NOL to evaluate the formula.

1. 

a gdl-numeric-entry-point	
Notes	OK
Error	""
Comments	none
Sensor value	*****
Dp out	****

table
subtable
delete subtable
transfer
show attribute display
disable
describe configuration
describe
hide table

2.and 3.

a gdl-quantitative-entry-point-variable, the dp out of some gdl-numeric-entry-point	
Options	do not forward chain, breadth first backward chain
Notes	the dp out of GDL-NUMERIC-ENTRY-POINT-XXX-451: OK
Item configuration	none
Names	none
Tracing and breakpoints	default
Data type	quantity
Initial value	none
Last recorded value	no value
History keeping spec	do not keep history
Validity interval	indefinite
Formula	average(the volume of t1, the volume of t2, the volume of t3)
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	inference engine
Default update interval	10 seconds
Name of sensor	none
Quality	ok



Using Your Own Variable Definition

You can replace the variable in an Entry Point with a variable created from your own object definition. This method is especially useful when you would like to add attribute to the Entry Point or change the inheritance of the embedded variable.

To use your own variable definition:

- 1 Create the variable definition.

This figure shows a completed attribute table for a variable definition that uses GSI:

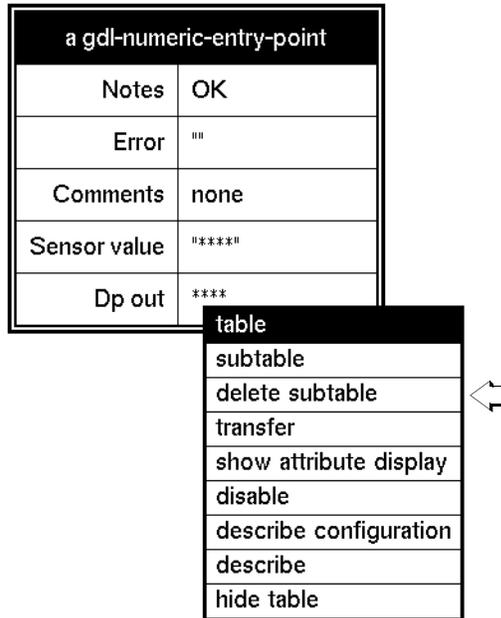
MY-GSI-VARIABLE



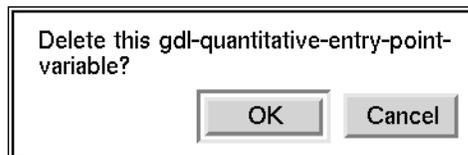
MY-GSI-VARIABLE, an object-definition	
Notes	OK
User restrictions	none
Class name	my-gsi-variable
Superior class	float-variable
Attributes specific to class	none
Capabilities and restrictions	gsi-data-service
Class restrictions	none
Change	none
Menu option	a final menu choice
Inherited attributes	none
Default settings	none
Attribute displays	inherited
Stubs	inherited
Color	inherited
Icon description	inherited

- 2 Go into Administrator mode.

- 3 Delete the subtable for the Entry Point's variable by clicking on the variable in the Entry Point's attribute table, and selecting **delete subtable** from the menu that appears:



- 4 Click OK in the dialog that G2 displays, asking you to confirm that you want to delete the subtable.



- 5 Add a new subtable for your variable definition by clicking on the variable in the Entry Point's attribute table, and selecting add optional subtable from the menu that appears:

a gdl-numeric-entry-point	
Notes	OK
Error	""
Comments	none
Sensor value	*****
Dp out	none

table
add optional subtable ▶
edit
transfer
show attribute display
hide table

G2 displays a menu asking you to choose a class.

- 6 Choose **g2-variable** from the menu, and follow the menu hierarchy down until you see the name of the superior class used to define the new variable class specified in Step 1.

The final menu contains your new variable definition. Click on the name of your variable definition.

G2 displays the subtable for your variable.

7 Edit the variable to suit your needs.

a my-gsi-variable, the dp out of some gdl-numeric-entry-point	
Options	do not forward chain, breadth first backward chain
Notes	OK
User restrictions	none
Names	none
Tracing and breakpoints	default
Data type	float
Initial value	none
Last recorded value	no value
History keeping spec	do not keep history
Validity interval	supplied
Formula	none
Simulation details	no simulation formula yet
Initial value for simulation	default
Data server	GSI data server
Default update interval	none
Gsi interface name	none
Gsi variable status	0

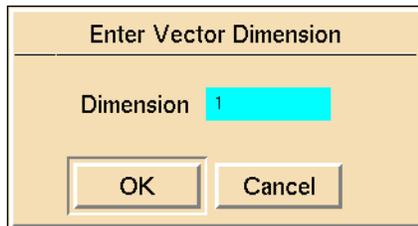
Using Vector Entry Points

To generate values from a Vector Entry Point, you override the block. You specify the dimension of the vector, and then the vector values.

To override a Vector Entry Point:

- 1 Select the **override** menu choice on the entry point.

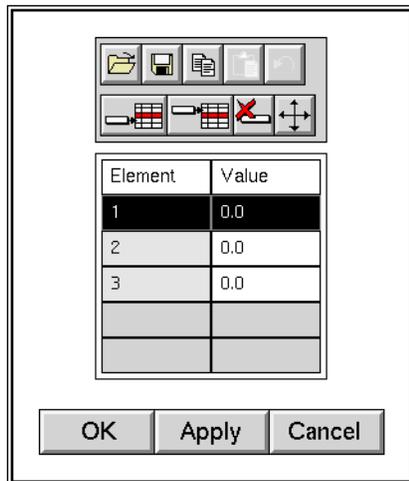
NOL displays this dialog:



The dialog box is titled "Enter Vector Dimension". It contains a label "Dimension" followed by a text input field containing the number "1". Below the input field are two buttons: "OK" and "Cancel".

- 2 Enter the dimension of the vector and select OK.

NOL displays a spreadsheet for editing the vector values. For example, here is a spreadsheet for entering a 3-dimensional vector:



The spreadsheet has a toolbar at the top with icons for file operations and editing. Below the toolbar is a table with two columns: "Element" and "Value". The table contains three rows of data, each with a value of 0.0. Below the table are three buttons: "OK", "Apply", and "Cancel".

Element	Value
1	0.0
2	0.0
3	0.0

- 3 Enter the values for each dimension, and select OK.

For more information about using the spreadsheet to edit vectors, see [Using the GXL Spreadsheet to Edit Data](#).

Making Vector Values Permanent

When you choose **make permanent** from a Vector Entry Point's menu, the block saves its current output value. If you later reset G2, the block restores that value, but that value is not passed until the block evaluates.

Configuring

This is the configuration panel for the Numeric Entry Point.

Attribute	Description
Name of Sensor	The name of the G2 variable that supplies data to the Entry Point. The sensor you specify must exist when Data Source is external . For information on how to use an expression for the Name of Sensor, see Evaluating Expressions in Attributes in the <i>GDA User's Guide</i> .
Data Source	Determines whether the entry point obtains its output value from the embedded variable, which is an attribute of the entry point, for example, dp-out , or from an external variable, which is the value of the Name of Sensor attribute. When Data Source is external , the arrow on the icon changes to the active color for blocks.

Attribute	Description
Validity Interval	<p>The amount of time that the current value of the Entry Point remains valid, specified either as a time interval, for example, 5 seconds, or indefinite, in which case the data value never expires.</p> <p>The default Validity Interval is supplied, which means the Entry Point uses the Validity-interval supplied by the Name of Sensor variable.</p> <p>The specification of this attribute overrides the Validity-interval given by the variable.</p>
Value on Initialization	See Specifying Initial Values in the <i>NeurOn-Line User's Guide</i> .

This is the configuration panel for the Belief Entry Point.

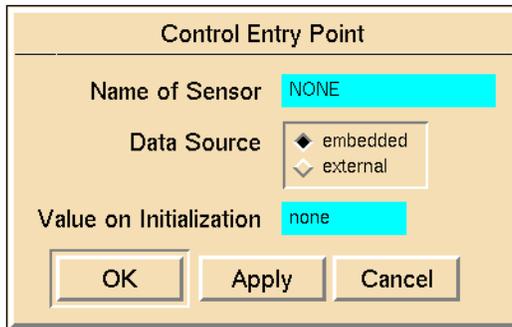
The screenshot shows a configuration window titled "Belief Entry Point". It contains the following fields and options:

- Name of Sensor:** A text field containing "NONE".
- Data Source:** A dropdown menu with "embedded" selected (indicated by a diamond) and "external" unselected (indicated by a diamond).
- Validity Interval:** A text field containing "supplied".
- Logic:** A dropdown menu with "discrete" selected (indicated by a diamond) and "fuzzy" unselected (indicated by a diamond).
- Status on Initialization:** A dropdown menu with "none" selected (indicated by a diamond) and "true", "false", and "unknown" unselected (indicated by diamonds).
- Output Uncertainty:** A text field containing "0.5".

At the bottom of the window, there are four buttons: "Descriptions", "OK", "Apply", and "Cancel".

Attribute	Description
Name of Sensor	The name of the G2 variable that supplies data to the Entry Point.
Data Source	Determines whether the entry point obtains its output value from the embedded variable, which is an attribute of the entry point, for example, <code>dp-out</code> , or from an external variable, which is the value of the Name of Sensor attribute. When Data Source is external , the arrow on the icon changes to the active color for blocks.
Validity Interval	<p>The amount of time that the current value of the Entry Point remains valid, specified either as a time interval, for example, 5 seconds, or indefinite, in which case the data value never expires.</p> <p>The default Validity Interval is supplied, which means the Entry Point uses the Validity-interval supplied by the Name of Sensor variable.</p>
Logic	This attribute is not supported in NOL.
Status on Initialization	See Specifying an Initial Status Value in the <i>NeurOn-Line User's Guide</i> .
Output Uncertainty	This attribute is not supported in NOL.
Description when True, Description when False, Description when Unknown	These attributes are not supported in NOL.

This is the configuration panel for the Control Entry Point.



Attribute	Description
Name of Sensor	The name of the G2 variable that supplies data to the Entry Point.
Data Source	Determines whether the entry point obtains its output value from the embedded variable, which is an attribute of the entry point, for example, <code>dp-out</code> , or from an external variable, which is the value of the Name of Sensor attribute. When Data Source is <code>external</code> , the arrow on the icon changes to the active color for blocks.
Value on Initialization	See Specifying an Initial Data Value in the <i>NeurOn-Line User's Guide</i> .

A Vector Entry Point has no configuration dialog. Instead, you override the vector entry point, as described in [Using Vector Entry Points](#).

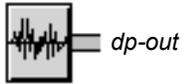
See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Overriding Block Values		<i>User's Guide</i>
Controlling the Flow of Data in an Application		<i>User's Guide</i>

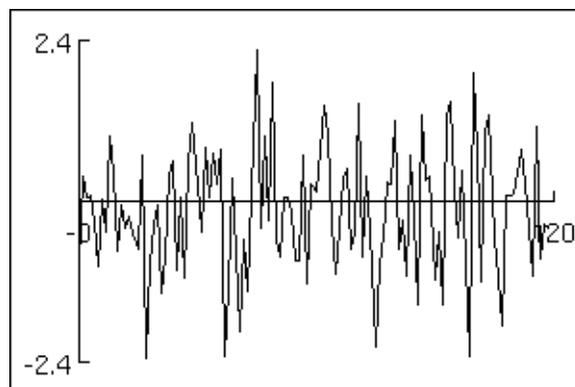
For more information on...	See...	In this book...
Reading the Output Value		<i>Reference Manual</i>
Specifying How Often to Generate Values		<i>Reference Manual</i>

White Noise



The White Noise block generates random values that are normally distributed around a mean. Its current output value does not depend on any previous output value. This block is especially useful when you add it to another signal to simulate noise.

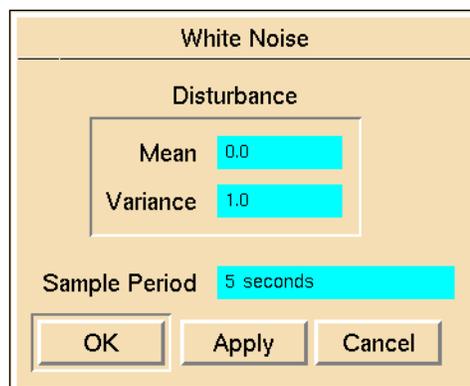
This figure shows the output from a White Noise block. The Disturbance Mean is 0 and the Disturbance Variance is 1.0.



To specify the range of output values, set the attributes Disturbance Mean and Disturbance Variance. Disturbance Mean is the mean output value, and Disturbance Variance is the variance of the output values.

Configuring

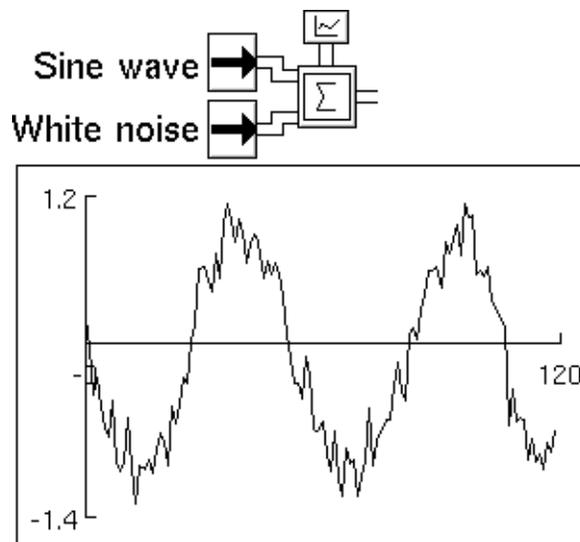
This is the configuration panel for the White Noise block.



Attribute	Description
Disturbance Mean	The mean output value of the block.
Disturbance Variance	The variance of the output values of the block.
Sample Period	How often the block passes a new value.

Example

The White Noise block in this figure adds noise to a Sine Wave signal. The Disturbance Mean is 0.0 and the Disturbance Variance is 0.025.



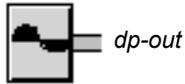
See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Overriding Block Values		<i>User's Guide</i>
Controlling the Flow of Data in an Application		<i>User's Guide</i>

For more information on...	See...	In this book...
Reading the Output Value		<i>Reference Manual</i>
Specifying How Often to Generate Values		<i>Reference Manual</i>

Sine Wave



The Sine Wave block generates sine values. It is a periodic signal generator and repeats its pattern of outputs cyclically.

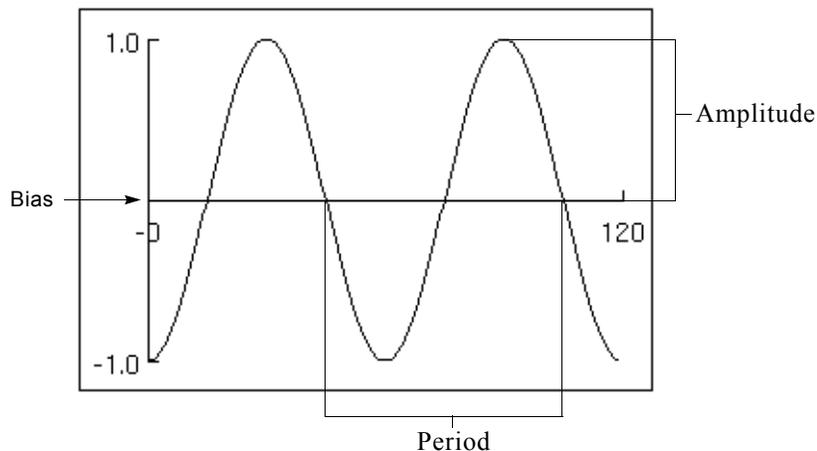
Specifying the Shape

The attributes Amplitude, Bias, and Period specify the shape of the sine wave. The Amplitude is half the difference between the minimum and maximum values. The Bias is the mean value between the minimum and the maximum. The Period is the amount of time that the block takes to complete a cycle.

Specifying a Phase

The Phase Angle determines where the Sine Wave block starts its cycle. It is a number of degrees between 0 and 360. For example, a Sine Wave block with a Phase Angle of 0 starts its cycle at the Bias going towards the maximum value. A Sine Wave block with a Phase Angle of 90 starts its cycle at the maximum value.

This figure shows a graph of a sine wave with a Period of 60, an Amplitude of 1.0, and a Bias of 0.0, and a Phase of 270.



Note Another way to change the place where the block starts its cycle is to set the attribute Reset Phase to **yes** and choose **reset** from the block's menu. For more information see "Resetting" below.

Resetting

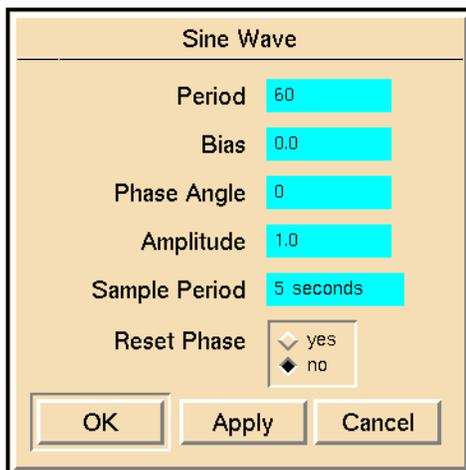
The attribute Reset Phase determines what happens when you choose **reset** from the block's menu:

If Reset Phase is...	The block does this when you choose reset ...
yes	Returns to the beginning of the wave's cycle.
no	Continues as before.

If you change Phase Angle as NOL is running, NOL uses the new value the next time you reset the block.

Configuring

This is the configuration panel for the Sine Wave block with its default values.



Attribute	Description
Period	The amount of time that the block takes to complete a cycle.
Bias	The value between the minimum and the maximum.
Phase Angle	A number of degrees between 0 and 360, which determines where the Sine Wave block starts its cycle.

Attribute	Description
Amplitude	Half the difference between the minimum and maximum values.
Sample Period	How often the block passes a new value.
Reset Phase	Whether the block returns to the beginning of the cycle when reset (the default) or continues where the signal left off.

See Also

For more information on how to use this block, see the sections below

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Overriding Block Values		<i>User's Guide</i>
Specifying Initial Values		<i>User's Guide</i>
Controlling the Flow of Data in an Application		<i>User's Guide</i>
Reading the Output Value		<i>Reference Manual</i>
Specifying How Often to Generate Values		<i>Reference Manual</i>

Path Displays

Describes the blocks that display the value on paths

Introduction 41

Data Path Display 42

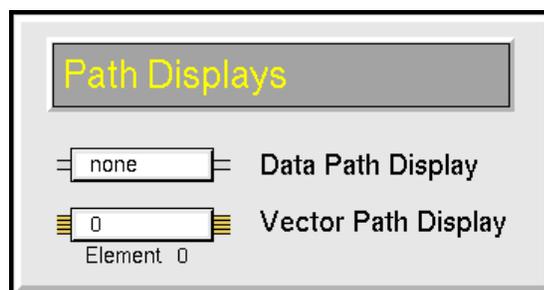
Vector Path Display 44



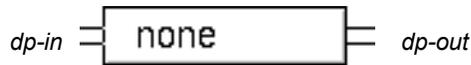
Introduction

NeurOn-Line provides you with two blocks that let you view the attributes of a data path or vector path.

You can find the Path Displays palette under the Entry & Paths submenu of the Palettes menu:



Data Path Display

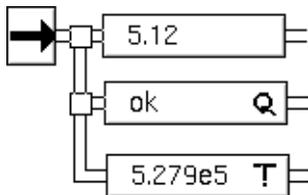


The Data Path Display block displays the value, quality, or collection time of its input value. By default, it displays the value. When it is displaying quality, a *Q* appears on the icon. When it is displaying collection time, a *T* appears on the icon.

Determining Which Path Attribute to Display

Use the menu choices *show collection time*, *show value*, and *show quality*, to display the path's *Collection-time*, *Data-value*, and *Quality* attributes, respectively.

The diagram below uses three Data Path Displays to display the value, quality, and collection time of one Entry Point.

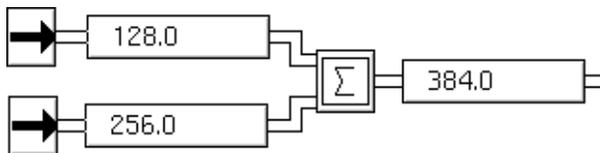


Configuring

The Data Path Display block has no configurable attributes.

Example

In the diagram below, Data Path Displays show an addition in progress.

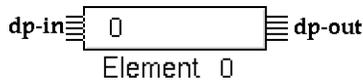


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Displaying the Value on a Path		<i>User's Guide</i>

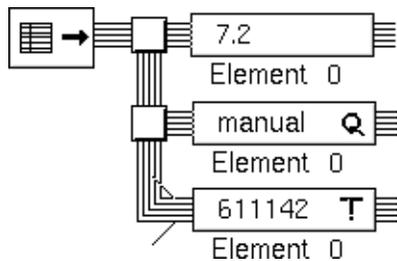
Vector Path Display



The Vector Path Display block displays the value of one element in its input vector, or the vector's quality or collection time. By default, it displays the element's value. When it's displaying quality, a Q appears on the icon. When it's displaying collection time, a T appears on the icon.

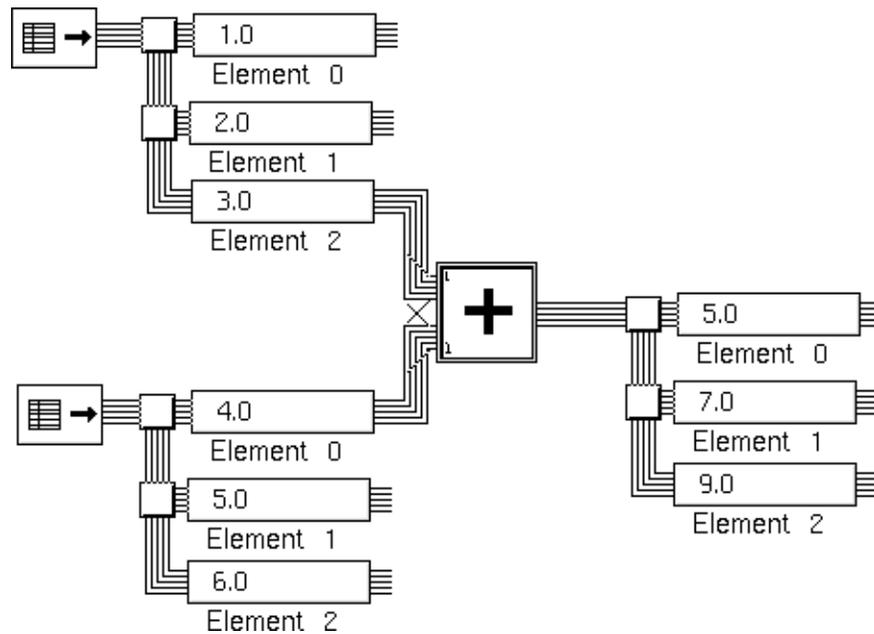
If you want to change which attribute it displays, choose an option from the block's menu. To display quality, choose show quality. To display collection time, choose show collection time. To display value, choose show value.

The diagram below uses three Data Path Displays to display the value, quality, and collection time of the first element in a Vector Entry Point.



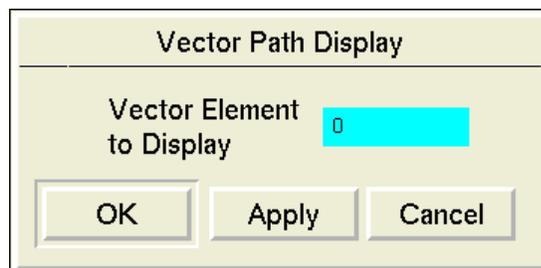
To change which element it displays, change the attribute Vector Element to Display in the block's configuration panel. To show all the elements in a vector, connect several Vector Path Displays to a path and change the Vector Element to Display attribute for each. Alternatively, choose the show value menu choice on the vector path to display a spreadsheet that shows the entire vector.

In the diagram below, Vector Path Displays show a vector addition in progress. Note that under normal circumstances, you should not use so many path displays, because doing so would significantly degrade the efficiency of your application.



Configuring

This is the configuration panel for Vector Path Display.



Field	Description
Vector Element to Display	Specifies which element in the input vector to display.

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Displaying the Value on a Path		<i>User's Guide</i>
Using Vector Paths		<i>User's Guide</i>

Connections

Describes the NeurOn-Line objects that control how data flows through the various types of paths.

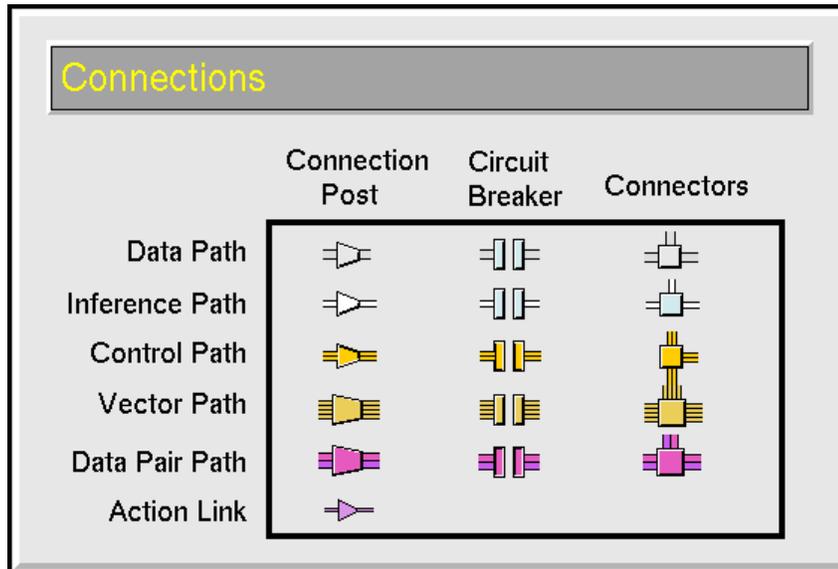
Introduction	47
Connection Posts	49
Connectors	53
Circuit Breakers	55



Introduction

The Connections palette contains connections that control how information flows through paths.

You find the Connections palette on the Other submenu of the Palettes menu:



These objects do not modify data or perform an action, but they let you control how information flows through paths. You can let paths cross workspaces, join paths together, and create loops.

The Connection Posts let paths cross workspace boundaries so you can create diagrams that span several workspaces.

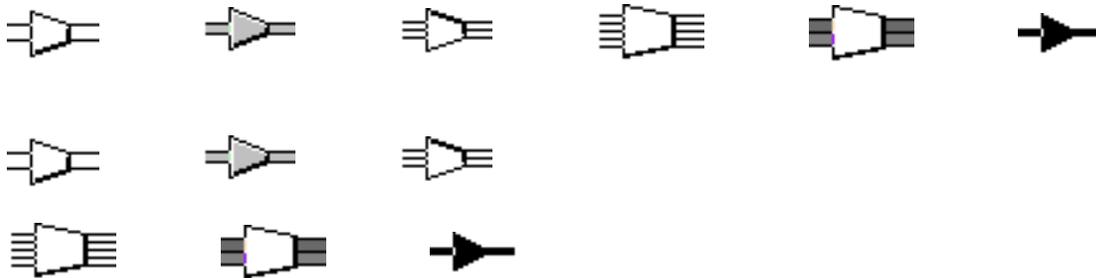
The Connectors allow you to join paths with a branch-in topology. They pass the last value received.

NOL allows you to create diagrams containing loops. You use circuit breakers to prevent NOL from entering an infinite loop. The Circuit Breakers let you control loops in your diagram.

The Connection Posts, Connectors, and Circuit Breakers associated with Item Paths behave the same way as the connections associated with the other types of paths, except they pass items. For more information, see [Connection Posts](#), [Connectors](#), and [Circuit Breakers](#).

For more information on item paths, see [Using Item Paths](#) in the *NeurOn-Line User's Guide*, [Customizing the Connection Stubs](#) in the *NeurOn-Line User's Guide*, and the *GDA API Reference*.

Connection Posts



Connection Posts let paths cross workspace boundaries so you can create diagrams that span several workspaces. G2 passes the data from an output connection post to all input connection posts with the same name. NeurOn-Line has six types of Connection Posts: Data Path Connection Posts, Inference Path Connection Posts, Control Path Connection Posts, Vector Path Connection Posts, Data Pair Path Connection Posts, and Action Link Connection Posts. They are identical except for the type of data they handle.

A Connection Post must have a name to work. An output connection post and its corresponding input connection posts must have the same name to pass data properly. To name the connection post, configure the connection post.

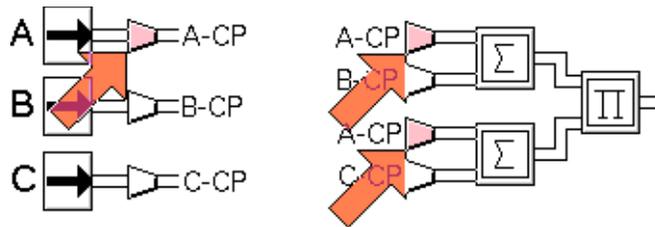
The Connection Posts on the Connections palette can function as input or output. To change one to an input post, just delete the input stub by dragging it onto the post, and vice versa for an output post.

Connection Posts let you break up a complex diagram and put related parts of the diagram on different workspaces. You can also use input and output connection posts on the same workspace to reduce the number of crossed paths.

Note Do not modify the Superior-connection attribute of a connection post.

Highlighting

If your diagram is large and contains many Connection Posts, you may want to see the output Connection Post that sends data to another post, or you may want to find all the input posts that receive data from another post. To highlight all connection posts with the same name, choose **highlight** from a Connection Post's menu. NOL brings to the front all workspaces that contain a connection post with that name, colors the post the block highlight color (pink, by default), and places an arrow beside it.



To change the post's color back and remove the arrow, choose do not highlight from the post menu for any of the highlighted posts.

Configuring

This is the configuration panel for a Data Path Connection Post:

Data Path Connection Post

Name none

OK
Apply
Cancel

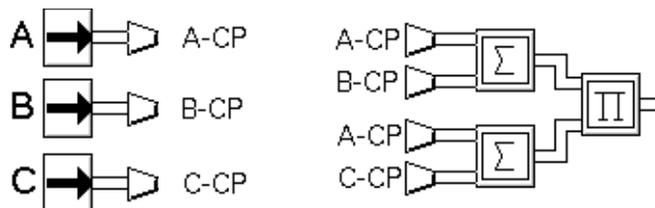
Attribute	Description
Name	The name of the connection post.

Examples

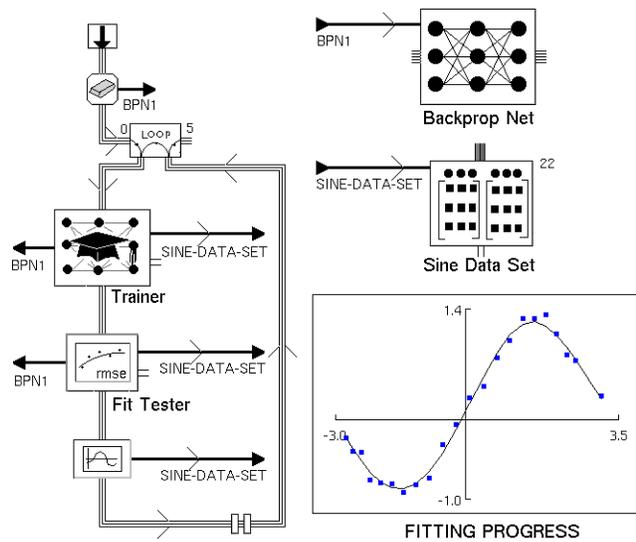
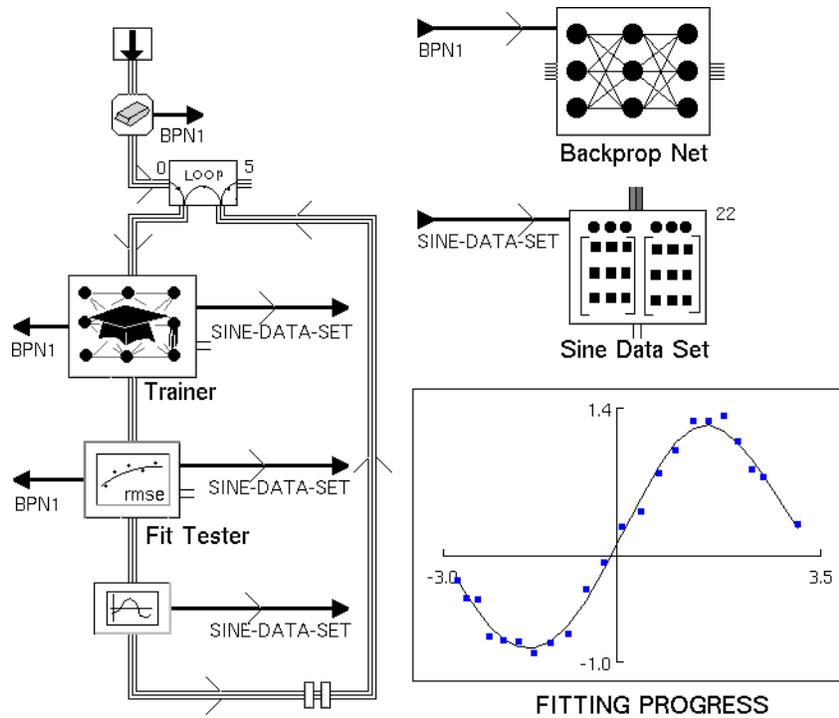
The following figure shows the equation:

$$(a + b)(a + c)$$

The Connection Posts let you use the value from Entry Point A in two different places, without crossing paths.



The example below uses action link Connect Posts to simplify a diagram in which a Backpropagation is trained and tested with a Data Set:

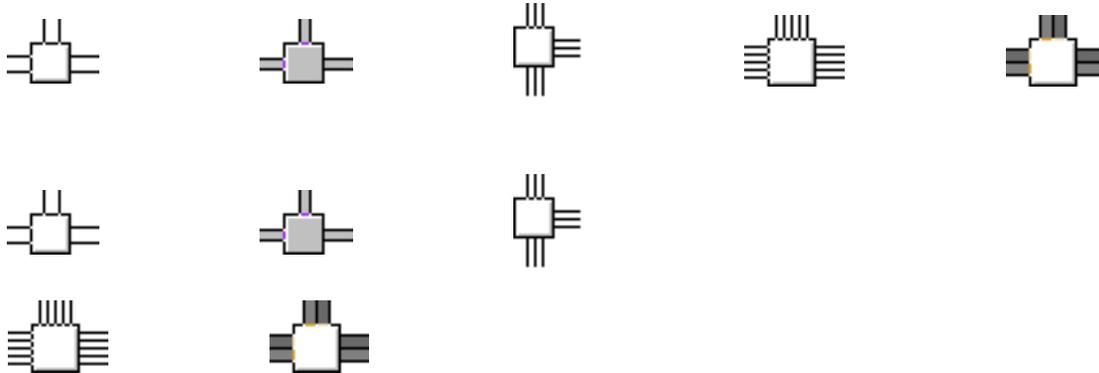


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Using Paths		<i>User's Guide</i>

Connectors



A Connector lets you combine two output paths, so a block's input port can get a value from either path. A connector passes the last value received on any of its input paths. NeurOn-Line has five types: Data Path Connectors, Inference Path Connectors, Control Path Connectors, Vector Path Connectors, and Data Pair Path Connectors.

The various types of Connectors are nearly identical, except for the type of data they handle.

Connectors are similar to splitters, which split an output path into two paths, so that one output path can enter two ports. G2 creates splitters for you automatically whenever you connect an input path to another path.

You can drag additional paths into a connector, similar to a peer input block.

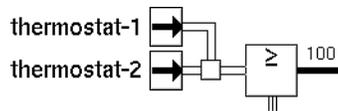
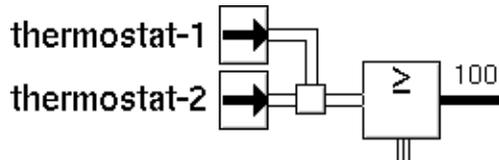
Note G2 does not create Connectors automatically. You must create them yourself by cloning them from the Connections palette. If you try to connect an output path to another path without a Connector, G2 deletes the path and generates a warning message.

Configuring

The Connectors have no configurable attributes.

Example

The following diagram checks the temperature of some equipment. The temperature can come from one of two Entry Points. A Connector combines the output paths from the Entry Points, so the High Value Observation can get input from either.



See Also

For more information on how to use this block, see the sections below.

For more information on...

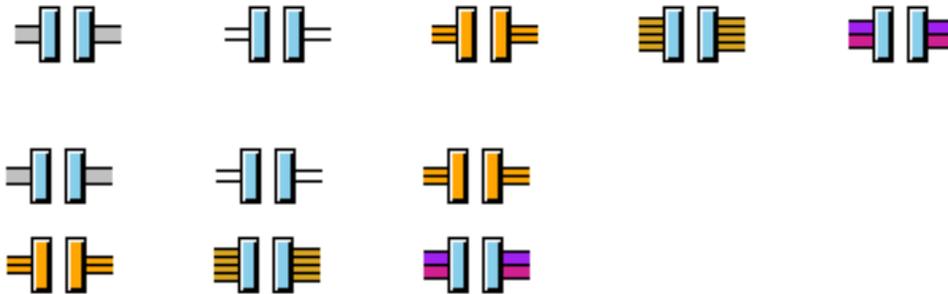
See...

In this book...

Basic Block Behavior

User's Guide

Circuit Breakers

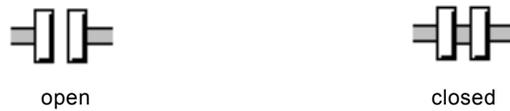


Circuit Breakers allow you to control a NOL diagram with a loop. They are not required if another block can halt the iteration of the loop. Otherwise, you use the circuit breaker to prevent infinite loops in a diagram.

When the Circuit Breaker is open, the value on the path is propagated across the breaker unchanged, but NOL does not evaluate the downstream block immediately. The next block evaluates when it receives data from somewhere else. When the Circuit Breaker is closed, the value on the path is propagated across the breaker, and the downstream block evaluates immediately.

Circuit Breakers contain a menu choice for opening and closing the circuit breaker. If the circuit breaker is open, the menu choice is **close breaker**, and if the circuit breaker is closed, the menu choice is **open breaker**.

Opening and closing the circuit breaker also changes the icon as shown in the following figure:



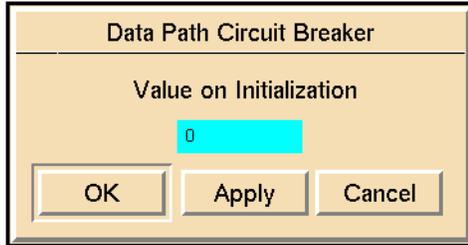
⚠ All types of circuit breakers are open by default.

Configuring

Circuit Breaker configuration panels vary depending on the type of circuit breaker.

Configuring the Data Path Circuit Breaker

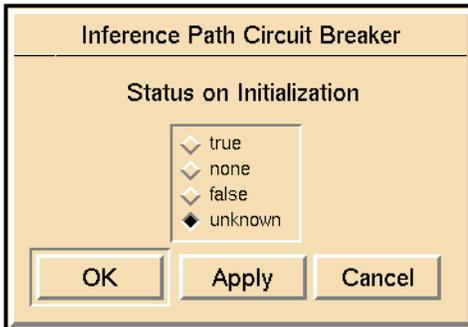
This is the configuration panel for the Data Path Circuit Breaker.



Attribute	Description
Value on Initialization	See Specifying an Initial Data Value in the <i>GDA User's Guide</i> .

Configuring the Inference Path Circuit Breaker

This is the configuration panel for the Inference Path Circuit Breaker.



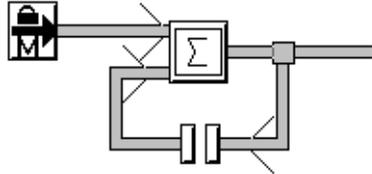
Attribute	Description
Status on Initialization	See Specifying an Initial Status Value in the <i>GDA User's Guide</i> .

Configuring the Control Path, Vector Path, and Data Pair Path Circuit Breakers

The Control Path Circuit Breaker, Vector Path Circuit Breaker, and Data Pair Path Circuit Breaker have no configurable attributes.

Example

In the following example, the Circuit Breaker is open, which allows the Summation block to accumulate values from the Numeric Entry Point each time a new value appears on the path. Do not close the Circuit Breaker in this situation.



See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Specifying Initial Values		<i>User's Guide</i>
Understanding the NOL Block Evaluation Engine		<i>User's Guide</i>

Data Processing

Chapter 4: Scalar Blocks

Describes blocks that operate on scalar values, such as by performing arithmetic, averaging, filtering, and delaying functions.

Chapter 5: Vector Blocks

Describes the blocks that create, manipulate, and operate on vectors.

Chapter 6: Data Set Blocks

Describes the blocks that store and manipulate the data with which you train a neural network.

Scalar Blocks

Describes blocks that operate on scalar values, such as by performing arithmetic, averaging, filtering, and delaying functions.

Introduction	62
Summation	66
Difference	68
Change Sign	69
Bias	70
Multiplication	72
Quotient	74
Inverse	75
Gain	76
Additive Noise	78
Outlier Filter	80
First-Order Exponential Filter	84
Sample Median	87
Average Input Value	89
Median Input Value	91
Data Delay	93
Data Inhibit	95
Data Output	97
Set Attribute	99
Data Shift	101

Variance 103

Moving Average 105

Arithmetic Function 107

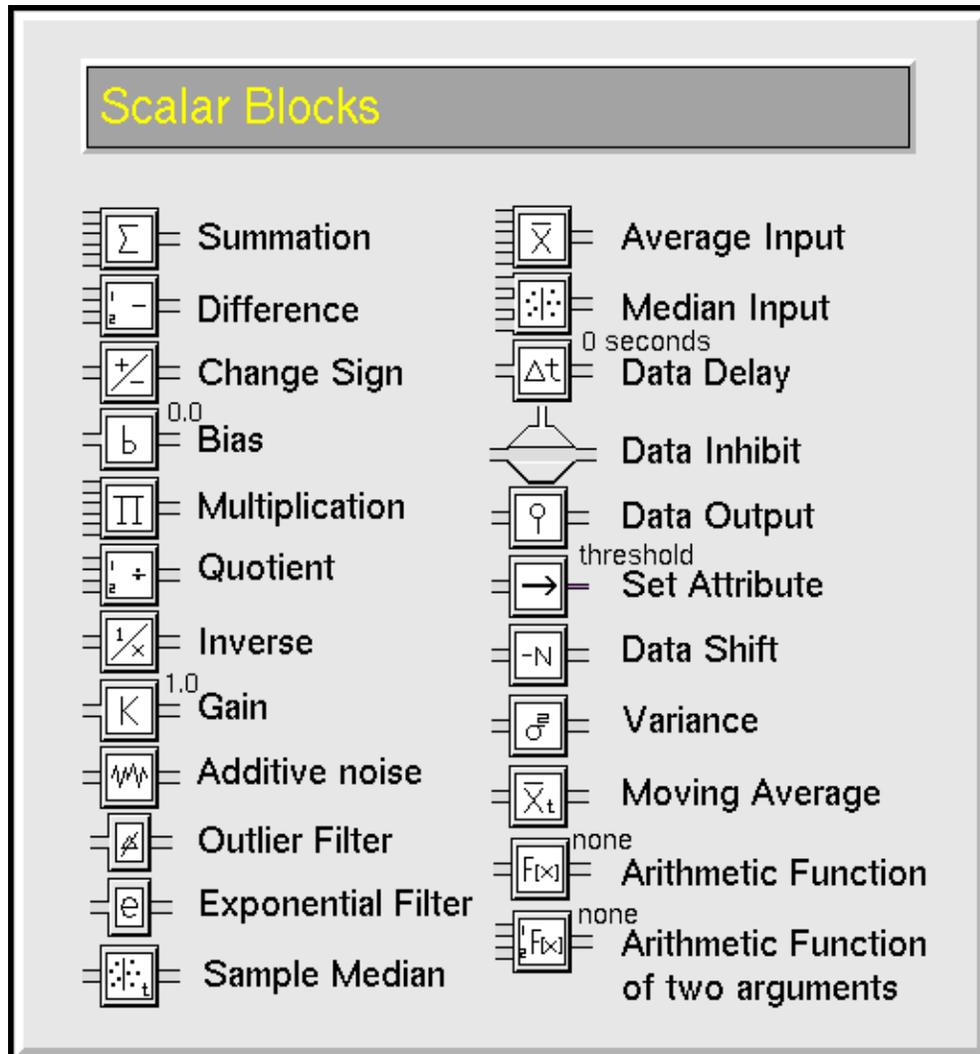
Arithmetic Function of Two Arguments 111



Introduction

NeurOn-Line provides a number of blocks that let you perform operations on scalar values, such as arithmetic, averaging, filtering, delaying, and applying your own functions.

You can find the Scalar Blocks palette under the Data Processing submenu of the Palettes menu:



Performing Arithmetic Operations

These blocks let you add, subtract, multiply, and divide data:

- The [Summation](#) block adds any number of input values.
- The [Difference](#) block subtracts one value from another.
- The [Change Sign](#) block inverts the sign of a values.
- The [Bias](#) block adds a constant value to its input.
- The [Multiplication](#) block multiplies any number of input values.

- The [Quotient](#) block divides one input value by another.
- The [Inverse](#) block passes the inverse of a value.
- The [Gain](#) block multiplies its input by a constant value.

Adding and Filtering Noise

- The [First-Order Exponential Filter](#) block performs low-pass filtering and is useful for noise reduction.
- The [Outlier Filter](#) block passes only those values that stay inside the range. This filter is useful for ignoring large sampling errors and impossibly extreme sensor readings.

Averaging Values

These blocks compute the average or median value of their input values:

- The [Average Input Value](#) block computes the average value of any number of input values.
- The [Median Input Value](#) block computes the median value of any number of input values.
- The [Moving Average](#) block computes the average of the history of values.

Stopping and Pausing Data

These blocks let you stop or pause the flow of data:

- The [Data Delay](#) block holds a value for a set period of time.
- The [Average Input Value](#) block stops data propagation as long as an inference input has a given value.
- The [Data Shift](#) block delays passing its input value until it's received more input values.

Outputting Data

These blocks let you set a block's attribute, a G2 parameter, or a G2 variable to the input data value:

- The [Set Attribute](#) block sets another block's attribute to the input data value.
- The [Data Output](#) block sets a G2 parameter or variable to the input data value.

Computing Statistical Properties

The [Variance](#) block computes the variance of the history of values.

Defining Your Own Function

These blocks let you apply your own function or a built-in G2 function to its input value:

- The [Arithmetic Function](#) block lets you apply a function with one argument.
- The [Arithmetic Function of Two Arguments](#) block lets you apply a function with two arguments.

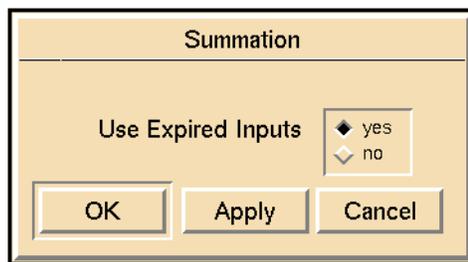
Summation



The Summation block adds all its input values together and passes the result. This block is a peer input block; you can add inputs by dragging additional input paths into the block.

Configuring

This is the configuration panel for the Summation block.



Attribute	Description
Use Expired Inputs	See Determining Whether a Block Uses Expired Inputs in the <i>NeurOn-Line User's Guide</i> .

Example

This figure shows a diagram that computes $(a + b) - (c + d)$. Note that the Summation block has an extra input port and has been enlarged with the Change Size menu choice.

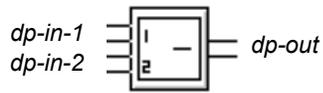
See Also

For more information on how to use this block, see the pages below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

For more information on...	See...	In this book...
Connecting to Peer Input Blocks		<i>User's Guide</i>
Vector Sum		<i>Reference Manual</i>

Difference



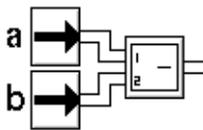
The Difference block subtracts the value of the lower path from the value of the upper path and passes the result.

Configuring

The Difference block has no configurable attributes.

Example

This figure shows the equation $a - b$.

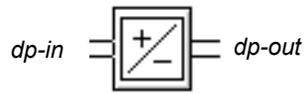


See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Vector Difference		<i>Reference Manual</i>

Change Sign



The Change Sign block changes the sign of the input value, as if you multiplied it by -1 . This block makes negative values positive and positive values negative.

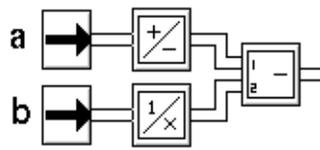
Configuring

The Change Sign block has no configurable attributes.

Example

This figure shows a diagram that computes this equation:

$$-a - \frac{1}{b}$$



The block marked “ $1/x$ ” is an [Inverse block](#). The block marked “ $-$ ” is a [Difference block](#).

See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Vector Rescaler		<i>Reference Manual</i>

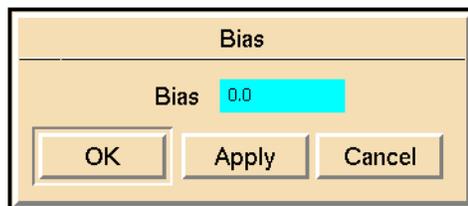
Bias



The Bias block adds a fixed constant to its input value. Specify the constant in the block's Bias attribute.

Configuring

This is the configuration panel for the Bias block.



Attribute	Description
Bias	The value that the block adds to its input.

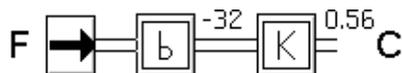
Example

This figure shows the equation for converting temperatures from Fahrenheit to Celsius:

$$^{\circ}\text{C} = \frac{^{\circ}\text{F} - 32}{1.8}$$

or

$$^{\circ}\text{C} = (^{\circ}\text{F} + (-32)) \times 0.56$$



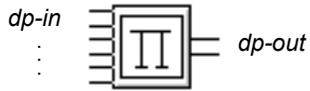
The block marked "K" is the [Gain block](#).

See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Editing Attribute Displays		<i>User's Guide</i>
Vector Rescaler		<i>Reference Manual</i>

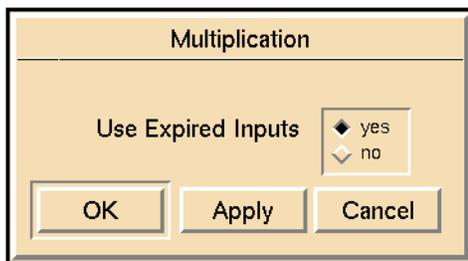
Multiplication



The Multiplication block multiplies all its input values and passes the result.

Configuring

This is the configuration panel for the Multiplication block.



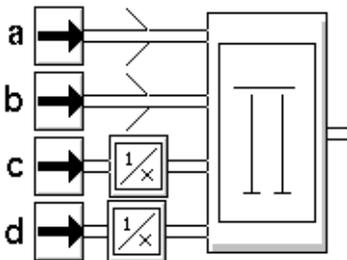
Attribute	Description
Use Expired Inputs	See Determining Whether a Block Uses Expired Inputs in the <i>NeurOn-Line User's Guide</i> .

Example

This figure shows a diagram that computes this equation:

$$\frac{ab}{cd}$$

The Multiplication block has an extra input port and has been enlarged with the Change Size menu choice.

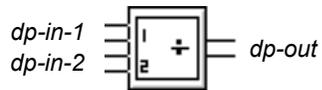


See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Connecting to Peer Input Blocks		<i>User's Guide</i>
Vector Product		<i>Reference Manual</i>

Quotient



The Quotient block divides the value of the upper path by the value of the lower path and passes the result.

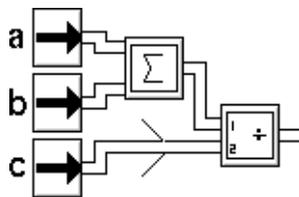
Configuring

The Quotient block has no configurable attributes.

Example

This figure shows this equation:

$$\frac{a + b}{c}$$

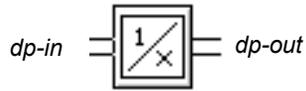


See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Vector Quotient		<i>Reference Manual</i>

Inverse



The Inverse block computes the multiplicative inverse of the input value.

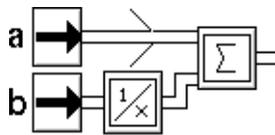
Configuring

The Inverse block has no configurable attributes.

Example

This figure shows a diagram that computes this equation:

$$a + \frac{1}{b}$$



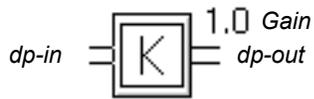
The block marked “ Σ ” is a [Summation block](#).

See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Vector Quotient		<i>Reference Manual</i>
Vector Rescaler		<i>Reference Manual</i>

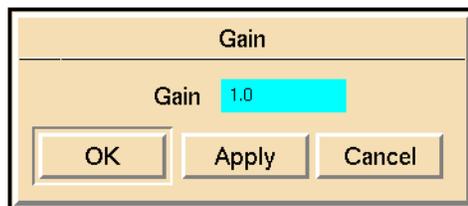
Gain



The Gain block multiplies its input value by a fixed constant. Specify the constant in the block's Gain attribute.

Configuring

This is the configuration panel for the Gain block.

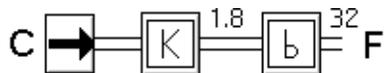


Attribute	Description
Gain	The value that the block multiplies by its input.

Example

This figure shows the equation for converting temperatures from Celsius to Fahrenheit using this equation:

$$^{\circ}\text{F} = ^{\circ}\text{C} \times 1.8 + 32$$



The block marked "b" is the [Bias block](#).

See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

For more information on...	See...	In this book...
Editing Attribute Displays		<i>User's Guide</i>
Vector Quotient		<i>Reference Manual</i>

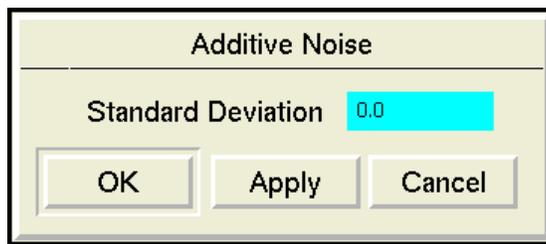
Additive Noise



The Additive Noise block adds a random value to its input and passes the sum. The random value is normally distributed around zero. Specify the standard deviation for the random value in block's configuration panel.

Configuring

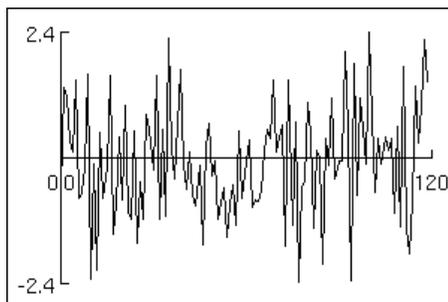
This is the configuration panel for the Additive Noise block.



Attribute	Description
Standard Deviation	The standard deviation for the random values that this block produces.

Example

The figure below shows the output of an Additive Noise block that is connected to a Numeric Entry Point that passes 0. The Standard Deviation for the Additive Noise block is 1.0.



See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

Outlier Filter



The Outlier Filter passes only values that are within a band you specify. Specify the band with the attributes Band Center and Band Range. If Outlier Replacement is **yes**, the filter replaces out-of-range values with the nearest in-range value. If Outlier Replacement is **no**, the filter discards out-of-range values.

The band can be fixed or floating. If Band Type is **floating**, each time the block passes a value, that value becomes the center of the range. If Band Type is **fixed**, the range always remains the same.

Specifying a Range

The attribute Band Range specifies the size of the block's range. Set it to the difference between the highest and lowest values in the range. The filter's range contains the values between the range's center minus Band Range/2 and the range's center plus Band Range/2.

The attribute Band Type controls how the range's center is set. If Band Type is **fixed**, you set the attribute Band Center and NOL uses the value of Band Center as the center of the range.

If Band Type is **floating**, NOL uses the last value it passed as the center, and NOL ignores the value of the attribute Band Center. Because the block always passes in-range values, the center is always set to an in-range input value.

When the block receives an out-of-range value, what it passes depends on the value of Outlier Replacement, the attribute which controls whether the block replaces an out-of-range value with another value. If Outlier Replacement is **yes**, it changes the center of the range to the in-range value nearest to the input value. If Outlier Replacement is **no**, the center does not change.

Note If Band Range is **none**, NOL uses a Band Range of 0.0. If Band Center is **none**, NOL proceeds as if Band Type were **floating**.

Specifying How to Round Output Values

To round the output values, set the field Quantization. The block rounds its output value to the unit you specify. For example, if Quantization is 0.1, the block rounds to the nearest tenth, and if Quantization is 1.0, the block rounds to the nearest integer.

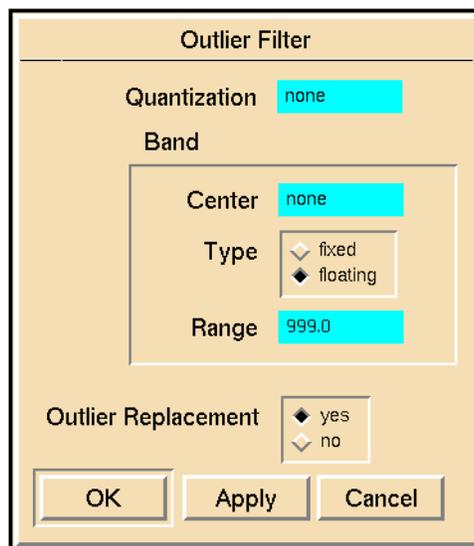
This table shows some examples of rounding.

If Quantization is...	The filter passes these values...		
none	0.5	1.43	1.77
0.1	0.5	1.4	1.8
0.5	0.5	1.5	2.0
1.0	1.0	1.0	2.0

This field is especially useful when you need to keep output values within known accuracy limits.

Configuring

This is the configuration panel for the Outlier Filter.



Attribute	Description
Quantization	A number that specifies the smallest increment that the block uses when rounding output values.
Band Center	The center of the band when Band Type is fixed; otherwise, this attribute is ignored.

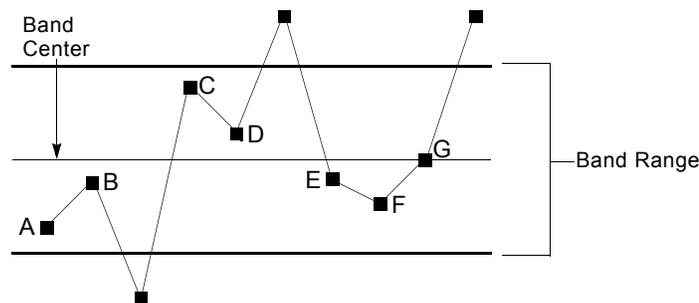
Attribute	Description
Band Type	How the center of the range is set. The values are either fixed , in which case the block uses the value of Band Center as the center, or floating , in which case the current output value is the new band center.
Band Range	The extent of the band's range.
Outlier Replacement	When Band Type is floating , whether the block replaces the band center with the nearest in-range value when the block receives an out-of-range value (yes), or whether the center remains the same (no).

Examples

This section has some examples of a fixed and floating Outlier Filter.

Fixed Band

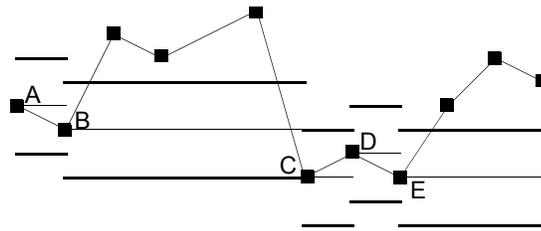
This figure shows some input values for an Outlier Filter with Band Type set to **fixed**. The Band Range and Band Center are marked. Letters mark values that are passed without replacement. If Outlier Replacement is **yes**, the filter replaces unmarked values with the nearest in-range value. Otherwise, the filter does not pass any value.



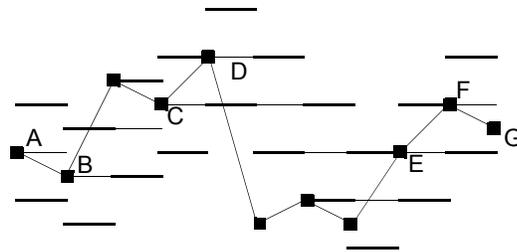
Floating Band

The next two figures show some input values for an Outlier Filter with Band Type set to **floating**. They differ in how Outlier Replacement is set. In both, the range is marked with a bold line, and the center of the range is marked with a thin line.

In this figure, Outlier Replacement is **no**. Notice that the center of the range changes only when a new input value falls in the same range as the previous value. Letters mark values that the block passes.



In this figure, Outlier Replacement is **yes**. Notice that when a value is out of range, the center of the new range becomes the maximum or minimum of the old range. Letters mark values that the block passes without replacing. For unmarked values, the filter passes the closest in-range value.

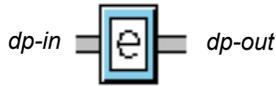


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Pair Outlier Filter		<i>Reference Manual</i>
Novelty Filter		<i>Reference Manual</i>

First-Order Exponential Filter



The First Order Exponential Filter performs low-pass filtering to filter out high frequency noise. Its output value depends on the previous output value and the current input value. To specify how much the filter weights previous output values, set the attribute Filter Constant. The larger the Filter Constant, the more weight the previous output values have.

This filter does not require inputs that are equally spaced in time.

Filtering

The First-Order Exponential Filter uses this equation to compute its output value. The variables in the equation are explained in the table that follows.

$$\text{output}_n = \alpha \cdot \text{input} + (1 - \alpha)\text{output}_{n-1} \text{ where } \alpha = e^{-\frac{\Delta \text{time}}{\text{Filter-constant}}}$$

This variable...	Is...
output_n	The current output value
output_{n-1}	The previous output value
input	The current input value
Δtime	The difference in time between the arrival of the last input value and the current input value

The coefficient α is the amount of weight that the filter gives the current input, and the coefficient $(1-\alpha)$ is the amount of weight that the filter gives the previous output value. The block computes α with both the Filter Constant and the difference in time between the arrival of the last input and the arrival of the current input. The constant α is computed such that values received long ago will not have as much effect on the current value as ones that were received a short time ago.

Specifying How to Round Output Values

To round the output values, set the field Quantization. The block rounds its output value to the unit you specify. For example, if Quantization is 0.1, the block rounds to the nearest tenth, and if Quantization is 1.0, the block rounds to the nearest integer.

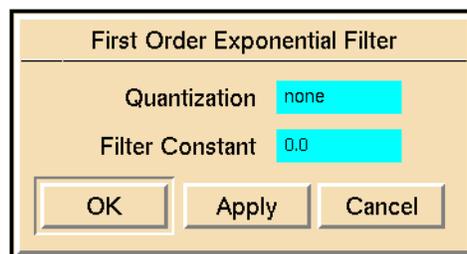
This table shows some examples of rounding.

If Quantization is...	The filter passes these values...		
none	0.5	1.43	1.77
0.1	0.5	1.4	1.8
0.5	0.5	1.5	2.0
1.0	1.0	1.0	2.0

This field is especially useful when you need to keep output values within known accuracy limits.

Configuring

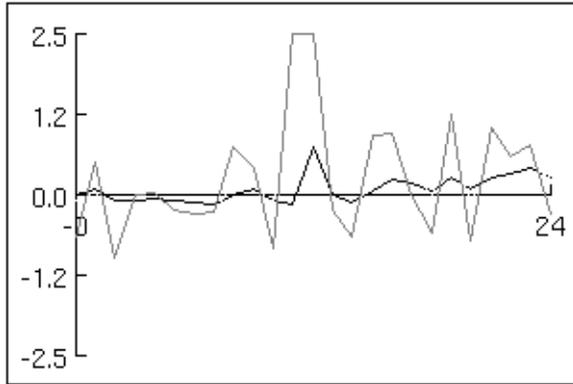
This is the configuration panel for the First Order Exponential Filter.



Attribute	Description
Quantization	A number that specifies the smallest increment that the block uses when rounding output values.
Filter Constant	The amount of weight that the filter gives the current input over the previous input.

Example

This figure shows the chart of a First-Order Exponential Filter with a Filter Constant of 5. The raw signal has more variation and the filtered signal is smoother.



See Also

For more information on how to use this block, see the sections below.

For more information on...

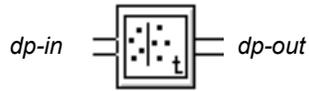
See...

In this book...

Basic Block Behavior

User's Guide

Sample Median



The Sample Median block passes the median of the history of its input values.

If the Sample Median block has an odd number of history values, it passes the middle value. If the block has an even number of history values, it passes the lesser of the two middle values.

Configuring

This is the configuration panel for the Sample Median block.

Attribute	Description
Sample Type and Sample Size	See Specifying the Size of the History in the <i>NeurOn-Line User's Guide</i> .
Update Type and Update Size	See Specifying When to Propagate Data in the <i>NeurOn-Line User's Guide</i> .
Erase History when Reset	See Specifying What Happens to History Upon Reset in the <i>NeurOn-Line User's Guide</i> .

Attribute	Description
Require Full History	See Specifying What to Do With Partial History in the <i>NeurOn-Line User's Guide</i> .
Value on Initialization	See Specifying an Initial Data Value in the <i>NeurOn-Line User's Guide</i> .

Example

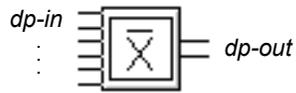
If the history of a Sample Median block has the five points 0.7, 0.9, 0.3, 0.7, and 0.5, it passes on the value 0.7, the middle value. If the history of a Sample Median block has the four points 0.3, 0.9, 0.6, and 0.8, it passes on the value 0.6, the lesser of 0.6 and 0.8.

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Maintaining a History of Values		<i>User's Guide</i>
The Median Input Value block		<i>Reference Manual</i>

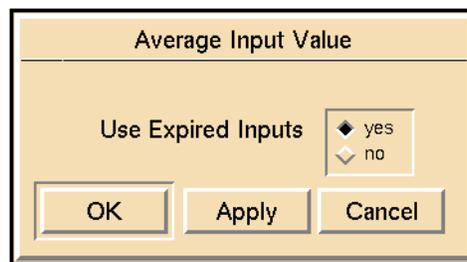
Average Input Value



The Average Input Value block passes the average of all its input values. It accepts any number of inputs.

Configuring

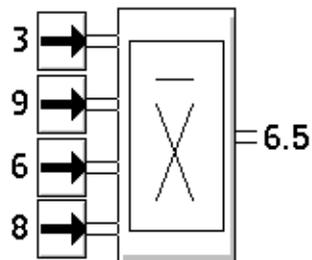
This is the configuration panel for the Average Input Value block.



Attribute	Description
Use Expired Inputs	See Determining Whether a Block Uses Expired Inputs in the <i>NeurOn-Line User's Guide</i> .

Example

This figure shows an Average Input Value block with the input values 3, 9, 6, and 8. It passes the value 6.5. Note that the Average Input Value block has an extra input port and has been enlarged with the **Change Size** menu choice.

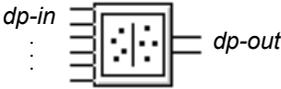


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Connecting to Peer Input Blocks		<i>User's Guide</i>
The Moving Average block		<i>Reference Manual</i>

Median Input Value

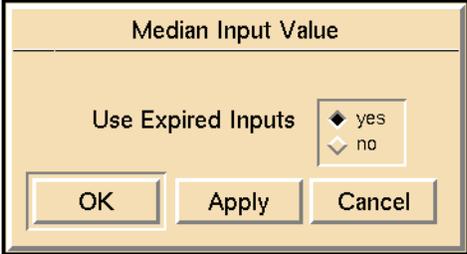


The Median Input Value block passes the median of all its input values. It accepts any number of inputs.

If the Median Input Value block has an odd number of input values, it passes the middle input value. If the block has an even number of input values, it passes the minimum of the two middle input values.

Configuring

This is the configuration panel for the Median Input Value block.

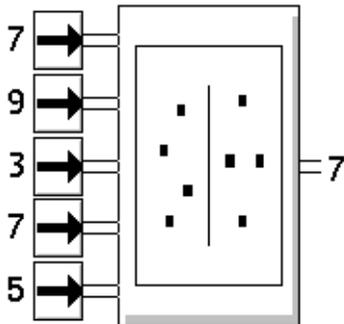


Attribute	Description
Use Expired Inputs	See Determining Whether a Block Uses Expired Inputs in the <i>NeurOn-Line User's Guide</i> .

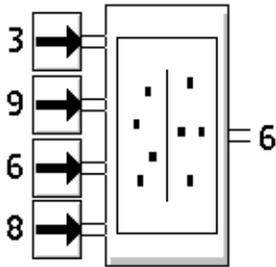
Example

The following figures are examples of how a Median Input Value block handles both odd and even numbers of inputs. In both cases, note that the Median Input Value block has extra input ports and has been enlarged with the Change Size menu choice.

This figure shows a Median Input Value block connected to five inputs. It passes 7. (The median of 3, 5, 7, 7, 9 is the middle value.)



This figure shows a Median Input Value block connected to four inputs. It passes 6. (The median of 3, 6, 8, 9 is the lesser of 6 and 8.)

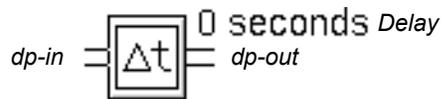


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Connecting to Peer Input Blocks		<i>User's Guide</i>

Data Delay



The Data Delay block delays passing its input value for a specified amount of time. It does not modify its input value. Specify the length of the delay with the Delay attribute.

Handling Multiple Signals

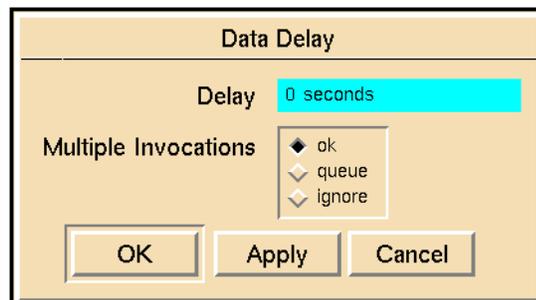
If Multiple Invocations is **ok**, the block can begin a delay while in the midst of delaying another value. If Multiple Invocations is **queue**, the second and subsequent delays will be started after the previous delay is finished, using the current input value. If Multiple Invocations is **ignore**, the block handles only one delay at a time, and inputs received during another delay are ignored. We recommend specifying Multiple Invocations for a Data Delay as either **ok** or **ignore**.

Resetting

When you reset a Data Delay block, values being delayed are lost.

Configuring

This is the configuration panel for the Data Delay block.



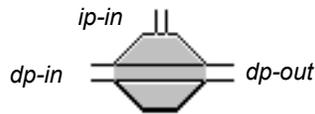
Attribute	Description
Delay	The amount of time to delay passing the value.
Multiple Invocations	See Specifying How to Handle Multiple Values in the <i>NeurOn-Line User's Guide</i> .

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Editing Attribute Displays		<i>User's Guide</i>

Data Inhibit



The Data Inhibit block lets an inference path turn a data path on and off.

When the status value of the block's input inference path matches the value of the attribute **Trigger On**, the block inhibits the input data value. If **Value on Initialization** has a value, it passes that value. Otherwise it passes nothing.

When the status value of the inference path no longer matches **Trigger On**, the block passes the current value of the input data path and continues to pass input data values normally.

If the block's inference path has not received a value yet (that is, it has a **quality of no-value**), the block passes nothing even when it receives a value from its data path.

NOL evaluates this block whenever the data path receives a new value or when the data inference path changes to or from the **Trigger On** value. It also evaluates the block when the data inference path changes from one non-trigger value to another, for example, from **false** to **unknown**.

Resetting

When you reset a Data Inhibit Block, the block does not pass a value until it receives a value from its inference input path, even if it receives a value from its input data path.

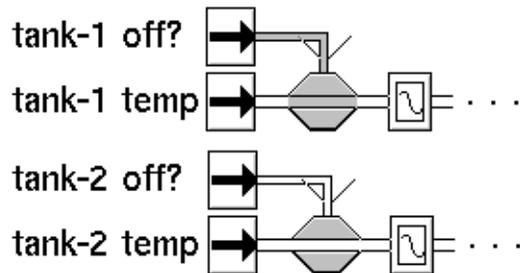
Configuring

This is the configuration panel for the Data Inhibit block.

Attribute	Description
Value on Initialization	See Specifying an Initial Data Value in the <i>NeurOn-Line User's Guide</i> .
Trigger On	The truth value that causes the block to inhibit data.

Example

This figure shows a portion of a flow diagram that uses two Data Inhibit blocks to test whether a tank is on before analyzing its temperature. Tank-1 is off and the path that crosses through the middle of the Data Inhibit block is filled in to show that it is inhibiting its input. Tank-2 is on and the middle of the Data Inhibit block is empty to show that the block is passing along its input.

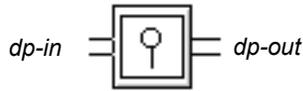


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
The Inference Inhibit block		<i>Reference Manual</i>
The Control Inhibit block		<i>Reference Manual</i>

Data Output



The Data Output block is obsolete, because you can connect variables and parameters directly to the output of any block. For information on how to do this, see [Using Variables and Parameters](#) in the *NeurOn-Line User's Guide*.

Use this block only when you cannot attach a data path directly to the target variable or parameter. For example, if the variable or parameter is an attribute of an object or if you want variables and parameters to appear on a separate workspace, use the Data Output block.

When you set the Data-server attribute in the table of the variable or parameter to inference engine, NOL uses the G2 conclude action to change values locally within NOL. When you set the Data-server attribute to anything other than inference engine, however, NOL uses the G2 set action to change values in the external system outside of G2.

Note You can set Target Variable to an expression that evaluates to a G2 variable or parameter. For more information, see [Evaluating Expressions in Attributes](#) in the *NeurOn-Line User's Guide*.

This block is especially useful for setting external setpoints or controlling external events.

The Entry Points perform the opposite action of the Data Output block: they pass information from G2 parameters and variables into NOL diagrams.

Configuring

This is the configuration panel for the Data Output block.

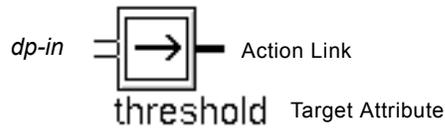
Attribute	Description
Target Variable	The name of the variable or parameter whose value the block updates.

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Using Variables and Parameters		<i>User's Guide</i>
Entry Points		<i>Reference Manual</i>

Set Attribute



The Set Attribute block sets the value of an attribute to the block's input data value. Specify the object whose attribute you would like to set by connecting the Set Attribute block's action link to it. You can connect the action link to any G2 object or block. Specify the attribute you would like to set by setting Target Attribute to its name.

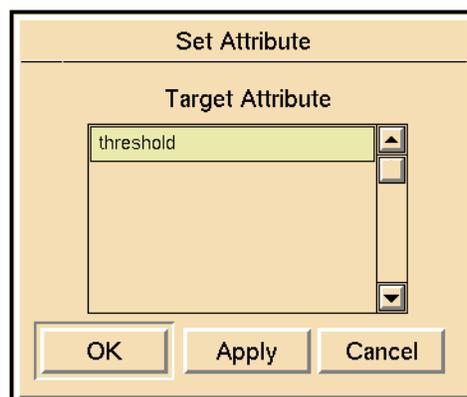
When you connect a block to the Set Attribute block's action link, the configuration panel displays a list of all attributes defined by the connected block. Thus, you should always connect the Action Link to the target *before* configuring the block.

Take the following precautions when using the Set Attribute block:

Caution When you use the Set Attribute block, be sure that the changes to the attribute value leave the target block in a valid state; otherwise, errors will result. For example, if you use the Set Attribute block to set the Upper Threshold of an observation block to a number that is less than the Lower Threshold, you will receive a warning and a runtime error might result.

Configuring

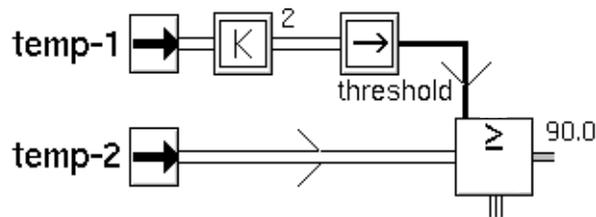
This is the configuration panel for the Set Attribute block.



Attribute	Description
Target Attribute	An attribute of the block connected to the action link, whose value the Set Attribute block sets.

Example

The following figure uses the Set Attribute block to test whether the temperature of tank-2 is more than twice the temperature of tank-1:



See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Editing Attribute Displays		<i>User's Guide</i>
Attribute Transfer		<i>Reference Manual</i>

Data Shift



The Data Shift block delays its input by a certain number of inputs; it waits to pass an input value until it has received more data. You specify how many more input values it must receive with the Sample Size attribute.

Specifying How to Delay Values

The block buffers its inputs in a first-in first-out queue of length Sample Size. When it receives a value, it deletes the value at the beginning of the queue and passes that value.

If the block does not have a value that satisfies its conditions, it passes no value. For example, when the block does not have Sample Size points yet, it outputs nothing.

Configuring

This is the configuration panel for the Data Shift block.

Attribute	Description
Sample Size	The number of points the block must receive before passing a value.

Attribute	Description
Value on Initialization	See Specifying an Initial Data Value in the <i>NeurOn-Line User's Guide</i> .
Erase History When Reset	See Specifying What Happens to History Upon Reset in the <i>NeurOn-Line User's Guide</i> .

Example

This table shows sample input and output for a Data Shift block with a Sample Size of 3. The first row is the values it received. The second row is the values it passed on.

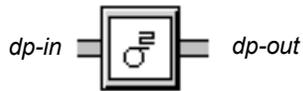
Input	1.3	2.3	4.5	5.0	7.2
Output				1.3	2.3

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Delay		<i>Reference Manual</i>

Variance



The Variance block passes on either the statistical variance or the standard deviation of the history of its input value. You can choose which it passes with the attribute Output as Std Deviation. If Output as Std Deviation is **yes**, the block passes the standard deviation. Otherwise, it passes the variance, which is the square of the standard deviation.

Configuring

This is the configuration panel for the Variance block.

Variance

<p style="text-align: center;">Sample</p> <div style="border: 1px solid gray; padding: 5px; margin-bottom: 5px;"> <p>Type <input checked="" type="radio"/> points <input type="radio"/> fixed <input type="radio"/> time</p> <p>Size <input style="width: 50px;" type="text" value="2"/></p> </div>	<p style="text-align: center;">Update</p> <div style="border: 1px solid gray; padding: 5px; margin-bottom: 5px;"> <p>Type <input checked="" type="radio"/> points <input type="radio"/> time</p> <p>Size <input style="width: 50px;" type="text" value="1"/></p> </div>
Output as Std Deviation	<input type="radio"/> yes <input checked="" type="radio"/> no
Require Full History	<input checked="" type="radio"/> yes <input type="radio"/> no
Erase History When Reset	<input type="radio"/> yes <input checked="" type="radio"/> no
Value on Initialization	<input style="width: 50px;" type="text" value="none"/>
<input type="button" value="OK"/> <input type="button" value="Apply"/> <input type="button" value="Cancel"/>	

Attribute	Description
Sample Type and Sample Size	See Specifying the Size of the History in the <i>NeurOn-Line User's Guide</i> .
Update Type and Update Size	See Specifying When to Propagate Data in the <i>NeurOn-Line User's Guide</i> .

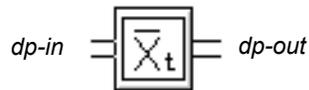
Attribute	Description
Output as Std Deviation	Whether the block outputs the value as a standard deviation (yes) or variance (no).
Require Full History	See Specifying What to Do With Partial History in the <i>NeurOn-Line User's Guide</i> .
Erase History when Reset	See Specifying What Happens to History Upon Reset in the <i>NeurOn-Line User's Guide</i> .
Value on Initialization	See Specifying an Initial Data Value in the <i>NeurOn-Line User's Guide</i> .

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Maintaining a History of Values		<i>User's Guide</i>

Moving Average



The Moving Average block passes the average of the history of its input values.

Configuring

This is the configuration panel for the Moving Average block.

Attribute	Description
Sample Type and Sample Size	See Specifying the Size of the History in the <i>NeurOn-Line User's Guide</i> .
Update Type and Update Size	See Specifying When to Propagate Data in the <i>NeurOn-Line User's Guide</i> .
Erase History when Reset	See Specifying What Happens to History Upon Reset in the <i>NeurOn-Line User's Guide</i> .

Attribute	Description
Require Full History	See Specifying What to Do With Partial History in the <i>NeurOn-Line User's Guide</i> .
Value on Initialization	See Specifying an Initial Data Value in the <i>NeurOn-Line User's Guide</i> .

Example

If the block's history contains 3, 0, 5, 8, 2, and 6, it passes on 4 or:

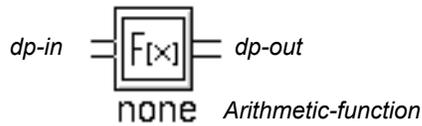
$$\frac{3 + 0 + 5 + 8 + 2 + 6}{6}$$

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Maintaining a History of Values		<i>User's Guide</i>
Average Input Value		<i>Reference Manual</i>

Arithmetic Function



The Arithmetic Function block lets you use a function or procedure as a block in a diagram. The block applies the function or procedure to its input value and passes the result. Specify the name of the function in the attribute Arithmetic Function.

You can use any:

- Built-in G2 function
- User-defined function
- Procedure
- Tabular-function

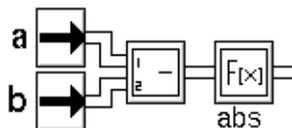
Note The input value for this block can be text, symbolic, or numeric.

Built-in G2 Function

You may set the attribute Arithmetic Function to any of these built-in G2 functions:

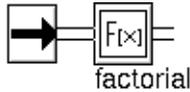
abs	arctan	ceiling	cos
exp	floor	int	ln
log	random	sin	sqrt
tan	truncate		

This figure shows a diagram that computes the absolute value of the difference of two values:



User-Defined Function

You may use any user-defined function that accepts one quantitative argument and returns one quantitative value. Set the attribute Arithmetic Function to the name of the function. This figure shows a diagram that figures the factorial of a value:



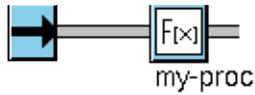
```
factorial (n) =  
  (if n <= 1  
   then 1  
   else n * factorial(n - 1))
```

Procedure

You can use a procedure that accepts three arguments, described in the following table, and returns a single value. Set the attribute Arithmetic Function to the name of your procedure. The block passes the procedure's return value as its output value. The block automatically passes the **Collection-time** and **Quality** of the block's input data path.

Argument	Type	Description
input-value	value	The block's input value.
collection-time	float	The Collection-time of the block's input path.
quality	symbol	The Quality of the block's input path.

This figure shows a diagram with an Arithmetic Function block that uses a procedure:



```

my-proc(input-value: float, collection-time:
float, quality: symbol) = (float)
delta: float;
begin
MY-PROC  delta = the current time - collection-time;
         if delta > 10 then
           return 0
         else if delta > 5 then
           return input-value/2
         else
           return input-value
         end
end
    
```

Tabular Function

You can use a `tabular-function-of-1-arg` that accepts one argument and returns a value. Set the attribute Arithmetic Function to the name of your function. The block passes its input value to the function as an argument and passes the function's return value as its output value. If the function cannot evaluate the input value, NOL signals an error. For information on a G2 `tabular-function-of-1-arg`, see the *G2 Reference Manual*.

This table shows a diagram with an Arithmetic Function block that uses a `tabular-function-of-1-arg`:

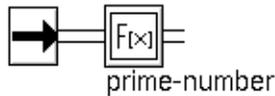
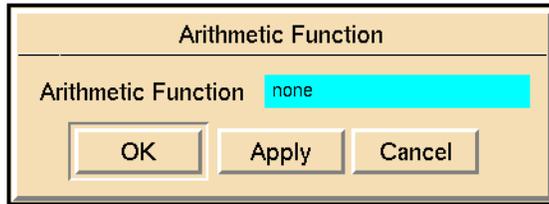


Table of values	
add or delete rows	
x	prime-number (x)
1	1
2	2
3	3
4	5
5	7
6	11

PRIME-NUMBER

Configuring

This is the configuration panel for the Arithmetic Function block.



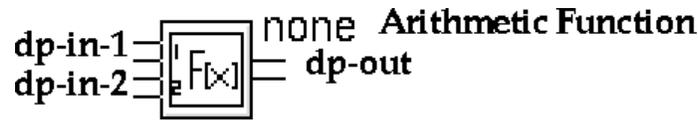
Attribute	Description
Arithmetic Function	The name of a G2 function that the block executes.

See Also

For more information on attributes and menu choices that are not described in this section, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Editing Attribute Displays		<i>User's Guide</i>
Custom Block Wizard		<i>User's Guide</i>
The Arithmetic Function of Two Arguments block		<i>Reference Manual</i>
Rule Action		<i>Reference Manual</i>

Arithmetic Function of Two Arguments



The Arithmetic Function of Two Arguments block lets you use a function or procedure as a block in a NeurOn-Line diagram. The block applies the function or procedure to its input values and passes a single result. Specify the name of the routine in the field Arithmetic Function.

You can use any:

- Built-in G2 function
- User-defined function
- Procedure

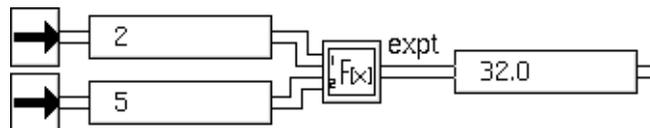
Note The input value for this block can be text, symbolic, or numeric.

Using Built-in G2 Function

You can set the field Arithmetic Function to any of these built-in G2 functions:

arctan average expt max
 min quotient random remainder

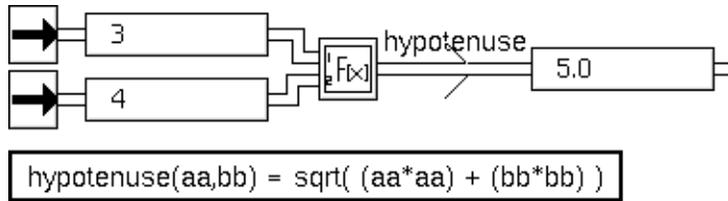
For example, this diagram computes 2^5 .



Using a User-Defined Function

You can use any user-defined function that accepts two quantitative arguments and returns one value. Set the field Arithmetic Function to the name of the function.

For example, this diagram uses the Pythagorean Theorem to figure the size of a right triangle's hypotenuse given the sizes of the other two sides.

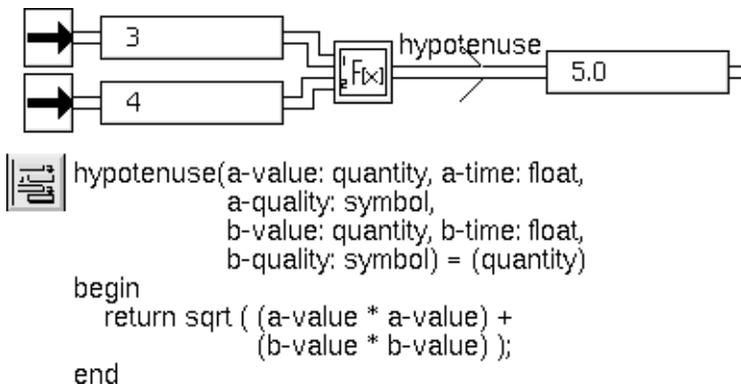


Using a Procedure

You can use a procedure that accepts six arguments, described below, and returns a single value. Set the field Arithmetic Function to the name of your function. The block passes the procedure's return value as its output value. The block automatically passes the collection-time and quality of the block's input data path.

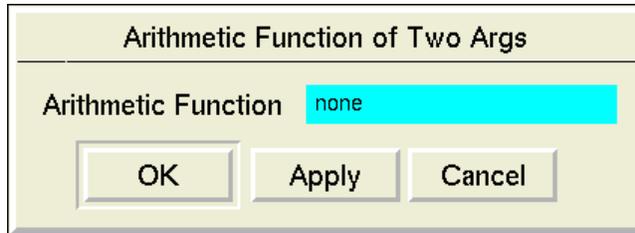
Argument	Type	Description
input-value-1	value	The block's top input value.
collection-time-1	float	The collection-time of the block's top input path.
quality-1	symbol	The quality of the block's top input path.
input-value-2	value	The block's bottom input value.
collection-time-2	float	The collection-time of the block's bottom input path.
quality-2	symbol	The quality of the block's bottom input path.

For example, this diagram uses the Pythagorean Theorem to figure the size of a right triangle's hypotenuse given the sizes of the other two sides.



Configuring

This is the configuration panel for Arithmetic Function of Two Arguments.



Field	Description
Arithmetic Function	The name of the function to apply to this block's two input values.

See Also

For more information on menu choices that are not described in this section, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Editing Attribute Displays		<i>User's Guide</i>

Vector Blocks

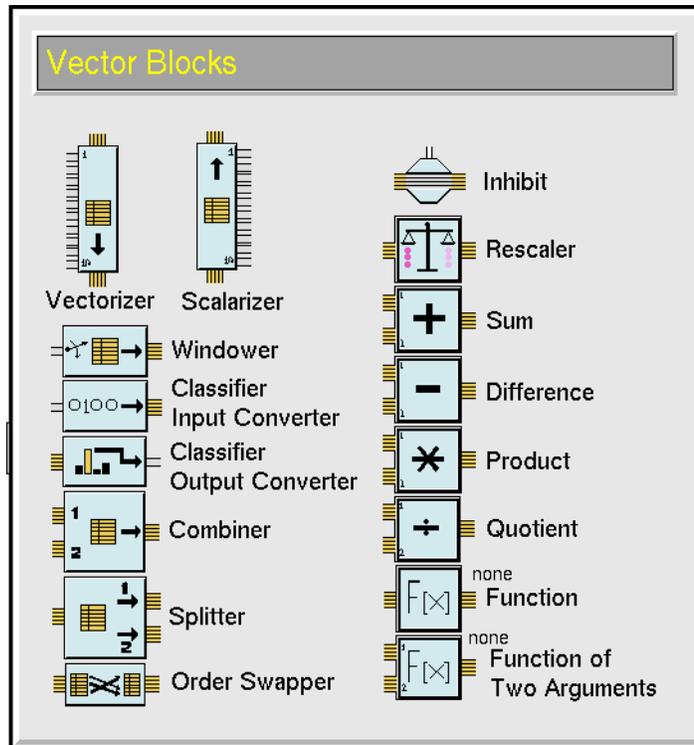
Describes the blocks that create, manipulate, and operate on vectors.

Introduction	116
Vectorizer	119
Scalarizer	122
Windower	124
Classifier Input Converter	126
Classifier Output Converter	128
Vector Combiner	129
Vector Splitter	131
Vector Order Swapper	133
Vector Inhibit	136
Vector Rescaler	138
Vector Sum	141
Vector Difference	143
Vector Product	145
Vector Quotient	147
Vector Function	149
Vector Function of Two Arguments	154

Introduction

NeurOn-Line provides you with a number of blocks that let you create, manipulate, and operate on vectors.

You can find the Vector Blocks palette under the Data Processing submenu of the Palettes menu:



Choosing When to Evaluate

The attribute Triggering Method lets you choose when a block executes. A block can execute when it receives one new input, all new inputs, or an input from a particular port.

If Triggering Method is...	The block evaluates when...
trigger on any	Any of its input ports receives a new value.
trigger on all	All of its input ports receive new values.
trigger on 1	The top input port receives a new value.
trigger on 2	The bottom input port receives a new value.

The following blocks contain this attribute:

- [Vectorizer](#)
- [Vector Combiner](#)
- [Vector Sum](#)
- [Vector Difference](#)
- [Vector Product](#)
- [Vector Quotient](#)
- [Vector Function of Two Arguments](#)

Note that in the Vectorizer block, you can only choose trigger on any or trigger on all.

When you reset a block that supports the attribute Trigger Method, NOL treats all the valid values at its input ports as newly received values. If its input values are valid, it will pass a vector the next time you evaluate it, regardless of the setting of Triggering Method.

Creating Vectors and Scalars

These blocks let you create a single vector path out of one or more scalar data paths, or create multiple scalar paths out of one vector data path:

- The [Vectorizer](#) block combines the scalars from several data paths into one vector.
- The [Windower](#) block creates a vector out of consecutive data from one scalar path.
- The [Scalarizer](#) block creates multiple scalar data paths from one vector data path.

Using Vectors with Classifiers

These blocks are useful when you are using a neural network to classify data:

- The [Classifier Input Converter](#) block converts a number that represents a class to a vector that represents a class.
- The [Classifier Output Converter](#) block returns the index of the maximum element in a vector. If each element in the vector is the probability that an input belongs to a class, the maximum element is the class to which the input most likely belongs.

Manipulating Vectors

These blocks let you combine, split, and reorder vectors:

- The [Vector Combiner](#) block combines two vector paths into one.
- The [Vector Splitter](#) block splits one vector path into two.
- The [Vector Order Swapper](#) block reorders the elements in a vector.

Inhibiting Vectors

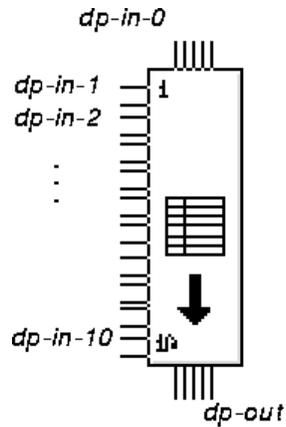
The [Vector Inhibit](#) block cuts off the flow of data in a vector path until an inference input port receives a particular value.

Operating on Vector Elements

These blocks let you perform arithmetic operations and other functions on the elements of a vector:

- The [Vector Rescaler](#) block lets you rescale each element of a vector using different factors for each element.
- The [Vector Sum](#) block adds the elements from one vector to the elements of another.
- The [Vector Difference](#) block subtracts the elements of one vector from the elements of another.
- The [Vector Product](#) block multiplies the elements of one vector by the elements of another.
- The [Vector Quotient](#) block divides the elements of one vector by the elements of another.
- The [Vector Function](#) block applies a function to each element of a vector, or collectively on its elements.
- The [Vector Function of Two Arguments](#) block applies a function to each element of two vectors, or collectively on its elements.

Vectorizer



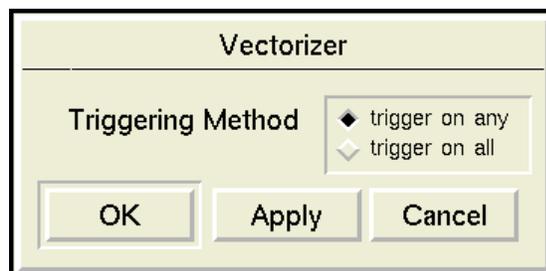
The Vectorizer block appends its input scalar values to its input vector. If there is no input vector, the block creates a new vector with its input scalar values as its elements.

To add more than ten values to a vector, append several Vectorizer blocks together. To add fewer than ten values to a vector, delete the remaining ports by dragging them onto the block. If you do not delete the unused ports, the block will not execute.

Note All the Vectorizer's input ports (including the vector input port `dp-in-0`) must be connected and have valid values before the block will pass a vector. The block does not pass a vector if it has any input ports that are not connected to a path or if any of the paths for the input ports has a quality of no-value.

Configuring

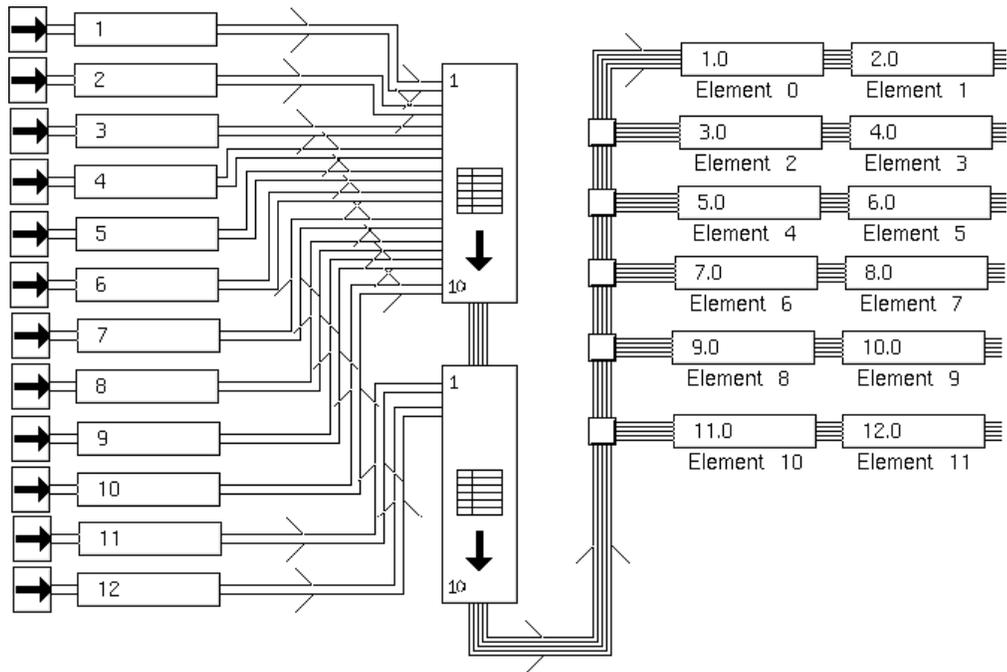
This is the configuration panel for Vectorizer.



Attribute	Description
Triggering Method	When to evaluate the block. For more information, see Choosing When to Evaluate .

Example

The diagram below shows how to use a Vectorizer block to create a vector out of twelve input Data Entry Points.

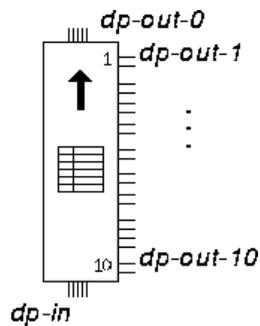


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Using Vector Paths		<i>User's Guide</i>
Scalarizer		<i>Reference Manual</i>
Windower		<i>Reference Manual</i>

Scalarizer



The Scalarizer block passes the elements of its input vector as scalar values. If the input vector contains more elements than the number of output ports, the block passes the remaining elements as an output vector.

To pass more than ten values from a vector, append several Scalarizer blocks together. To pass fewer than ten values from a vector, delete the remaining ports by dragging them into the block. If the block passes all the elements of the vector, you can also delete the output vector port by dragging it into the block.

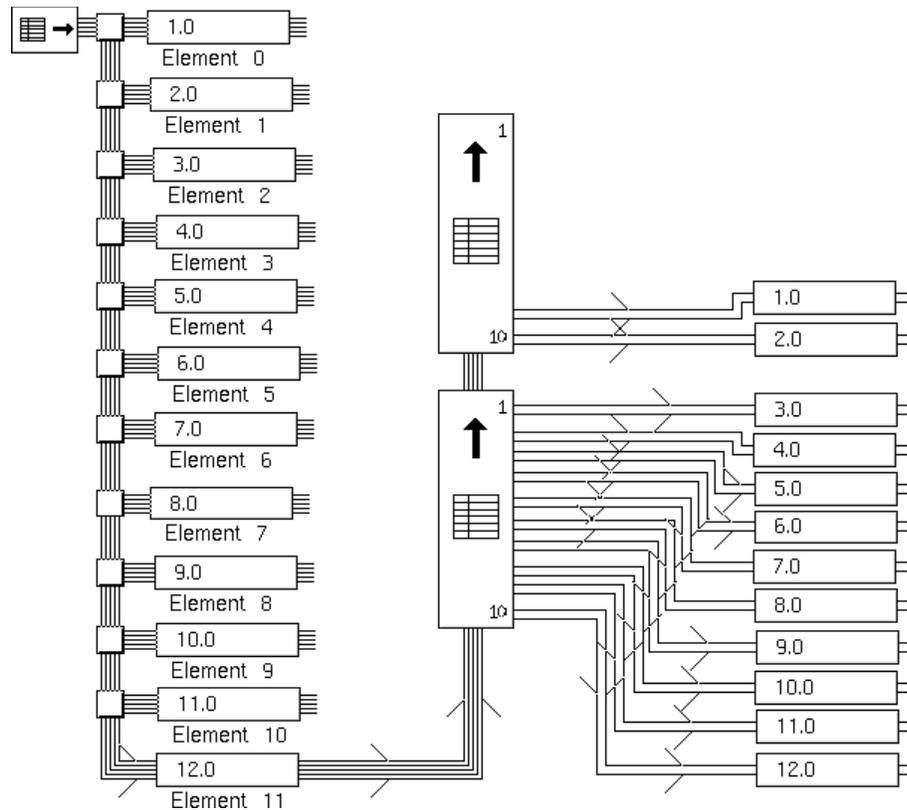
NeurOn-Line displays an error if the input vector is larger than the number of output scalar ports and the output vector port is deleted.

Configuring

This block has no configuration panel.

Example

The diagram below shows how to convert a 12-element vector into 12 scalar values.

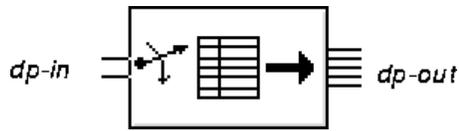


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Using Vector Paths		<i>User's Guide</i>
Vectorizer		<i>Reference Manual</i>
Vector Splitter		<i>Reference Manual</i>

Windower



The Windower block creates a vector from consecutive input values. You specify the size of the vector with the attribute Window Width. The block combines the specified number of previous input values into a vector, with the most recently received value being the first element of the vector. The block does not pass an output vector until it has received enough input to fill its window.

Configuring

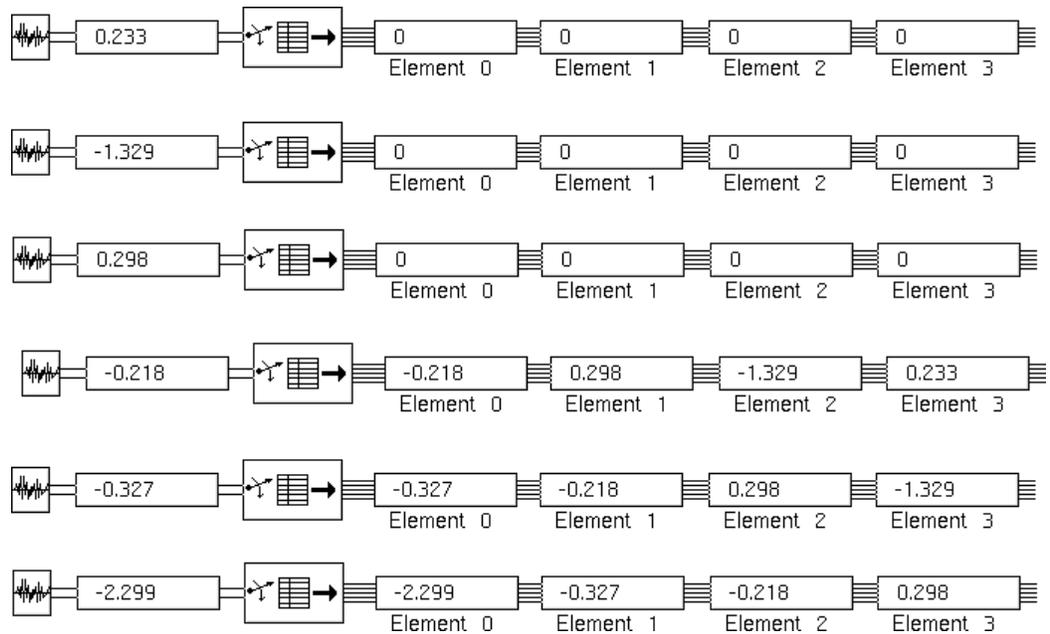
This is the configuration panel for Windower.



Attribute	Description
Window Width	Specifies how many consecutive input values to put in the output vector.

Example

The six diagrams below show a Windower with a Window Width of 4 over a period of time. Note that it does not pass a vector until it has received 4 values.

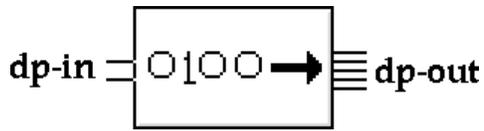


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Using Vector Paths		<i>User's Guide</i>
Vectorizer		<i>Reference Manual</i>

Classifier Input Converter



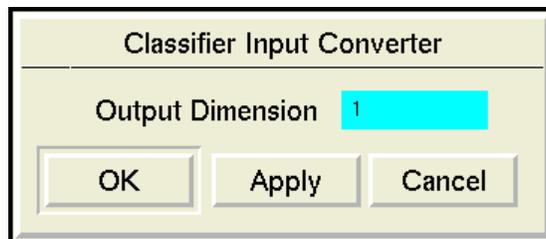
The Classifier Input Converter block converts an integer to a vector that you can use as a training target for a classification problem.

Specify the total number of possible classes in the attribute Output Dimension. The input value specifies one of those classes. The block outputs a vector which contains all zeros, except that the element specified by the input contains a 1. For example, if the Output Dimension is 5 and the input value is 2, the output vector contains (0,0,1,0,0). Note that the first element in the vector is 0.

If the input value is a floating-point number, the block rounds it to the nearest integer value. If the input value is greater than the Output Dimension or less than zero, the block generates an error.

Configuring

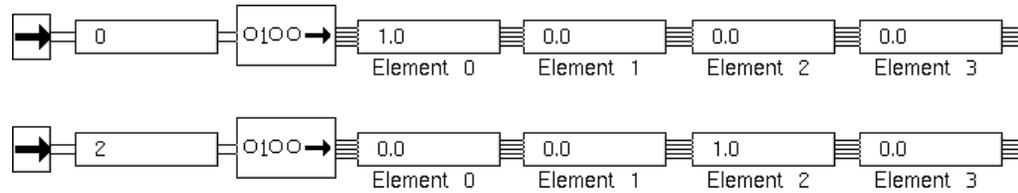
This is the configuration panel for Classifier Input Converter.



Attribute	Description
Output Dimension	Specifies the total number of possible classes.

Example

The two diagrams below show a Classifier Input Converter with an Output Dimension of 4.



See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Using Vector Paths		<i>User's Guide</i>
Classifier Output Converter		<i>Reference Manual</i>

Classifier Output Converter



The Classifier Output Converter block outputs the index of the largest element in its input vector. This block is especially useful when you need to understand the output of a neural network classifier. The largest value in the network's output vector usually corresponds to the most likely class as predicted by the network.

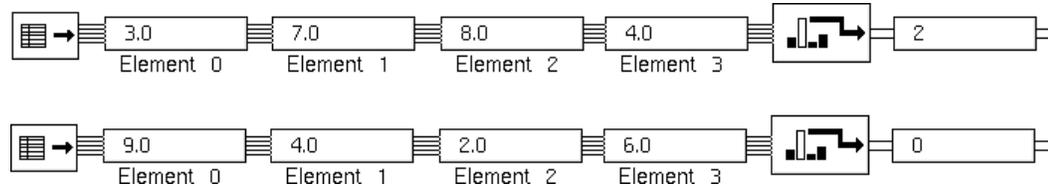
For example, if the input vector is (0.1, 0.0, 0.5, 0.2, 0.3), the output would be 2. Note that the first index for the vector is 0.

Configuring

This block has no configuration panel.

Example

The two diagrams below show a Classifier Output Converter.

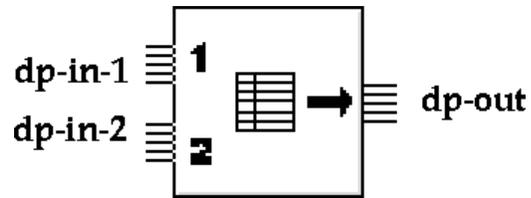


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Using Vector Paths		<i>User's Guide</i>
Classifier Input Converter		<i>Reference Manual</i>

Vector Combiner

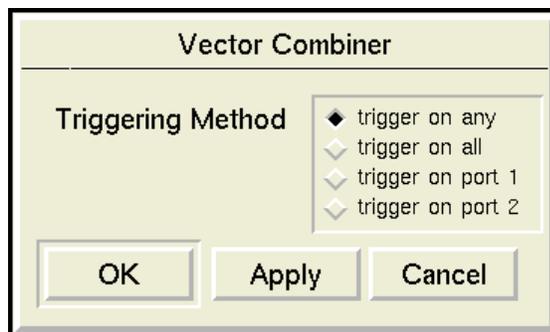


The Vector Combiner combines two vectors together by appending the second vector to the end of the first.

This block will not pass a value if the quality of either input path is no-value.

Configuring

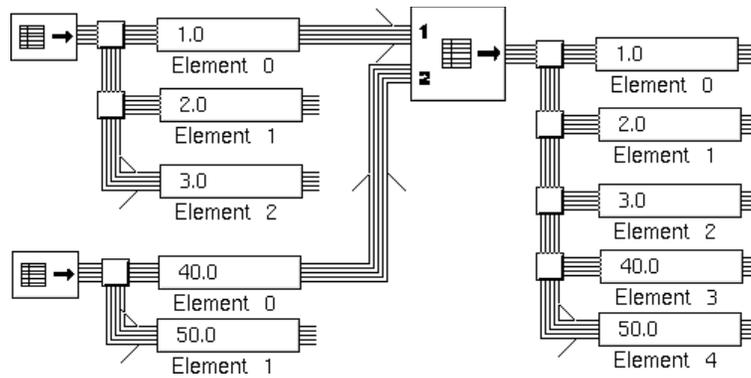
This is the configuration panel for Vector Combiner.



Attribute	Description
Triggering Method	When to evaluate the block. For more information, see Choosing When to Evaluate .

Example

The diagram below shows how a Vector Combiner appends a two-element vector to a three-element vector.



See Also

For general information on how to use this block, see the sections below.

For more information on...

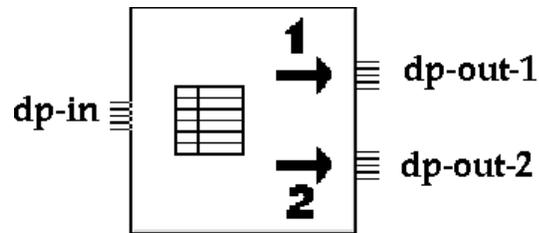
See...

In this book...

Basic Block Behavior

User's Guide

Vector Splitter

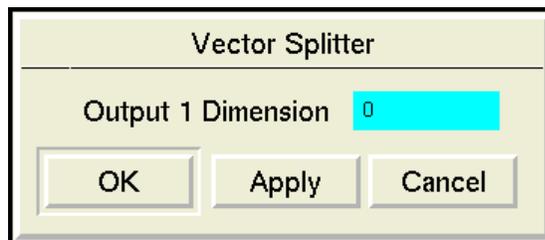


The Vector Splitter splits the input vector into two smaller vectors. Specify the size of the top input vector with the attribute Output 1 Dimension. The bottom vector contains what remains of the input vector after removing the specified number of elements.

If the size of the input vector is less than the Output 1 Dimension, the block generates an error.

Configuring

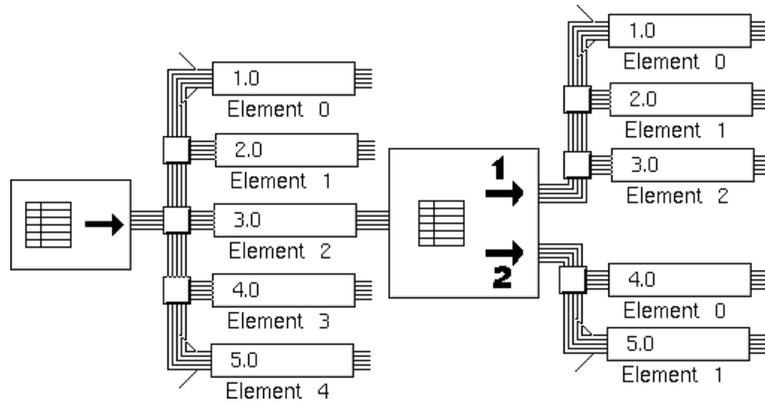
This is the configuration panel for Vector Splitter.



Attribute	Description
Output 1 Dimension	Specifies the size of the first output vector.

Example

The diagram below shows a Vector Splitter with Output 1 Dimension set to 3.



See Also

For general information on how to use this block, see the sections below.

For more information on...

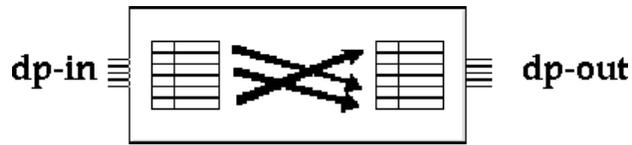
See...

In this book...

Basic Block Behavior

User's Guide

Vector Order Swapper



The Vector Order Swapper re-orders the elements of a vector.

To specify how to reorder the elements, choose configure from the block's menu. NeurOn-Line displays a GXL spreadsheet for editing the vectors.

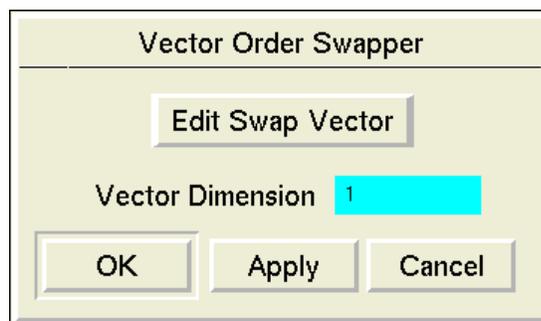
If the dimension on the input vector does not equal the dimension of the block's vector, the block generates an error.

Making Values Permanent

When you choose make permanent from the block's menu, it saves the current swapping order.

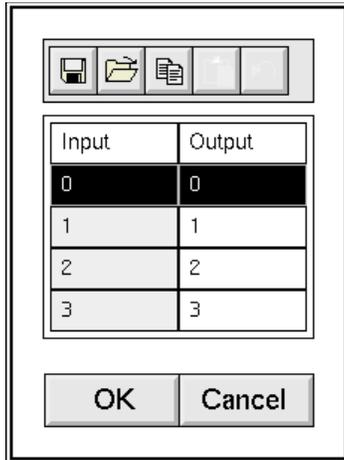
Configuring

When you choose configure from the block's menu, NOL displays this dialog, which lets you enter the dimension of the vector to swap:



Enter a value for Vector Dimension, and click the Edit Swap Vector button to display a spreadsheet for swapping the vector.

Here is the spreadsheet for swapping the order of a vector of length 4:



Input	Output
0	0
1	1
2	2
3	3

OK Cancel

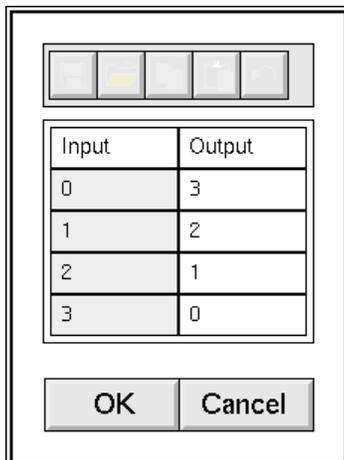
In the Output column, enter the new positions for the indexes. You cannot repeat indexes, and you must use each index once.

When you are done editing the vector, click the OK button in the spreadsheet, and click the OK or Apply button in the configuration panel.

For more information on how to use the spreadsheet, see [Using the GXL Spreadsheet to Edit Data](#).

Example

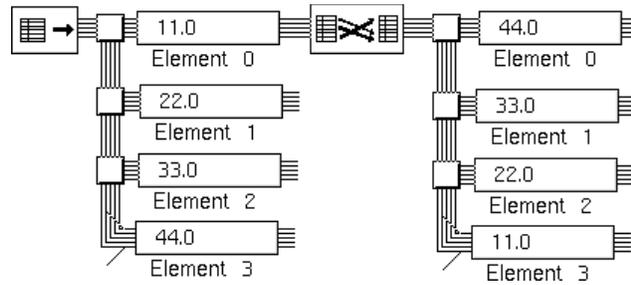
This spreadsheet for a Vector Order Swapper reverses the order of the elements in the input vector:



Input	Output
0	3
1	2
2	1
3	0

OK Cancel

This diagram shows that Vector Order Swapper in action.

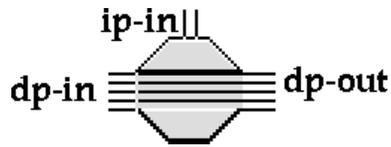


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>

Vector Inhibit



The Vector Inhibit block prevents data propagation on a vector path.

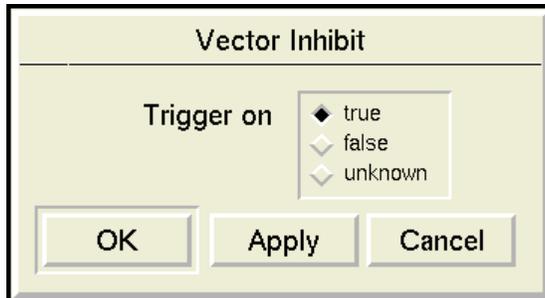
When the status value of the inference path matches the value of the attribute Trigger On, the block inhibits the input vector.

When the status value of the inference path no longer matches Trigger On, the block passes the current value of the input vector path and continues to pass the input vector normally.

If the block's inference path has not received a value yet (that is, it has a quality of no-value), the block passes nothing even when it receives a value from its vector path.

Configuring

This is the configuration panel for Vector Inhibit.



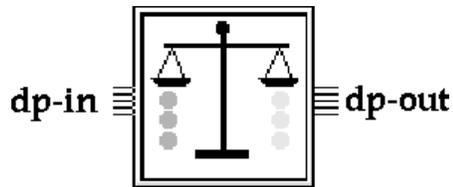
Attribute	Description
Trigger On	Specifies when to inhibit the vector value.

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Inhibit		<i>Reference Manual</i>
Inference Inhibit		<i>Reference Manual</i>

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 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:

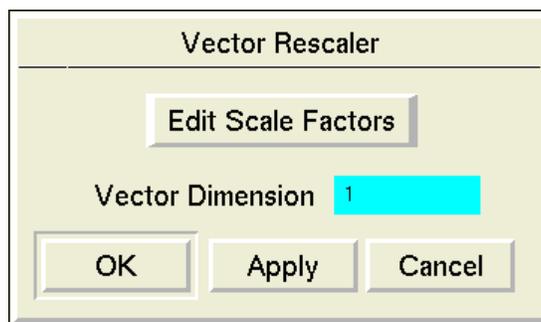
$$\text{Output}_i = M_i (\text{Input}_i + A_i)$$

Making Values Permanent

When you choose make permanent from the block's menu, it saves the current scaling factors.

Configuring

When you choose configure from the block's menu, NOL displays this dialog, which lets you enter the dimension of the vector to rescale:



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:

	Additive	Multiplicative
1	0.0	1.0
2	0.0	1.0
3	0.0	1.0
4	0.0	1.0

OK Cancel

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

When you are done editing the vector, click the OK button in the spreadsheet, and click the OK or Apply button in the configuration panel.

For more information on how to use the spreadsheet, see [Using the GXL Spreadsheet to Edit Data](#).

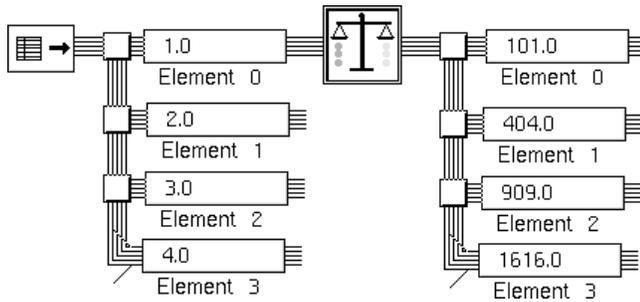
Example

These are the scale factors for a four-element vector.

	Additive	Multiplicative
1	100.0	1.0
2	200.0	2.0
3	300.0	3.0
4	400.0	4.0

OK Cancel

Here is an example of how it scales a vector.

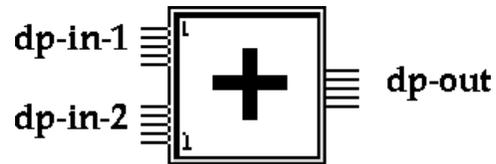


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>
Vector Function		<i>Reference Manual</i>

Vector Sum

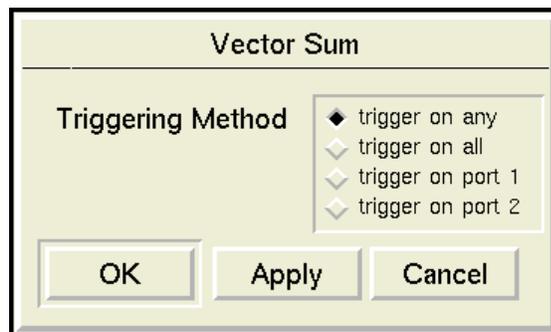


The Vector Sum block adds the elements of its two input vectors. It passes a vector in which each element is the sum of the input vectors' corresponding elements. For example, the first element of the output vector is the sum of the first elements of the first input vector and the first element of the second input vector.

This block will not pass a value if the quality of either input path is no-value. If the two input vectors have different lengths, NeurOn-Line generates an error.

Configuring

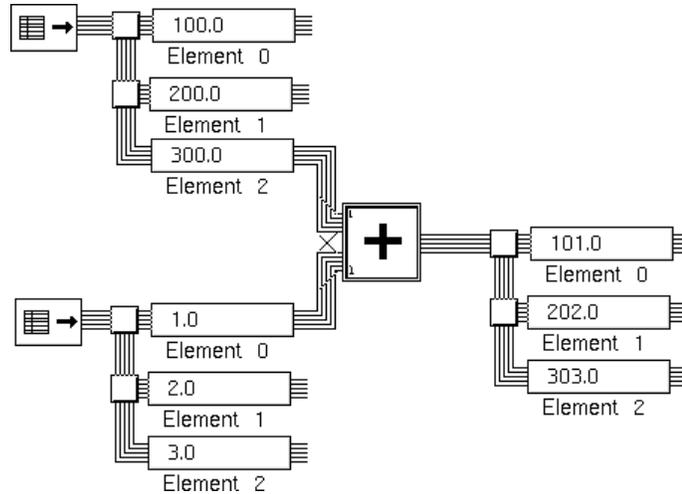
This is the configuration panel for Vector Sum.



Attribute	Description
Triggering Method	When to evaluate the block. For more information, see Choosing When to Evaluate .

Example

The example below adds the elements of two vectors.

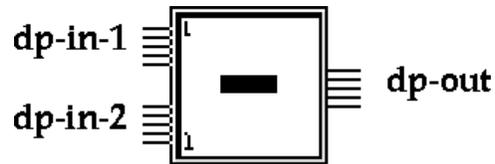


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Summation		<i>Reference Manual</i>

Vector Difference

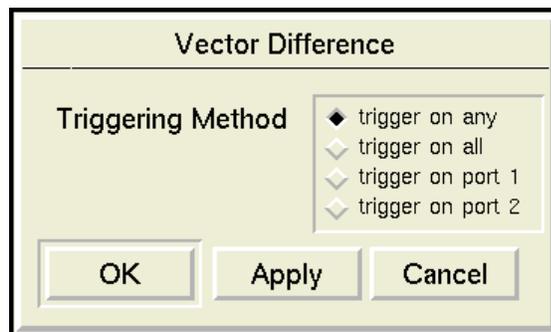


The Vector Difference block subtracts the elements of its second input vector from the elements of its first input vector. For example, the first element of the output vector is the first element of the first vector minus the first element of the second vector.

This block will not pass a value if the quality of either input path is no-value. If the two input vectors have different lengths, NeurOn-Line generates an error.

Configuring

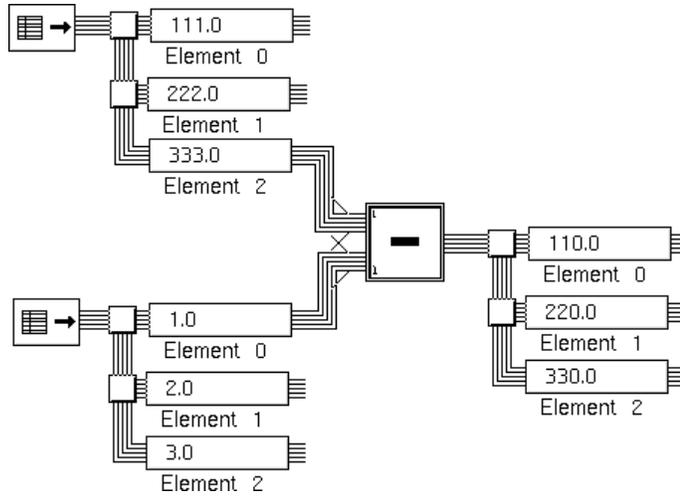
This is the configuration panel for Vector Difference.



Attribute	Description
Triggering Method	When to evaluate the block. For more information, see Choosing When to Evaluate .

Example

The example below subtracts the elements of two vectors.



See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Difference		<i>Reference Manual</i>

Vector Product

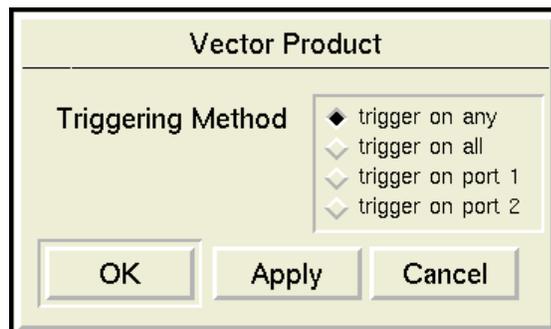


The Vector Product block multiplies the elements of its two input vectors. It passes a vector in which each element is the product of the input vectors' corresponding elements. For example, the first element of the output vector is the product of the first elements of the first input vector and the first element of the second input vector.

This block will not pass a value if the quality of either input path is no-value. If the two input vectors have different lengths, NeurOn-Line generates an error.

Configuring

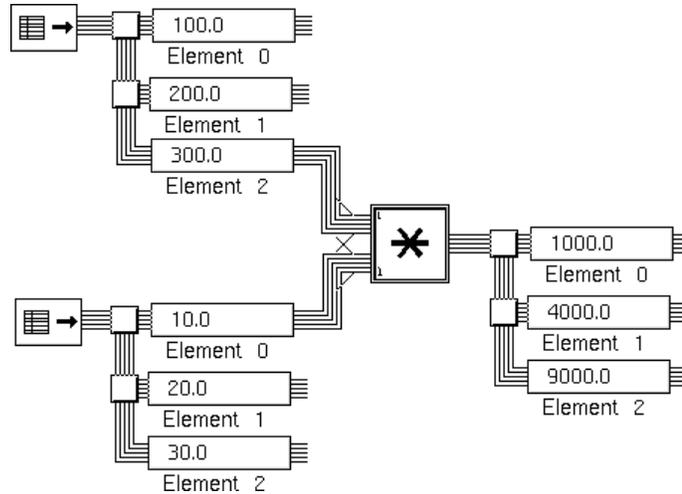
This is the configuration panel for Vector Product.



Attribute	Description
Triggering Method	When to evaluate the block. For more information, see Choosing When to Evaluate .

Example

The example below multiplies the elements of two vectors.



See Also

For general information on how to use this block, see the sections below.

For more information on...

See...

In this book...

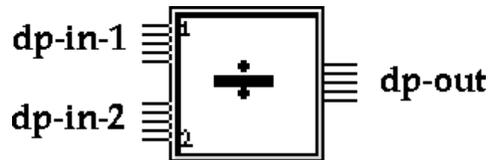
Basic Block Behavior

User's Guide

Multiplication

*Reference
Manual*

Vector Quotient

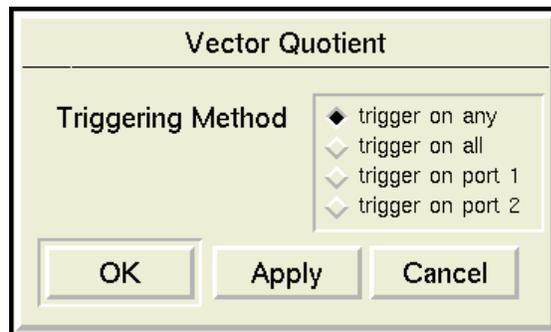


The Vector Quotient block divides the elements of its first input vector by the elements of its second input vector. For example, the first element of the output vector is the first element of the first vector divided by the first element of the second vector.

This block will not pass a value if the quality of either input path is no-value. If the two input vectors have different lengths, NeurOn-Line generates an error.

Configuring

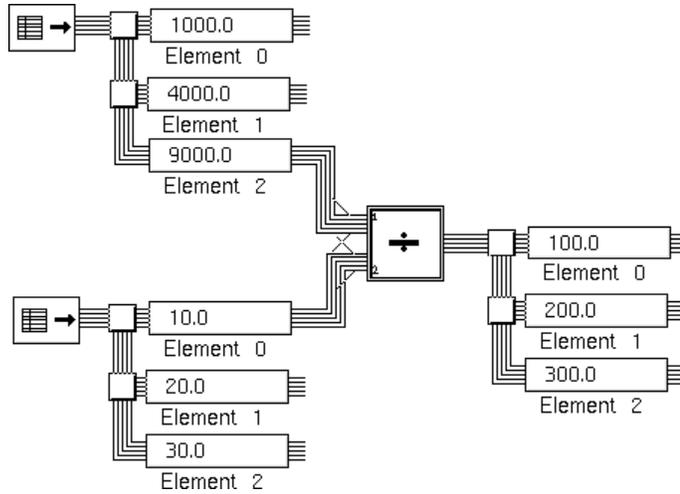
This is the configuration panel for Vector Quotient.



Attribute	Description
Triggering Method	When to evaluate the block. For more information, see Choosing When to Evaluate .

Example

The example below divides the elements of two vectors.



See Also

For general information on how to use this block, see the sections below.

For more information on...

See...

In this book...

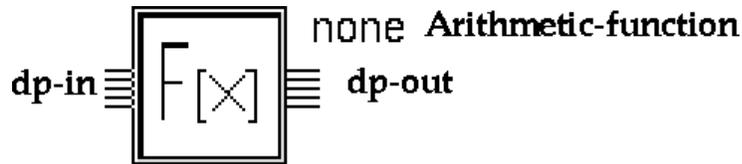
Basic Block Behavior

User's Guide

Quotient

*Reference
Manual*

Vector Function



The Vector Function block lets you use a function or procedure as a block in a NeurOn-Line diagram. The block applies the function or procedure to its input vector and passes the result. Specify the name of the routine in the attribute Arithmetic Function.

If Arithmetic Function is a function that takes a numeric argument, the block applies the function to each element in the input vector and passes a vector of the results. If Arithmetic Function is a procedure, the block applies the procedure to the input vector as a whole and passes the resulting vector.

You can use any:

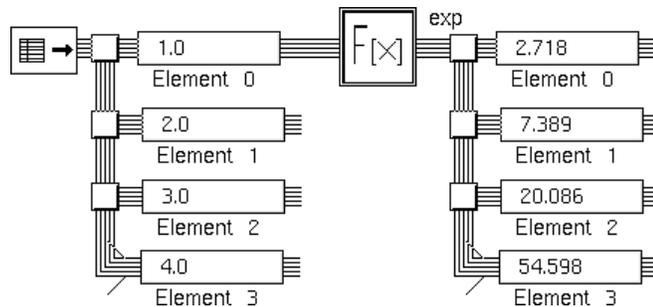
- Built-in G2 function
- User-defined function
- Procedure
- Tabular-function

Using a Built-in G2 Function

You can set the attribute Arithmetic Function to any of these built-in G2 functions:

abs	arctan	ceiling	cos
exp	floor	int	ln
log	random	sin	sqrt
tan			

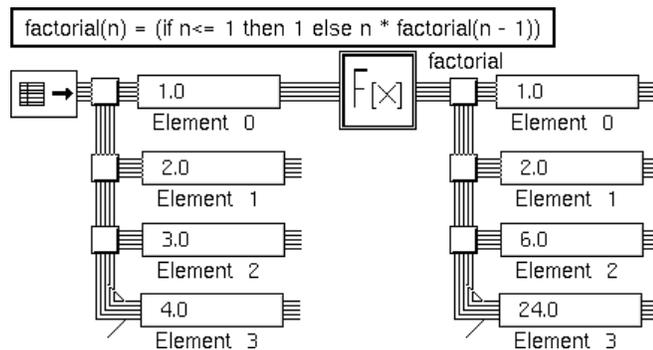
The picture below shows a diagram that applies the function `exp` to each element of a vector.



Using a User-Defined Function

You can use any user-defined function that accepts one quantitative argument and returns one quantitative value. Set the attribute Arithmetic Function to the name of the function.

The picture below shows a diagram that figures the factorial of every element in a vector.



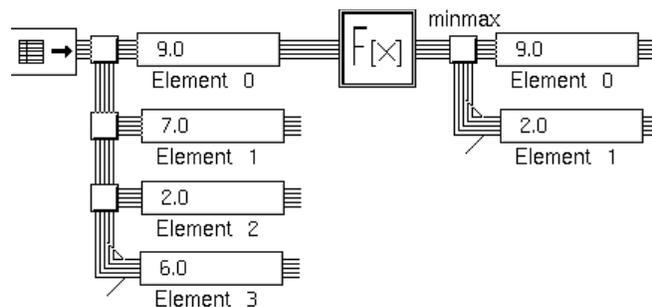
Using a Procedure

You can use a procedure that accepts a vector as an argument and returns a vector. Use a procedure if you want to operate on a vector as a whole, instead of operating on each element individually.

Set the attribute Arithmetic Function to the name of the procedure. The block passes the procedure's return value as its output value. The input vector and output vector can have different lengths. The vectors that the procedure accepts and returns should be of type vector-path-value, which has the superior class float-array and contains the following two additional attributes.

Attribute	Type	Description
collection-time	float	The collection-time of the block's input vector.
quality	symbol	The quality of the block's input vector.

This diagram contains a Vector Function block with a procedure that creates a two-element vector containing the minimum and maximum values of the input vector.



This is the definition of the procedure.

```

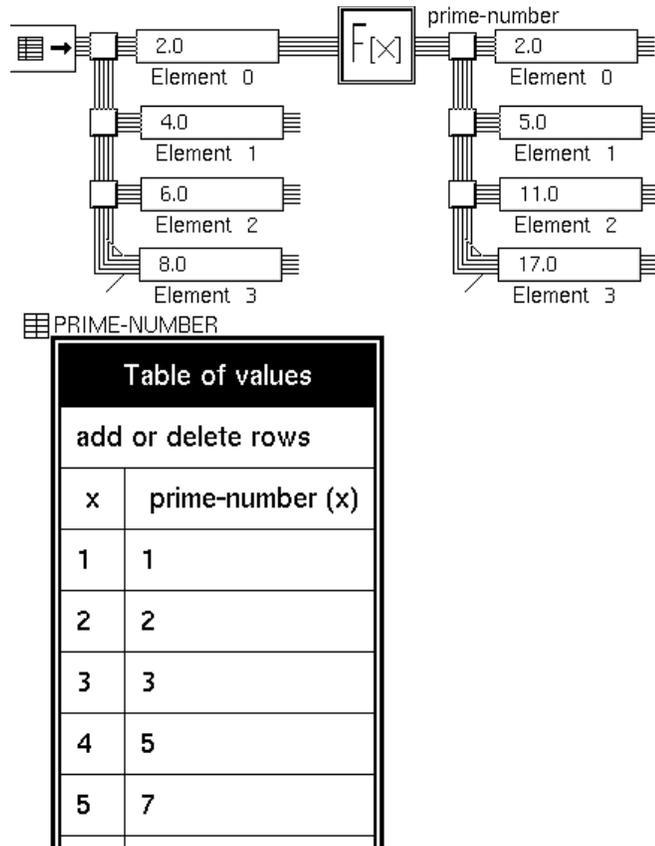
minmax(in-vec: class vector-path-value) = (class vector-path-value)
  out-vec: class vector-path-value;
  begin
    create a vector-path-value out-vec;
    change the array-length of out-vec to 2;
    change out-vec[0] = the maximum over each float F in in-vec of (F);
    change out-vec[1] = the minimum over each float F in in-vec of (F);
    return out-vec;
  end

```

Using a Tabular Function

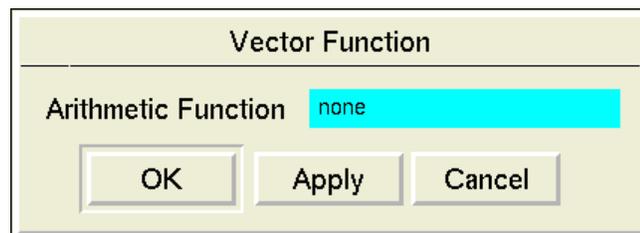
You can use the G2 tabular-function-of-1-arg function that accepts one argument and returns a value. Set the attribute Arithmetic Function to the name of your function. The block passes its input value to the function as an argument and passes the function's return value as its output value. If the function cannot evaluate the input value, NeurOn-Line signals an error. For information on a G2 tabular-function-of-1-arg, see the G2 Reference Manual.

This diagram shows a Vector Function block that uses a tabular-function-of-1-arg.



Configuring

This is the configuration panel for Vector Function.



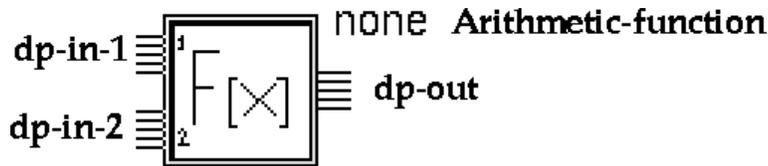
Attribute	Description
Arithmetic Function	The function to apply to each element of the input vector.

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Vector Function of Two Arguments		<i>Reference Manual</i>

Vector Function of Two Arguments



The Vector Function of Two Arguments block lets you use a function or procedure as a block in a NeurOn-Line diagram. The block applies the function or procedure to its two input vectors and passes one vector as the result. Specify the name of the routine in the attribute Arithmetic Function.

You can use any:

- Built-in G2 function
- User-defined function
- Procedure

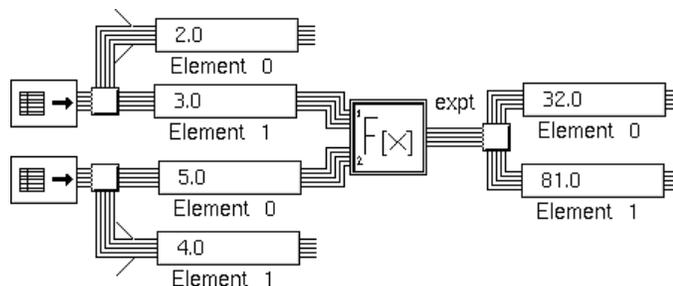
This block will not pass a value if the quality of either input path is no-value.

Using a Built-in G2 Function

You can set the attribute Arithmetic Function to any of these built-in G2 functions:

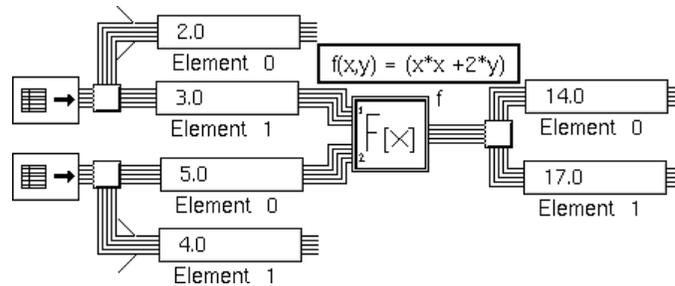
arctan average expt max
min quotient random remainder

The Vector Function of Two Arguments block applies the function to each of the elements in the two input vectors and creates an output vector of the results. For example, this diagram computes 25 and 34.



Using a User-Defined Function

You may use any user-defined function that accepts two quantitative arguments and returns one quantitative value. Set the attribute Arithmetic Function to the name of the function. The Two Argument Vector Function block applies the function to each of the elements in the two input vectors and creates an output vector of the results. For example, the diagram below applies the function to each member of two vectors. In this case, it computes and



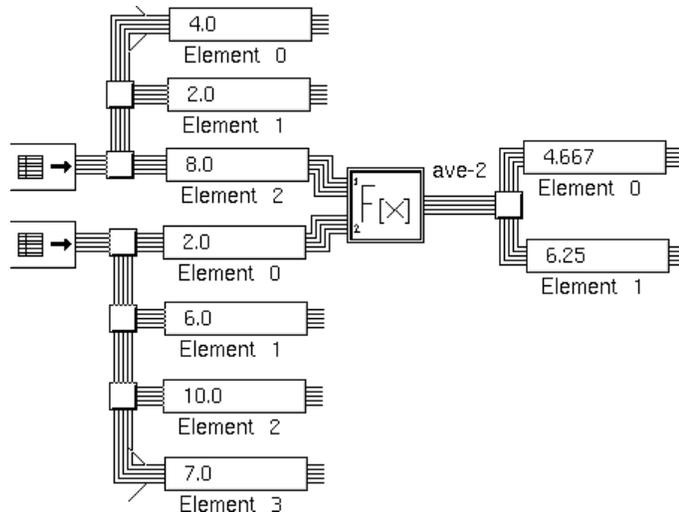
Using a Procedure

You can use a procedure that accepts two vectors as arguments and returns a vector. Use a procedure if you want to operate on the vectors as a whole, instead of operating on each element individually.

Set the attribute Arithmetic Function to the name of the procedure. The block passes the procedure's return value as its output value. The input vectors and the output vectors can all have different lengths. The vectors that the procedure accepts and returns should be of type vector-path-value, which has the superior class float-array and contains the following two additional attributes.

Attribute	Type	Description
collection-time	float	The collection-time of the block's input vector.
quality	symbol	The quality of the block's input vector.

This diagram contains a Vector Function of Two Arguments block with a procedure that creates a two-element vector containing the average values of the two input vectors.



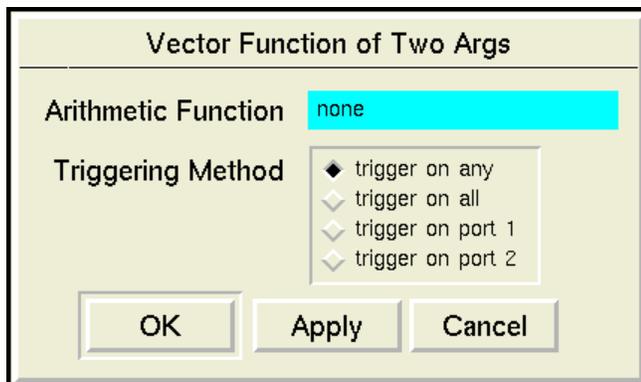
This is the definition of the procedure.

```

ave-2(v1: class vector-path-value, v2: class vector-path-value) =
  (class vector-path-value)
  out-vec: class vector-path-value;
  begin
    create a vector-path-value out-vec;
    change the array-length of out-vec to 2;
    change out-vec[0] = the average over each float F in v1 of (F);
    change out-vec[1] = the average over each float F in v2 of (F);
    return out-vec;
  end

```

This is the configuration panel for Vector Function of Two Arguments.



Attribute	Description
Arithmetic Function	The function to apply to each element of the input vectors.
Triggering Method	Specify when to evaluate the block. For more information, see Choosing When to Evaluate .

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Vector Function		<i>Reference Manual</i>

Data Set Blocks

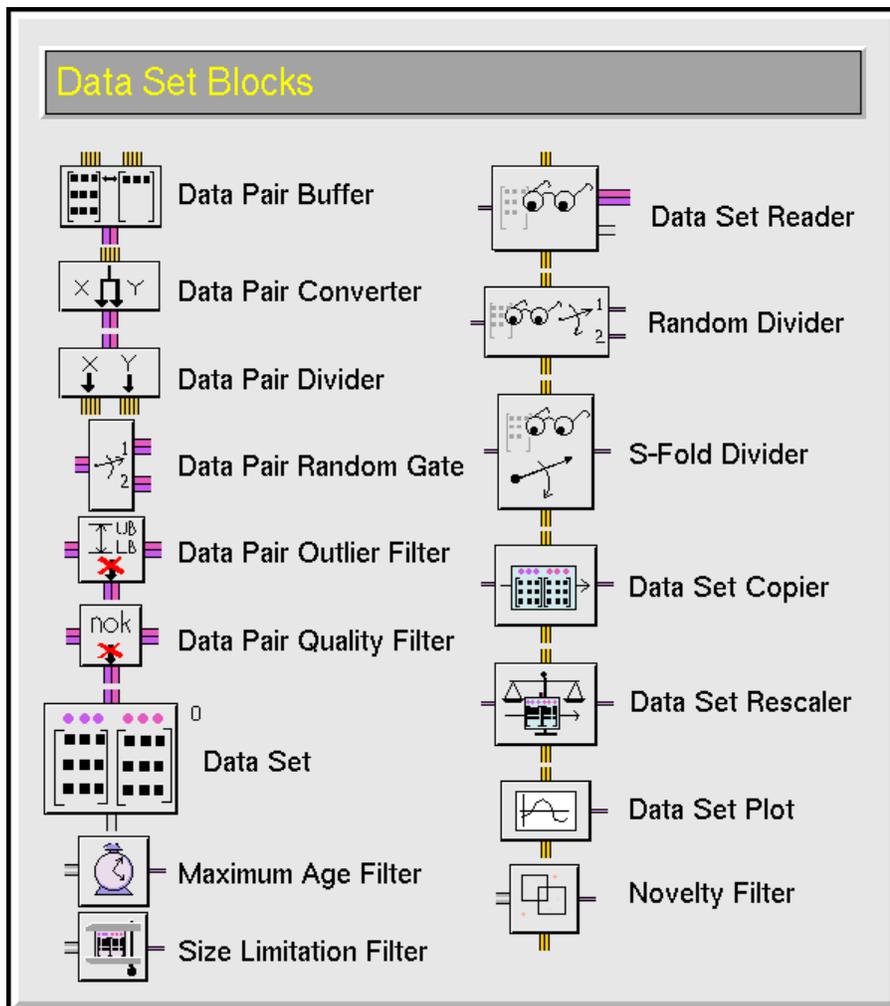
Describes the blocks that store and manipulate the data with which you train a neural network.

Introduction	160
Data Pair Buffer	163
Data Pair Converter	166
Data Pair Divider	168
Data Pair Random Gate	169
Data Pair Outlier Filter	171
Data Pair Quality Filter	175
Data Set	177
Maximum Age Filter	185
Size Limitation Filter	187
Data Set Reader	189
Random Divider	191
S-Fold Divider	193
Data Set Copier	195
Data Set Rescaler	197
Data Set Plot	201
Novelty Filter	209

Introduction

NeurOn-Line provides you with a number of blocks that store and manipulate the data with which you train a neural network.

You can find the Data Set Blocks palette under the Data Processing submenu of the Palettes menu:



The blocks on this palette let you store, filter, and manipulate the data you need to train and test a neural network. First, you create a data pair, which contains two vectors: the input data (X) and the target data (Y). Next, you add the data pairs to a data set. You can filter the data to make sure the data set has data pairs that are of a certain age, within certain bounds, and so on. You can copy the data in a data set to other data sets. Finally, you can view the data in a plot.

Creating Data Pairs

These blocks let you create data pairs and add them to a data set:

- The [Data Pair Buffer](#) block creates a data pair by combining two vectors: one vector becomes the data pair's input vector, the other becomes the data pair's output vector.
- The [Data Pair Converter](#) block creates a Data Pair by splitting a vector: one part of the vector becomes the data pair's input vector, the other becomes the data pair's output vector.
- The [Data Pair Random Gate](#) block randomly places a data pair into one of its two output data sets.

Filtering Data

These filter blocks let you exclude certain types of data from a data set. Every filter block can put the excluded data pairs into another data set.

These filters intercept data pairs before they are added to a data set:

- The [Data Pair Outlier Filter](#) block excludes data pairs that fall outside specified bounds.
- The [Data Pair Quality Filter](#) block excludes data pairs whose Quality attribute is not OK.

These filters attach directly to the data set. Each time the data set receives a data pair, the filter checks whether other data pairs in the data set need to be removed.

- The [Maximum Age Filter](#) block removes data pairs that are older than a specified age.
- The [Size Limitation Filter](#) block removes data pairs when the size of the data set exceeds a specified maximum.
- The [Novelty Filter](#) block removes data pairs that have values close to the newly received data pair.

Choosing When a Data Set Filter Executes

NeurOn-Line contains three filters that attach directly to a data set: the Novelty filter, the Size Limitation Filter, and the Maximum Age Filter. Whenever the data set receives a data pair, these filters remove other data pairs if they fit certain criteria. You can choose the order in which they evaluate by setting the attribute

Execution Priority, a number from 1 to 10, where 1 is the highest priority and 10 is the lowest. The table below shows the default values for each filter:

For this filter...	The default Execution Priority is...
Novelty Filter	1
Size Limitation Filter	2
Maximum Age Filter	3

Reading Data

These blocks read and plot the data that is stored in a data set:

- The [Data Set Reader](#) block copies consecutive data pairs from a data set onto a data pair path. It does not remove data pairs from the data set.
- The [Data Pair Converter](#) block converts a data pair path into two vector paths: one carrying the data pair's input vector, the other carrying the data pair's output vector.
- The [Data Set Plot](#) block plots the contents of a data set onto a G2 chart.

Copying Data

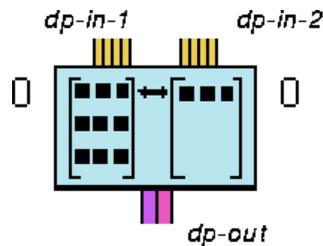
These blocks let you copy data from one or more input data sets into one or more output data sets:

- The [Random Divider](#) block randomly copies all the data pairs from one or more data sets to two output data sets.
- The [S-Fold Divider](#) block randomly copies all the data pairs from one or more input data sets to one or more output data sets. Each output data set receives a subset of approximately equal size.
- The [Data Set Copier](#) block copies or combines data sets. It pools the data pairs from all its input data sets and copies the complete pool to all the output data sets.

Scaling Data

The [Data Set Rescaler](#) block scales the input and target data in a data set. 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.

Data Pair Buffer



The Data Pair Buffer creates a data pair from two vectors. The left input becomes the X vector and the right input becomes the Y vector.

This block can operate two ways:

- If the Concurrency Window is none, it pairs X and Y vectors in the order they arrive, buffering any vectors from an input port if there are no corresponding inputs from the other vector port. For example, if two vectors arrive from the left input port and one from the right input port, the block pairs the first vector from the left port with the one from the right port and saves the second vector from the left port until another vector arrives on the right port.
- If the Concurrency Window is any other value, it pairs X and Y vectors only if they arrive within the specified concurrency window, discarding vectors that arrive outside the window. The following heading describes this situation in more detail.

Specifying Whether Values are Concurrent

Usually, you want to combine two vectors only if they were created at approximately the same time. However, vectors that were created at the same time may arrive at slightly different times due to computer speed or network speed. This block considers two vectors to be concurrent if they arrive within the period of time you specify in the attribute Concurrency Window.

For example, suppose that Concurrency Window is 1 second. If you receive a vector in the left port and then a half second later receive a value in the right port, the block considers the two values to be concurrent and passes them as a data pair. However, if you receive a value in the left port, and one second passes with no value received on the right port, the block discards the vector it received on the left port and does not pass a new data pair.

Note If you are testing a diagram by entering data manually (for example, by using the override menu choice), make sure the Concurrency Window is large enough so you can enter values at a reasonable pace.

Resetting

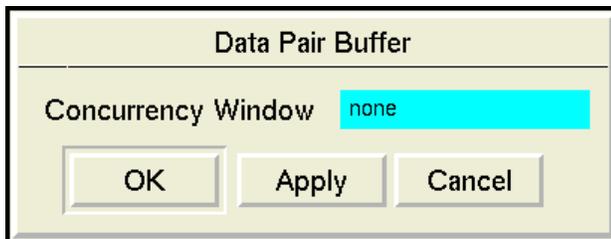
After you reset the Data Pair Buffer, the block treats all the valid values at its input ports as newly received values. If its input values are valid, it will pass a data pair the next time you evaluate it.

Clearing the Data Pair Buffer

To clear the x and y vector values, select the clear data pair buffer menu choice. The attribute displays indicate the stored x and y vectors are reset to zero. The block only clears the data pair buffer when Concurrency Window is none. If Concurrency Window is a value, clearing the data buffer does nothing; NOL does not clear the current waiting value.

Configuring

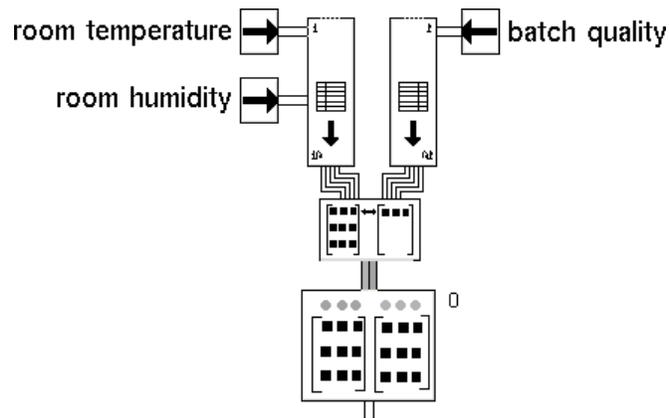
This is the configuration panel for the Data Pair Buffer.



Attribute	Description
Concurrency Window	The period of time in which two values must be received for the block to consider them to be concurrent.

Example

In the example below, the Data Pair Buffer combines two vectors into a data pair and then adds the data pair to a Data Set. The input vector contains the room temperature and room humidity. The output vector contains the batch quality, a single number that represents the quality of the batch created in that room.

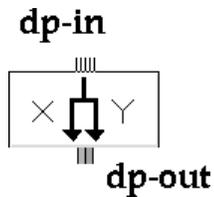


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Pair Converter		<i>Reference Manual</i>
Data Pair Divider		<i>Reference Manual</i>

Data Pair Converter

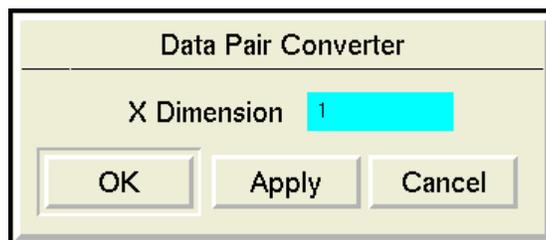


The Data Pair Converter forms a data pair by dividing its input vector into X and Y vectors. To specify how to split the vector, use the attribute X Dimension. The Data Pair Converter puts that many elements into the data pair's X vector and the rest of the elements into the data pair's Y vector.

NeurOn-Line generates an error if the dimension of the input vector is equal to or smaller than X Dimension.

Configuring

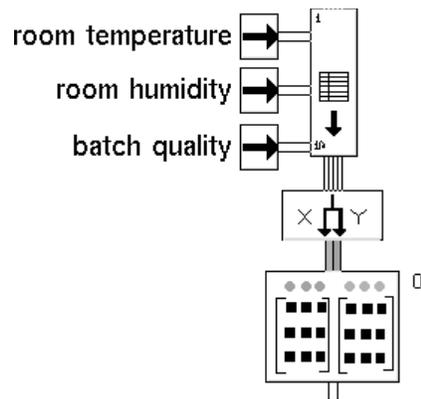
This is the configuration panel for the Data Pair Converter.



Attribute	Description
X Dimension	How many of the vector elements to put into the input part of the data pair.

Example

In the example below, the Data Pair Converter creates a data pair from a vector and adds the data pair to a Data Set. The X Dimension is 2, so the input vector contains the room temperature and room humidity, and the output vector contains the batch quality.

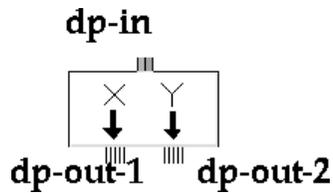


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Pair Buffer		<i>Reference Manual</i>
Data Pair Divider		<i>Reference Manual</i>

Data Pair Divider



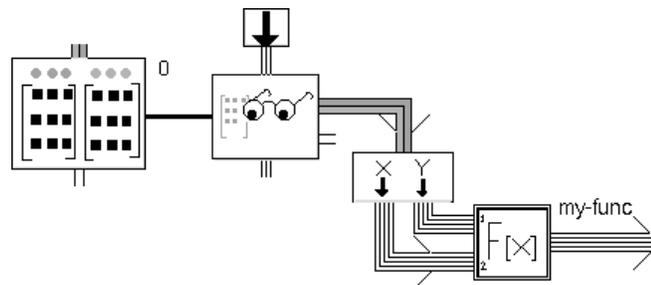
The Data Pair Divider splits its input data pair into two separate vectors: the left output port is the data pair's X vector and the right vector is the data pair's Y vector.

Configuring

This block has no configuration panel.

Example

In the example below, the Data Pair Divider takes a data pair from a Data Set and passes the input and target vectors to a Vector Function of Two Arguments block.

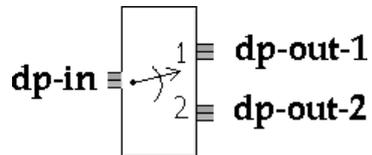


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Pair Buffer		<i>Reference Manual</i>
Data Pair Converter		<i>Reference Manual</i>

Data Pair Random Gate

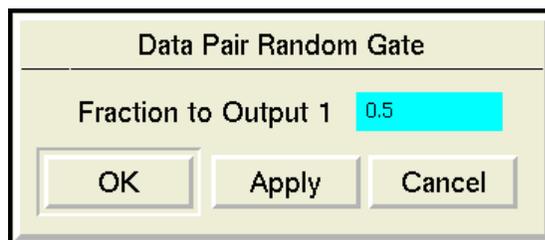


The Data Pair Random Gate randomly propagates its input data pair to one of two output paths. You must specify which proportion of the data pairs go to the top output port with the attribute Fraction to Output 1. For example, if Fraction to Output 1 is 0.6, then approximately 60% of the input data pairs go to the top output port, and approximately 40% go to the bottom output port.

This block is especially useful for splitting data pairs between a training and a test Data Set. To split data already contained in a data set, use a Random Divider.

Configuring

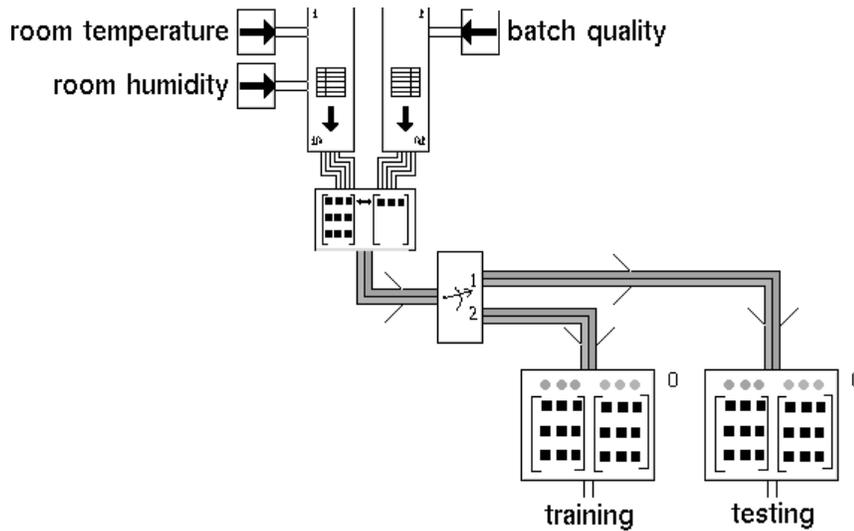
This is the configuration panel for the Data Pair Random Gate.



Attribute	Description
Fraction to Output 1	The proportion of the data pairs that go to the top output port.

Example

In the example below, the Data Pair Random Gate takes data pairs from a Data Pair Buffer and randomly splits them between two Data Sets: one for training and one for testing.

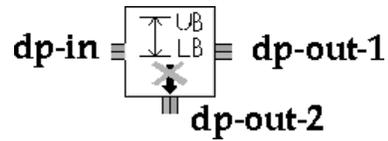


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Random Divider		<i>Reference Manual</i>
S-Fold Divider		<i>Reference Manual</i>

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, use the configure command. NOL displays this configuration panel:

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. NOL displays a spreadsheet for editing the input and output bounds.

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

Inputs	Lower	Upper
1	0.0	1.0
2	0.0	1.0

Targets	Lower	Upper
1	0.0	1.0

The editor has two columns: Lower 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.

When you are done editing the bounds, click the OK button in the spreadsheet and edit the other dimension. Click the OK or Apply button in the configuration panel when you are finished.

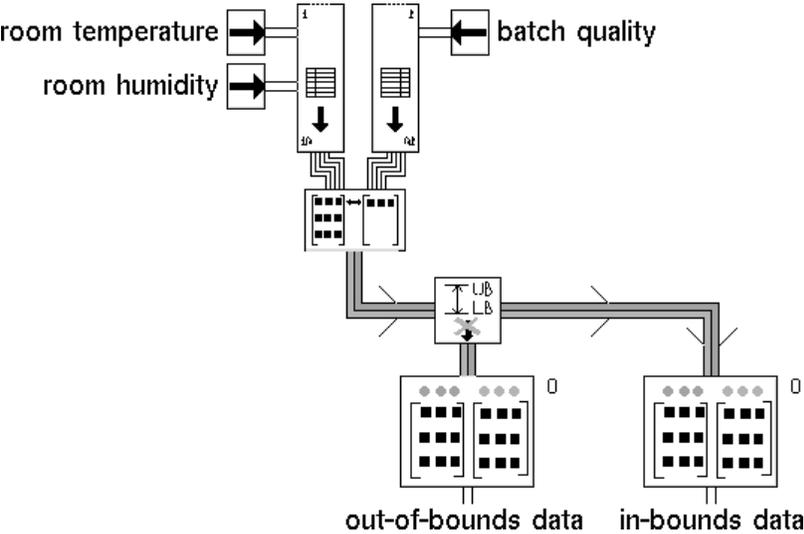
For more information on how to use the spreadsheet, see [Using the GXL Spreadsheet to Edit Data](#).

Making Values Permanent

When you choose make permanent from the block's menu, the block saves the filter's upper and lower bounds.

Example

In the example below, the Data Pair Outlier Filter takes data pairs from a Data Pair Buffer. It passes data pairs that fit into its bounds to the In-Bounds Data Set and the rest to the Out-of-Bounds Data Set.



These spreadsheets show the bounds for the Data Pair Outlier Filter above.

Inputs	Lower	Upper
1	60.0	80.0
2	20.0	70.0

OK Cancel

Targets	Lower	Upper
1	0.0	1.0

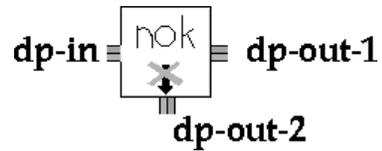
OK Cancel

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Outlier Filter		<i>Reference Manual</i>
Data Pair Quality Filter		<i>Reference Manual</i>
Maximum Age Filter		<i>Reference Manual</i>
Size Limitation Filter		<i>Reference Manual</i>
Novelty Filter		<i>Reference Manual</i>

Data Pair Quality Filter



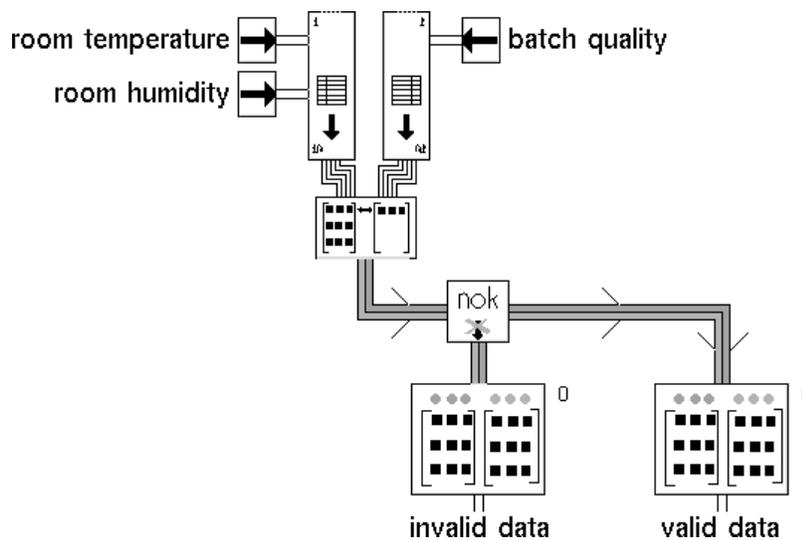
The Data Pair Quality Filter separates out data pairs from a path whose Quality attribute is not OK. data pairs with a Quality of OK are passed through the right output port. Other data pairs are passed through the bottom output port.

Configuring

This block has no configuration panel.

Example

In the example below, the Data Pair Quality Filter takes data pairs from a Data Pair Buffer. It passes data pairs with OK quality to the Valid Data Set and the rest to the Invalid Data Set.

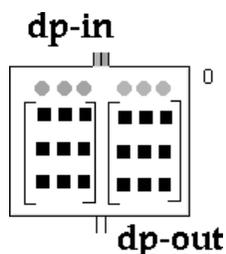


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Outlier Filter		<i>Reference Manual</i>
Data Pair Quality Filter		<i>Reference Manual</i>
Maximum Age Filter		<i>Reference Manual</i>
Size Limitation Filter		<i>Reference Manual</i>
Novelty Filter		<i>Reference Manual</i>

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.

The dp-out of the Data Set is the number of data pairs.

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.

To set the dimensions of the data set, select the edit data set menu choice on the Data Set block. When you first edit a Data Set that contains no data, NeurOn-Line

displays this dialog for entering the Number of Samples, the Number of Inputs, and the Number of Targets:

Enter Data Set Dimensions

Number of Samples

Number of Inputs

Number of Targets

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, NeurOn-Line does not display this dialog. Instead, NeurOn-Line displays the spreadsheet directly.

You can edit the dimensions of the data from the spreadsheet by selecting this button in the spreadsheet:

	Timestamps	Quality	Inputs			Outputs		
	1	1	1	2	3	1	2	3
1	0	OK	10.576	1062.89	629.0	2240.85	1755.81	2190.54
2	0.5	OK	10.643	1081.46	630.2	2237.6	1763.32	2190.4
3	1	OK	10.727	1100.33	631.0	2234.05	1770.41	2189.31
4	1.5	OK	10.812	1119.46	631.3	2230.45	1777.23	2187.56
5	2	OK	10.926	1138.84	631.5	2226.84	1783.75	2185.34
6	2.5	OK	11.017	1158.43	631.5	2223.1	1789.87	2182.73
7	3	OK	11.102	1178.2	631.4	2219.32	1795.53	2179.78
8	3.5	OK	11.187	1198.14	630.7	2215.4	1800.78	2176.65

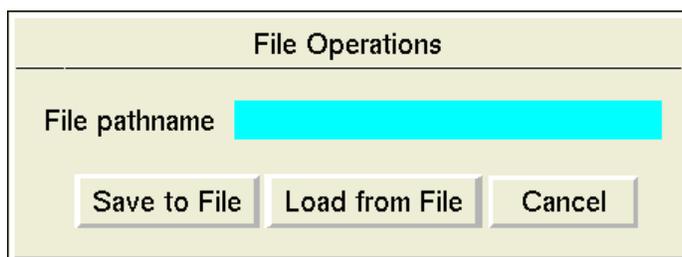
Selecting this buttons displays the Edit Data Set Dimensions dialog for you to edit the dimensions of the existing data.

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. NeurOn-Line 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. NeurOn-Line displays the spreadsheet directly.

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



The samples are numbered down the left side of the editor. The editor shows samples 1 through 8. 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:

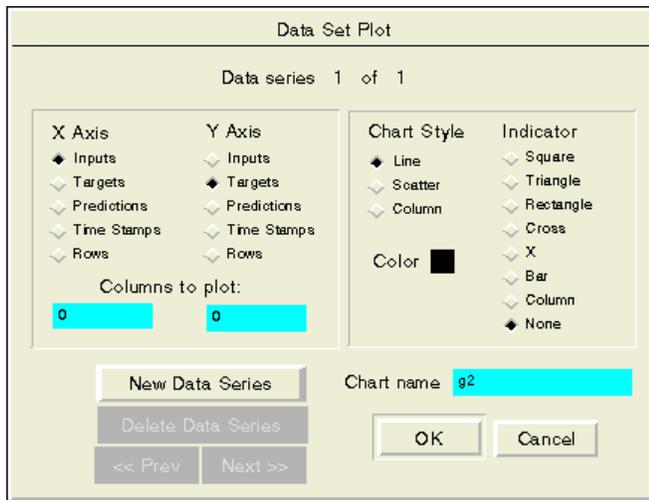
- Editing the spreadsheet cells directly, or
- Reading the data from a file.

For more information on how to use the spreadsheet, see [Using the GXL Spreadsheet to Edit Data](#).

Saving and Loading Data

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

To load a data set from a file, select the file operations menu choice on the Data Set block to display this dialog:



Enter the name of the file from which to load the data, and click the Load from File button.

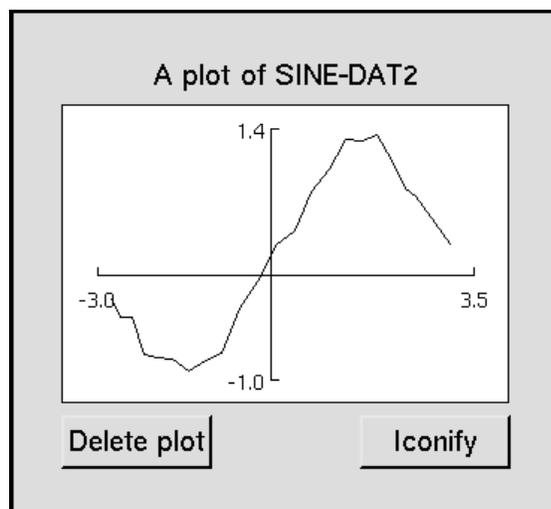
To save a data set to a file, select the file operations menu choice, enter the filename, and click the Save to File button.

You can also load and save parts of the data set by first selecting the cells or rows in the spreadsheet and then using the spreadsheet buttons for loading and saving data.

For more information on how to use the spreadsheet, see [Using the GXL Spreadsheet to Edit Data](#).

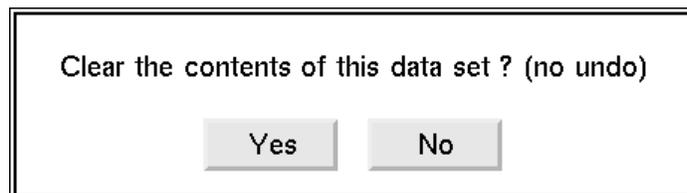
Plotting Data

To create a chart of the Data Set's input target data or predictions, select the plot data menu choice on the Data Set. NeurOn-Line displays this dialog:



For more information on how to use this dialog, see the [Data Set Plot block](#).

There is one difference between how the plot data menu choice and the Data Set Plot block work. If the attribute Chart Name is set to G2, NeurOn-Line creates a workspace for your chart, like the following.



When creating plots from the Data Set configuration dialog, this subworkspace includes two buttons that are not included if you are configuring a Data Set Plot block. If you press Delete Plot, NeurOn-Line deletes the subworkspace and its chart. If you press Iconify, NeurOn-Line creates a Data Set Plot block with the configuration you specified, and attaches it to the Data Set.

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 time stamp for the data pair. It can be either a float or an integer.
 - The quality of the data pair. It can be either 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.

Note For more information on NeurOn-Line's application programmers' interface (API), see [Chapter 12, Application Programmer's Interface](#) in the *NeurOn-Line User's Guide*.

In the data set block'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:

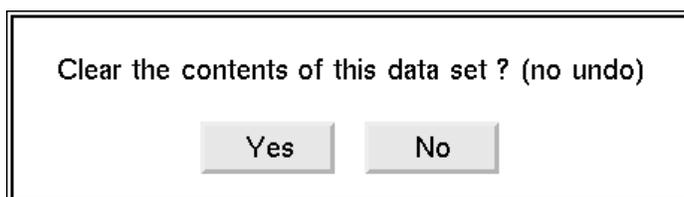
- Input-data-set (class a-matrix).
- Output-data-set (class a-matrix).
- Time-stamps (class quantity-array).
- Qualities (class symbol-array).

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 are provided for saving and loading parts of data sets:

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

Clearing the Data Set

To clear the data set, select the clear data set menu choice. NeurOn-Line displays this dialog:



Click Yes to clear the data set. Click No to keep the data set, unchanged.

Making Values Permanent

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

Configuring

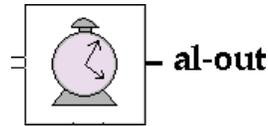
A Data Set has no configurable attributes.

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>
Training Blocks		<i>Reference Manual</i>

Maximum Age Filter

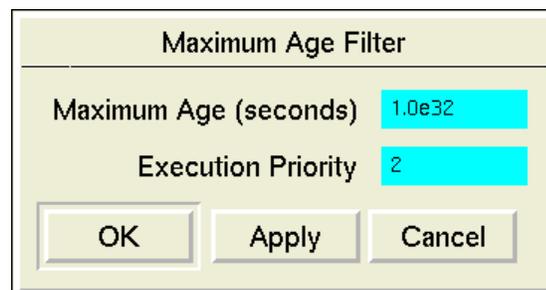


The Maximum Age Filter limits the age of the data pairs stored in the attached Data Set. Whenever the attached Data Set evaluates, the Maximum Age Filter removes any data pair whose age is greater than a specified limit. The filter can also archive the removed data pairs to another Data Set.

To filter a data set, connect the filter's capability link to the Data Set. To archive the removed data pairs in another data set, connect the filter's action link to the other Data Set. Neither of these connections requires a port. To specify the age limit, enter a number of seconds in the attribute Maximum Age. The block computes the age for a data pair by subtracting the data pair's timestamp from the current time.

Configuring

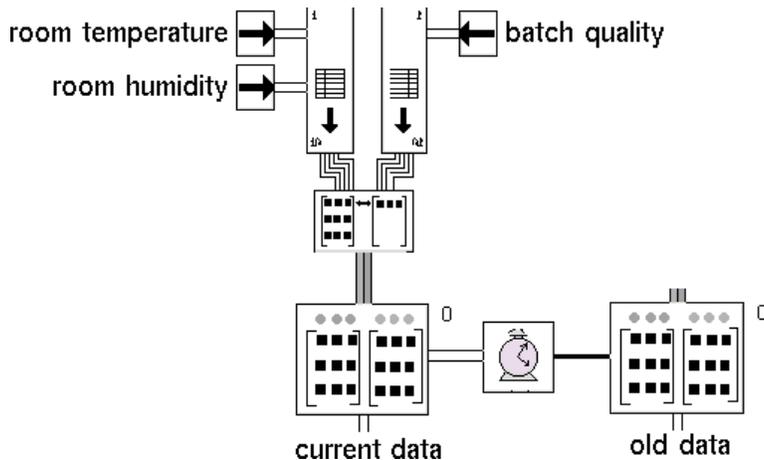
This is the configuration panel for the Maximum Age Filter.



Attribute	Description
Maximum Age (seconds)	The maximum of number of seconds that a data pair can be in the attached Data Set before the filter removes it.
Execution Priority	The order in which to execute this filter, if more than one filter is attached to the Data Set. For more information, see Choosing When a Data Set Filter Executes .

Example

In the example below, whenever the Current Data Set receives a data pair, the Maximum Age Filter removes any data pairs from Current Data that are older than a specified age and places those data pairs in Old Data.

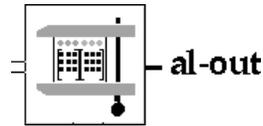


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Outlier Filter		<i>Reference Manual</i>
Data Pair Outlier Filter		<i>Reference Manual</i>
Data Pair Quality Filter		<i>Reference Manual</i>
Size Limitation Filter		<i>Reference Manual</i>
Novelty Filter		<i>Reference Manual</i>

Size Limitation Filter



The Size Limitation Filter limits the number of data pairs stored in the attached Data Set. Whenever the attached Data Set evaluates, the Size Limitation Filter checks its size. If the Data Set contains more data pairs than the maximum you specified, the filter removes enough data pairs from the top of the Data Set to keep the size at the maximum. The filter can also archive the removed data pairs to another Data Set.

To filter a data set, connect the filter's capability link to the Data Set. To archive the removed data pairs in another data set, connect the filter's action link to the other Data Set. Neither of these connections requires a port. To specify the maximum, enter a number in the attribute Maximum Size.

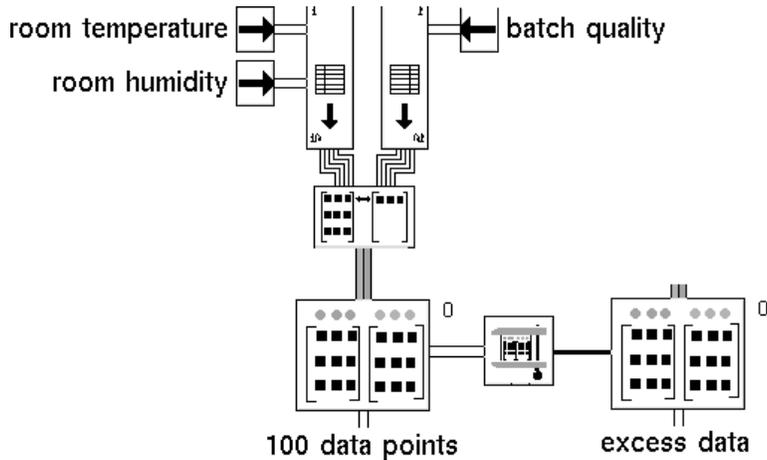
Configuring

This is the configuration panel for the Size Limitation Filter.

Attribute	Description
Maximum Size (Points)	The maximum of number of points that the attached Data Set may contain. When it contains more, the filter removes old data pairs.
Execution Priority	The order in which to execute this filter, if more than one filter is attached to the Data Set. For more information, see Choosing When a Data Set Filter Executes .

Example

In the example below, whenever the 100 Data Points Data Set receives a data pair and the Data Set contains over 100 data pairs, the Size Limitation Filter removes old data pairs from the 100 Data Points Data Set until its size is back to 100. It places the removed data pairs into Excess Data.

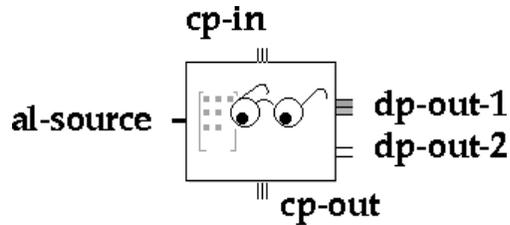


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Outlier Filter		<i>Reference Manual</i>
Data Pair Outlier Filter		<i>Reference Manual</i>
Maximum Age Filter		<i>Reference Manual</i>
Novelty Filter		<i>Reference Manual</i>

Data Set Reader



The Data Set Reader passes a copy of a data pair from the attached Data Set to its output data pair path. The first time it evaluates, it sends out the Data Set's first data pair. The second time it evaluates, it sends out the second. It continues sending out the successive pairs until it comes to the end of the Data Set. Then it starts from the beginning again.

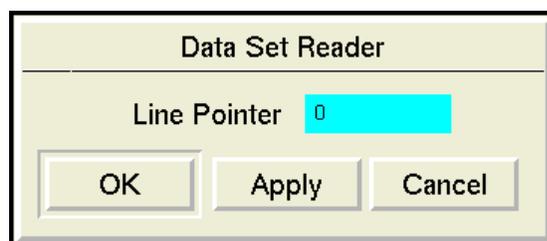
Connect the Data Set to the Data Set Reader's action link. The Data Set Reader passes a data pair out its data pair path and the position of that data pair out its scalar data path. Note that the Data Set Reader does not remove data pairs from the Data Set. It passes copies of them.

Resetting

When you reset the Data Set Reader, it starts reading from the beginning of the Data Set again.

Configuring

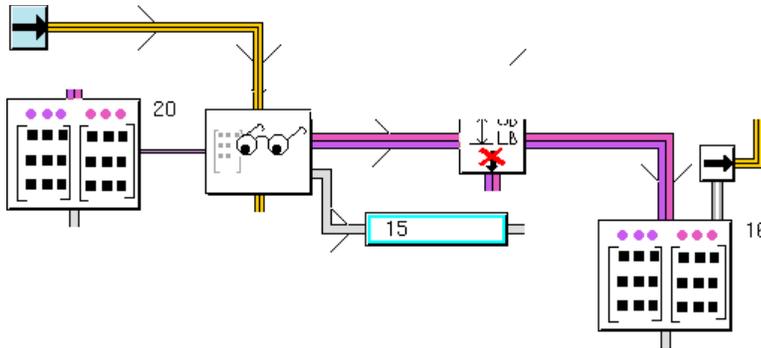
This is the configuration panel for the Data Set Reader.



Attribute	Description
Line Pointer	The next data pair to pass. The block increments this attribute each time it passes a data pair. Remember that the first data pair is number 0.

Example

In the example below, the Data Set Reader is in a loop. It reads points from one Data Set and passes them to a Data Pair Outlier Filter, which puts the points into another Data Set if they are within the filter's bounds. In the picture below, the Data Set Reader has just passed its sixteenth data pair. Since the data pairs are numbered starting with 0, the data pair's scalar output port passes 15.

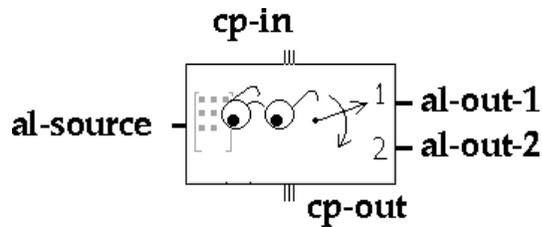


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Set Copier		<i>Reference Manual</i>

Random Divider



The Random Divider randomly copies all the data pairs from one or more Data Sets to two output Data Sets. This block 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.

Connect all the input Data Sets to the left action link. Connect one output Data Set to the top right action link and another output Data Set to the bottom right action link.

When the Random Divider evaluates, it pools the data pairs from all the input data sets together, and randomly copies some to the top output Data Set and the rest to the bottom output Data Set. If the attribute Clear Output Data Set is yes, it clears the output Data Sets before copying. Otherwise, it appends the data pairs to them. You choose the proportion of the data pairs that go to the top output Data Set with the attribute Fraction to Output 1.

Configuring

This is the configuration panel for the Random Divider.

Random Divider

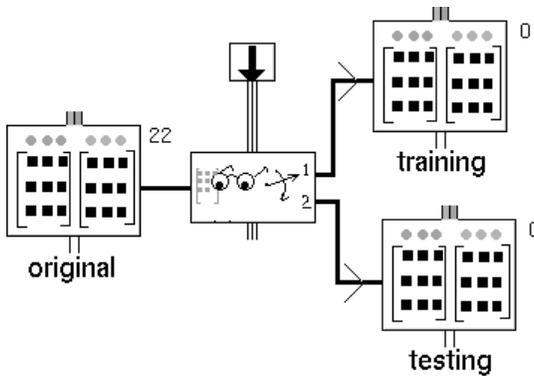
Fraction to Output 1

Clear Output Data Set? yes no

Attribute	Description
Fraction to Output 1	The proportion of the data pairs that go to the top output Data Set.
Clear Output Data Set?	Whether to clear the output Data Sets before adding data pairs to them.

Example

In the example below, the Random Divider splits the Original Data Set into two smaller Data Sets: Training and Testing.

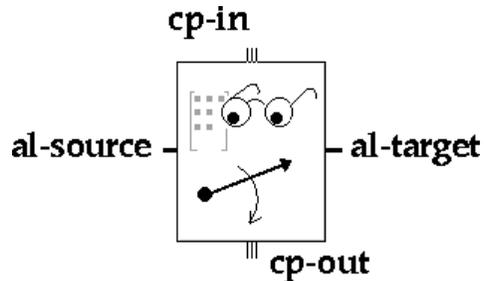


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Pair Divider		<i>Reference Manual</i>
S-Fold Divider		<i>Reference Manual</i>

S-Fold Divider



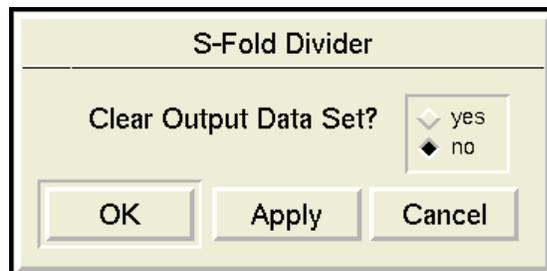
The S-Fold Divider copies all the data pairs from one or more input Data Sets to one or more output Data Sets, randomly dividing the input data between the output Data Sets.

Connect all the input Data Sets to the left action link. Connect all the output Data Sets to the right action link.

When the S-Fold Divider evaluates, it pools the data pairs from all the input data pairs together, and randomly distributes copies of the data pairs to each of the output Data Sets, as evenly as possible. If the attribute Clear Output Data Set is yes, it clears the output Data Sets before copying. Otherwise, it appends the data pairs to them.

Configuring

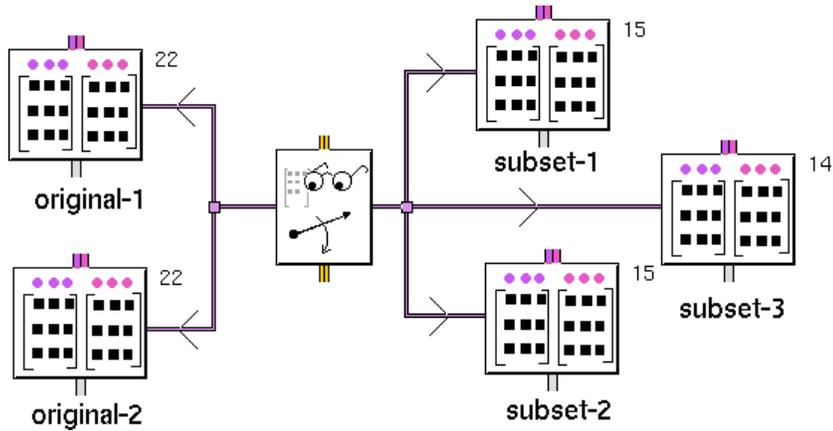
This is the configuration panel for the S-Fold Divider.



Attribute	Description
Clear Output Data Set?	Whether to clear the output Data Sets before adding data pairs to them.

Example

In the example below, the S-Fold Divider pools the Data from the two input Data Sets, randomly splits that pool into three approximately equal subsets, and puts those subsets of the pool into the three output Data Sets.

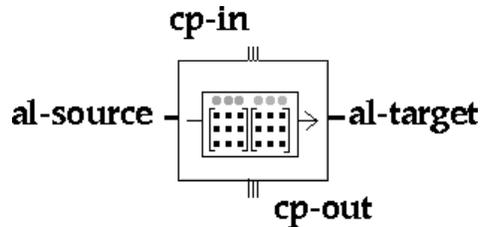


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Pair Divider		<i>Reference Manual</i>
Random Divider		<i>Reference Manual</i>

Data Set Copier



The Data Set Copier copies or combines Data Sets. It pools the data pairs from all its input Data Sets and copies the complete pool to all the output Data Sets.

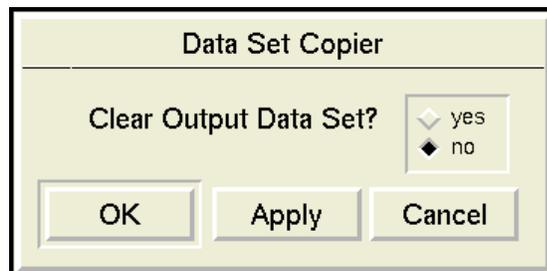
Connect the left action link to all the input Data Sets. Connect the right action link to all the output Data Sets.

When the S-Fold Divider evaluates, it pools the data pairs from all the input data pairs together, and randomly distributes copies of the data pairs to each of the output Data Sets, as evenly as possible. If the attribute Clear Output Data Set is yes, it clears the output Data Sets before copying. Otherwise, it appends the data pairs to them.

If the Clear Output Data Set attribute is yes, the Data Set Copier clears the output Data Sets before copying to them. If Clear Output Data Set is No, the Data Set Copier appends all the new data pairs to the end of each output Data Set. If an output Data Set has smaller input or target matrix dimensions, the Data Set Copier increases the dimensions of the matrix and pads the elements with zeros. If an output Data Set has larger input or target matrix dimensions, the Data Set Copier decreases the dimensions of the matrix and deletes the extra elements.

Configuring

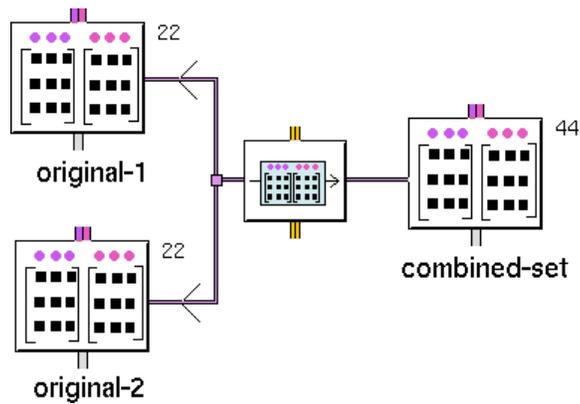
This is the configuration panel for the Data Set Copier.



Attribute	Description
Clear Output Data Set?	Whether to clear the output Data Sets before adding data pairs to them.

Example

In the example below, the Data Set Copier pools the Data from the two input Data Sets, and puts the complete pool in the output Data Set.

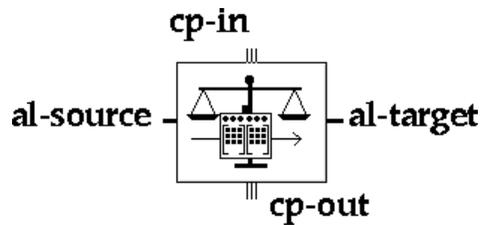


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Data Set Reader		<i>Reference Manual</i>
Data Pair Divider		<i>Reference Manual</i>
Random Divider		<i>Reference Manual</i>

Data Set Rescaler



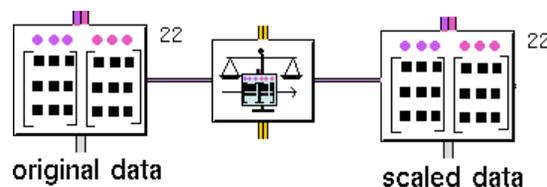
The Data Set Rescaler scales the input and target data in a Data Set. You can specify your own scaling factors or let the Data Set Rescaler create them using one of two standard scaling methods: Min-Max scaling or Mean-Standard Deviation scaling. It can also create two Vector Scaling blocks that scale 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. After you train the network with scaled data, however, 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.

This procedure explains how to use the Data Set Rescaler:

- 1 Attach the Data Sets to the Data Set Rescaler.

Attach the original Data Set to the Data Set Rescaler's left action link and another Data Set to the Data Set Rescaler's right action link.



- 2 Choose the scaling factors.

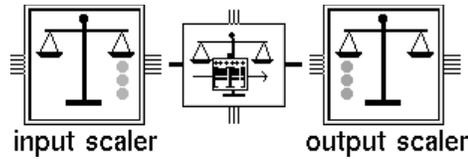
Choose configure from the Data Set Rescaler's menu and select one of the scaling options. For more information, see [Configuring](#).

- 3 Rescale the Data Set.

Either pass the Data Set Rescaler a control signal or choose evaluate from the Data Set Rescaler's menu. The block reads data from the left Data Set and places the scaled data in the right Data Set.

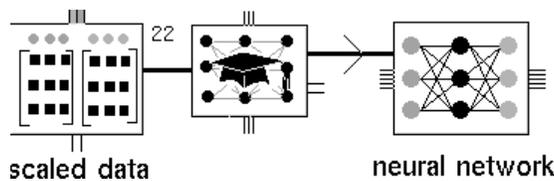
- 4 Create Vector Rescaler blocks.

Choose make vector rescalers from its menu. The block creates two Vector Rescalers and places them near the Data Set Rescaler: the left one contains the scaling factors that the Data Set Rescaler used for input data, and the right one contains the scaling factors the Data Set Rescaler used for target data.



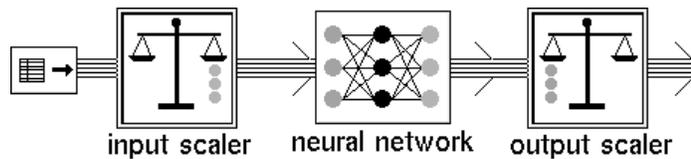
5 Train the neural network with the scaled Data Set.

Attach the scaled Data Set and the neural network to a training block and evaluate the training block.



6 Attach the trained neural network to the Vector Rescalers.

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



Making Values Permanent

When you choose make permanent from the block's menu, the block saves the scale factors.

Configuring

To choose how the Data Set Rescaler performs its scaling, choose configure from the block's menu. NeurOn-Line displays this configuration panel.

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. These are the options:

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 a table for entering custom scale factors for a target vector of width 2:

Targets	Additive	Multiplicative
1	0.0	1.0
2	0.0	1.0

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:

$$Column'_i = M_i (Column_i + A_i)$$

For more information on how to use the spreadsheet, see [Using the GXL Spreadsheet to Edit Data](#).

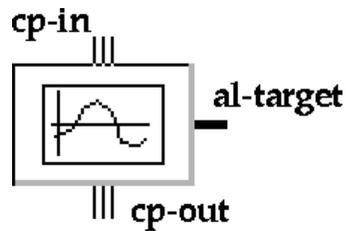
If the attribute Clear Output is yes, the block clears the output Data Set before it copies the rescaled data to it. Otherwise, the block appends the rescaled data to the end of the Data Set.

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>
Vector Rescaler		<i>Reference Manual</i>

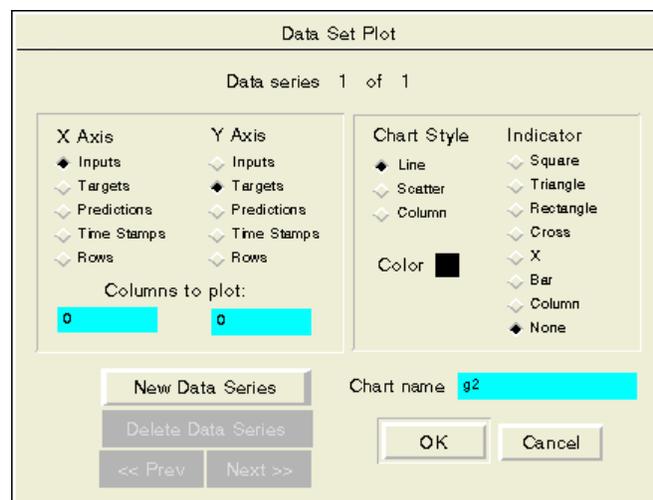
Data Set Plot



The Data Set Plot block lets you display charts of the data stored in the Data Set attached to its action link.

Configuring

When you choose configure from the block's menu, and you have attached a Data Set to its action link, the block displays the configuration panel below.



The following headings describe how to interact with the configuration panel.

Choosing What to Display

To specify what data from the Data Set is displayed on the chart, specify one of these options for X Axis and Y Axis:

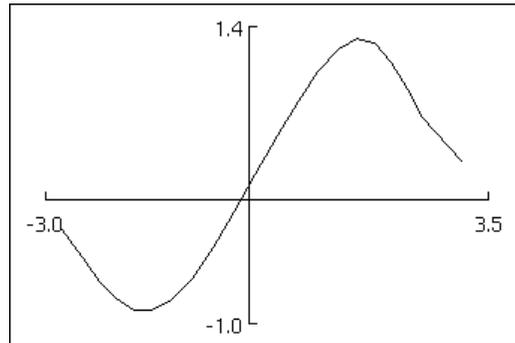
Option	Description
Inputs	The input values
Targets	The target values for neural network training

Predictions	The predictions of a neural network
Time Stamps	The time the sample was received
Rows	The row number for this sample

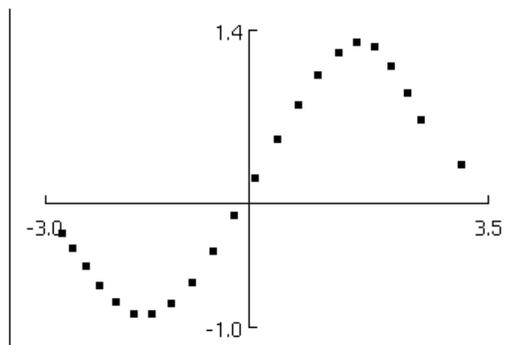
If the data set has more than one column of data for the option you selected, enter the number for the column you want displayed in the field under the options.

Choosing How to Display the Data

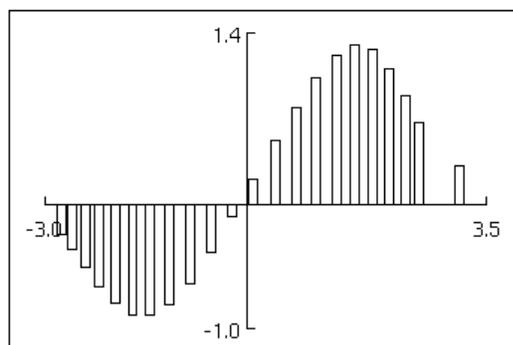
To specify how the data is displayed, use the options on the right of the dialog. Choose the type of chart from the options listed under Chart Style. You can choose a line, scatter, or column chart, as shown below.



Line



Scatter



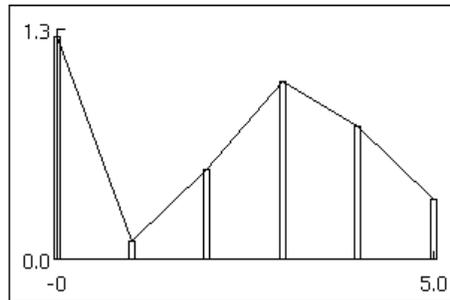
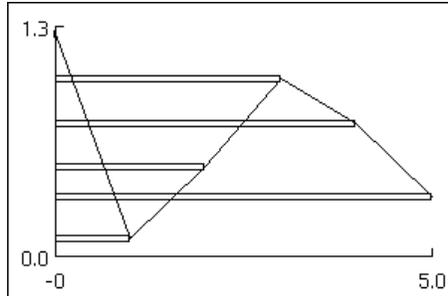
Column

To put a marker on each point in a data series, choose one of the options listed under Indicator. If the Chart Style is Line, you can choose any of these options. If

the Chart Style is Scatter, the option None is not available. If the Chart Style is Column, only the option Column is available. The chart below shows you what the indicators look like.

Option	Icon
Square	
Rectangle	
Triangle	
Cross	
X	
Bar	See below
Column	See below
None	Points are unmarked

The bar and column indicators are a little different from the rest. They draw a line from the point to an axis. A bar goes to the Y axis, and a column goes to the X axis, as shown below.

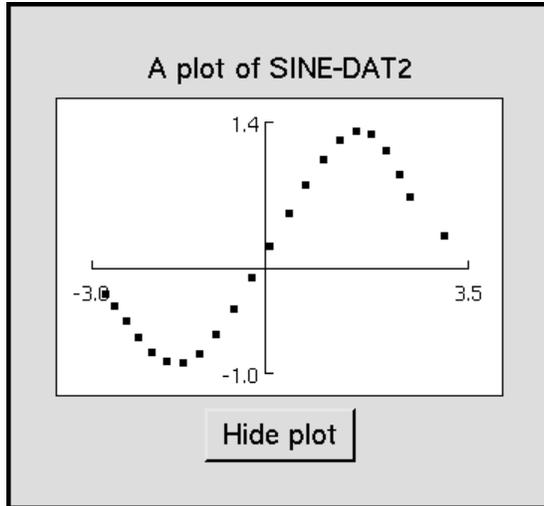


To choose a color for the data series, use the Color list. It contains all the available G2 colors. You can scroll through the list by using the scroll bar on the left. The names of the unselected colors are black. When you select a color, its name becomes white.

Choosing Where to Display the Data

To specify the chart to display the data on, enter the chart's name in the attribute Chart Name.

If you enter G2 as the chart name, the Data Set Plot block creates a subworkspace for itself and places a chart on that workspace, as shown below. To hide the subworkspace, click the Hide Plot button.



Creating and Deleting Data Series

Sometimes you might want to display more than one view of your data on a single chart. For example, you might want to plot a view of the inputs versus the targets, and a view of the inputs versus the predictions. Each of these views is called a data series. The second line from the top of the dialog displays which data series you are editing and how many data series there are for this data set. (For example, if the second line reads **Data Series 2 of 3**, there are three data series and you are editing the second.) The buttons in the lower left corner of the dialog let you create, delete, and move between data series.

By default, the Data Set Plot block has only one data series. To create a new data series, click the New Data Series button. The dialog changes the settings of the options to their defaults. To delete the currently displayed data series, click the Delete Data Series button.

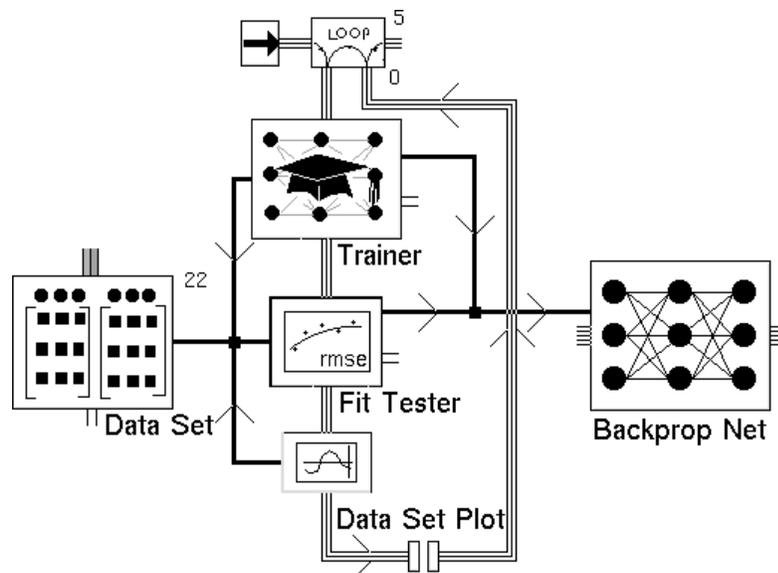
To move between data series, use the Next>> and <<Prev buttons. When you go to a data series, the dialog changes the settings of the options to the series' settings.

Making Values Permanent

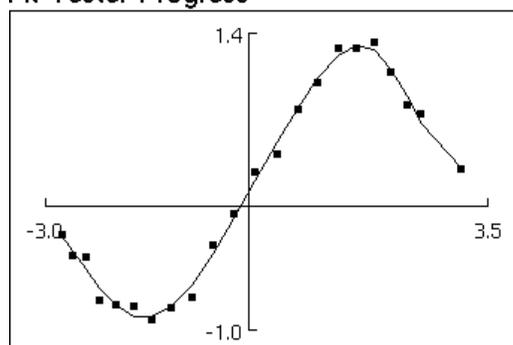
When you choose make permanent from the block's menu, the block saves the data set.

Example

In the diagram below, the Data Set Plot block graphs the progress of the Fit Test block. The Data Set Plot block has two data series: one to show the target values and one to show the neural net's predictions. In the first data series, the X Axis is inputs, the Y Axis is predictions, the Chart Style is Line, and the Indicator is None. In the second data series, the X Axis is inputs, the Y Axis is Targets, the Chart Style is Scatter, and the Indicator is Square.



Fit Tester Progress

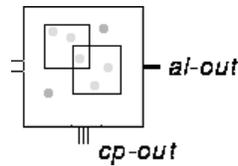


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>
Plotting Data		<i>Reference Manual</i>

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. The filter can also archive the removed data pairs to another Data Set.

To filter a Data Set, connect the filter's capability link to it. To archive the removed data pairs in another Data Set, connect the filter's action link to it. Neither of these connections requires a port in the Data Set.

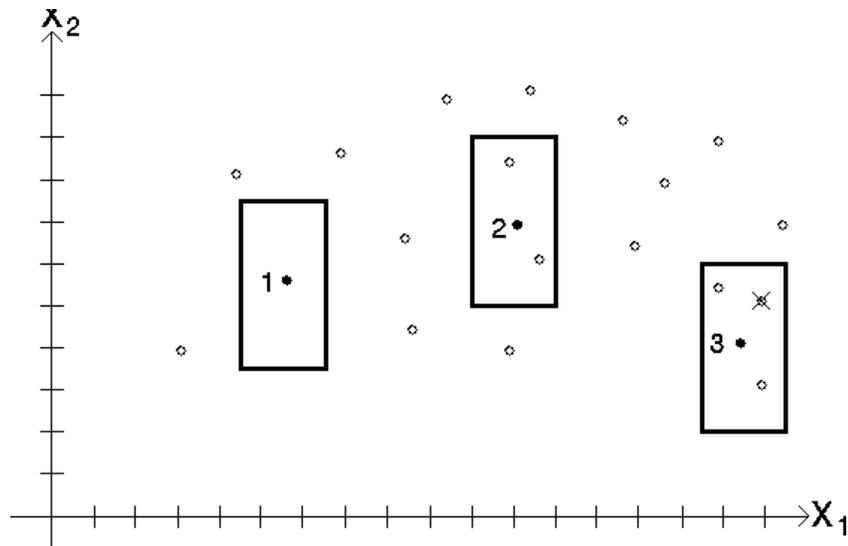
The Novelty Filter sends a control signal on its cp-out path whenever the filter detects a novel data pair.

Choosing Which Points to Keep

Whenever the attached 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 attribute Points per Cell, the Novelty Filter removes the oldest data pair from it. You set the sizes of the cell in the configuration panel. 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 configuration panel, the size for X1

is 1 and the size for X_2 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 following list describes how the Novelty Filter handles the three new points:

- 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 this picture, 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

When you select make permanent from the block's menu, the block saves the sizes of all the input and output values.

Configuring

This is the configuration panel for the Novelty Filter.

Novelty Filter	
<input type="button" value="Edit Input Cell Sizes"/> <input type="button" value="Edit Target Cell Sizes"/>	
Number of Inputs	1
Number of Targets	1
Points Per Cell	0
Execution Priority	1
<input type="button" value="OK"/> <input type="button" value="Apply"/> <input type="button" value="Cancel"/>	

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.

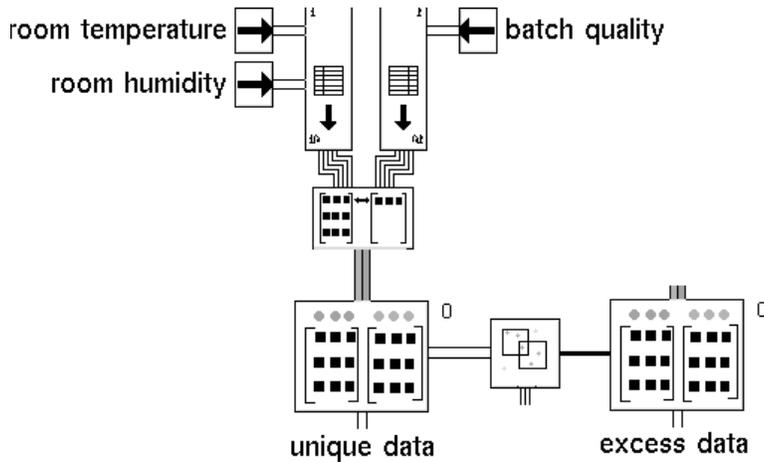
Once you have specified these attributes, click Edit Input Cell Sizes and Edit Target Cell Sizes to edit the cell sizes. The block displays a spreadsheet, which lets you enter the size for each input and output value.

For more information on the spreadsheet, see [Using the GXL Spreadsheet to Edit Data](#).

When you connect more than one filter to a Data Set, Execution Priority lets you determine when each executes. For more information, see [Choosing When a Data Set Filter Executes](#).

Example

In the example below, whenever the Current Data Set receives a data pair, the Novelty Filter removes any data pairs from the Unique Data Set that are close to it and places those data pairs in the Excess Data Set.



See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>
Outlier Filter		<i>Reference Manual</i>
Data Pair Outlier Filter		<i>Reference Manual</i>
First-Order Exponential Filter		<i>Reference Manual</i>
Data Pair Quality Filter		<i>Reference Manual</i>
Maximum Age Filter		<i>Reference Manual</i>
Size Limitation Filter		<i>Reference Manual</i>

Neural Networks

Chapter 7: Neural Network Blocks

Describes the blocks that let you create feed-forward, layered networks.

Chapter 8: Training Blocks

Describes the blocks that test and train networks.

Neural Network Blocks

Describes the blocks that let you create feed-forward, layered networks.

Introduction **216**

Backpropagation Net (BPN) **221**

Autoassociative Net **227**

Radial Basis Function Net (RBFN) **231**

Rho Net **234**

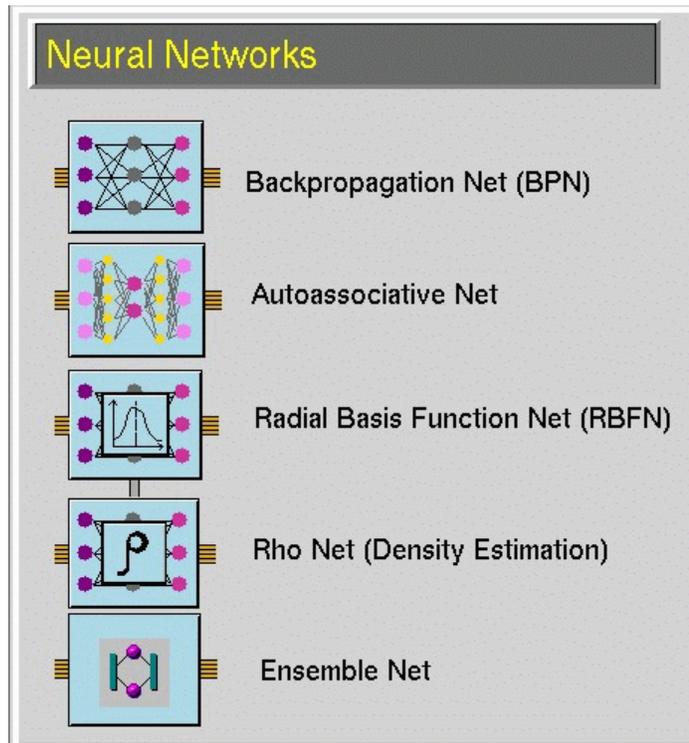
Ensemble Net (ENN) **238**



Introduction

The Neural Network blocks are the heart of NeurOn-Line.

You can find the Neural Networks palette under the Neural Networks submenu of the Palettes menu:



Saving and Loading Network Weights

All the Neural Network blocks let you save and load what they have learned so far. NeurOn-Line saves the information to text files, which you can examine yourself. This section describes the formats used for these files.

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 $N \times H$ 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.

Backpropagation and Autoassociative Networks

Both the [Backpropagation Net \(BPN\)](#) block and the [Autoassociative Net](#) block 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.

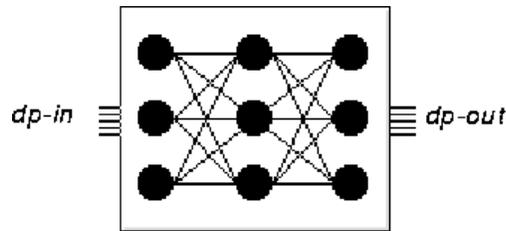
Radial Basis Function and Rho Networks

Both the [Radial Basis Function Net \(RBFN\)](#) block and the Rho Net block 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 \(ENN\)](#) block is an encapsulation block. 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.

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.

Note NeurOn-Line does not use G2 objects to represent the nodes and connections in a BPN. Instead, NeurOn-Line stores the network internally and lets you change the network's architecture with the `configure` menu choice.

Before you can pass data through a network, you must train the network. For more information, see [Chapter 8, Training Blocks](#). 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 block.

When you select configure on the block, NeurOn-Line displays this dialog:

Layer	Nodes	Transfer Function
1	1	linear
2	1	sigmoid
3	1	linear
4	0.0	
5	0.0	

To set the number of layers, use the arrows to the right of the Number of Layers attribute. To increase the number, click the up arrow. To decrease the number, click the down arrow. You can select 2 to 5 layers.

The rows below Number of Layers let you 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.

To set the number of nodes for a layer, enter a number in the Nodes attribute for each layer. To set the transfer function for a layer, click the linear or sigmoid button for the Transfer Function attribute for each layer. Selecting the button toggles the button between linear and sigmoid.

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 off 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, click the randomize

weights button. NeurOn-Line overwrites the block's current weights with new random weights.

Caution: When you click the randomize weights button, NeurOn-Line immediately commits your changes. Clicking the Cancel button does not discard them.

NeurOn-Line displays the dialog below, which lets you specify an absolute amount to randomize by, a percentage to randomize by, or both:

Randomize Weights

Enter percentage and/or absolute amount to randomly adjust weights:

Absolute

Percentage

This is the formula that the block 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.

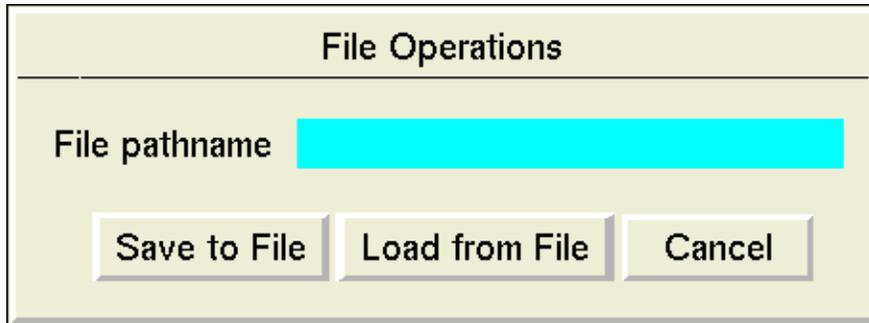
$$\text{NewWeight} = (1 + R_1 \cdot P/100) \cdot \text{OldWeight} + R_2 \cdot A$$

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](#).

To save or load network data, select the file operations menu choice to display this dialog:



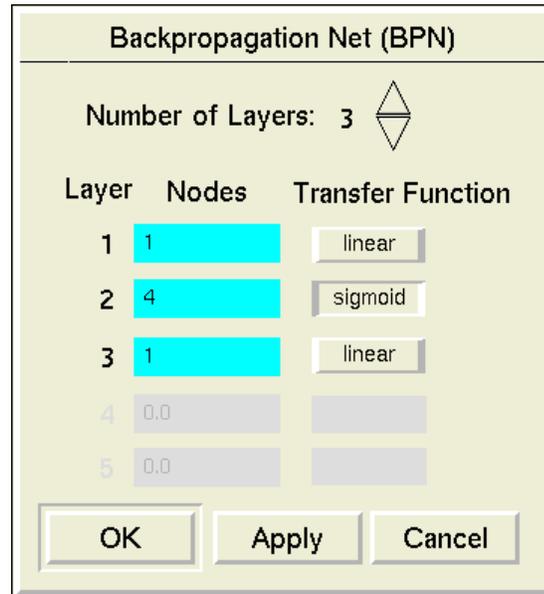
Edit the File Pathname attribute to specify the filename. To save the weights, click the Save to File button. To load weights, overwriting the weights that are currently in the network, click the Load from File button.

Making Values Permanent

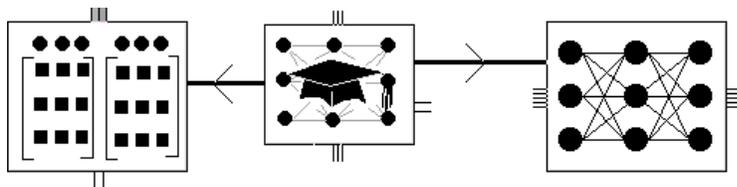
When you choose make permanent from the block's menu, it saves the network's internal configuration and weight so that resetting G2 has no effect on their values.

Examples

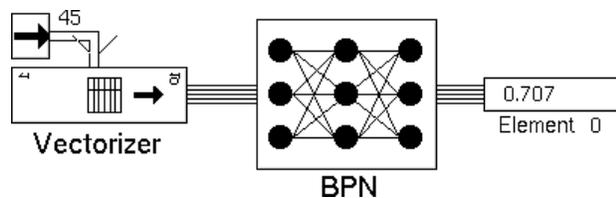
The following configure dialog is for a BPN that is being trained with a data set of angles and their sines. It contains three layers: the input layer, the output layer, and one hidden layer. Both the input and output layers have 1 node and use the linear function. The hidden layer has 4 nodes and uses the sigmoid function.



Below is a simple diagram for training a network. The data set on the left is filled with a sample of angles and their sines. The BPN is configured as described above. To train this network, choose **configure** from the BPN's menu and click **Randomize Weights**, then choose **evaluate** from the Training block's menu.



This diagram uses a trained BPN to estimate the sine for of a 45-degree angle:

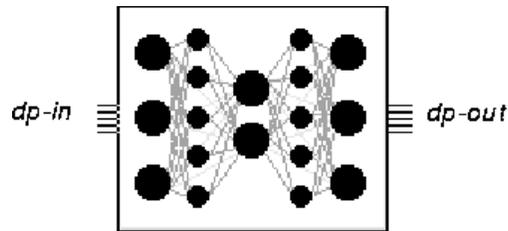


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>

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.

Note NeurOn-Line does not use G2 objects to represent the nodes and connections in an Autoassociative Network. Instead, NeurOn-Line stores the network internally and lets you change the network's architecture with the configure menu item.

Before you can pass data through a network, you must train the network. For more information, see [Chapter 8, Training Blocks](#). 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, NOL 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

When you choose **configure** from the block's menu, NeurOn-Line displays this dialog.

Layer	Nodes	Transfer Function
1	1	linear
2	1	sigmoid
3	1	linear
4	1	sigmoid
5	1	linear

Run Mode: filter noise only, correct gross errors

noise standard deviations

OK Apply Cancel

To configure the Network Architecture, choose the number of layers, specify the number of nodes, and specify the transfer functions for each layer.

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.

To set the number of layers, use the arrows to the right of the Number of Layers attribute. By default, the network has 5 layers, which is the recommended number for an Autoassociative network. To decrease the number, click the down arrow. To increase the number, click the up arrow. You can select 2 to 5 layers.

The rows below Number of Layers let you set the number of nodes and transfer function for each input layer. 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.

To set the number of nodes for a layer, enter a number in the Nodes attribute for each layer. To set the transfer function for a layer, click the linear or sigmoid button for the Transfer Function attribute for each layer. Selecting the button toggles the button between linear and sigmoid. 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 Run Mode attribute lets you choose whether the network replaces faulty input values. If you choose the **filter noise only** option, 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, the Noise Standard Deviations button becomes active. Click this button to display a spreadsheet that lets you choose the standard deviations the block uses.

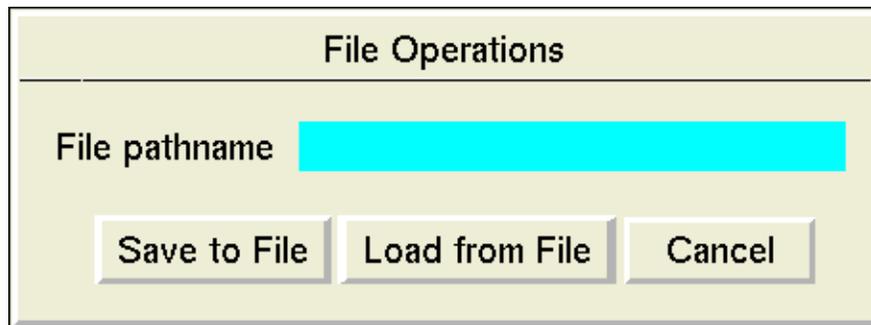
Adjusting Weights

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 block.

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 block.

To save or load network data, select the file operations menu choice to display this dialog:



Edit the File Pathname attribute to specify the filename. To save the weights, click the Save to File button. To load weights, overwriting the weights that are currently in the network, click the Load from File button.

Making Values Permanent

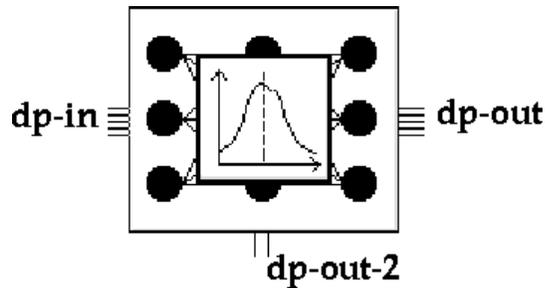
When you choose make permanent from the block's menu, it saves the network's internal configuration and weights.

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>

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.

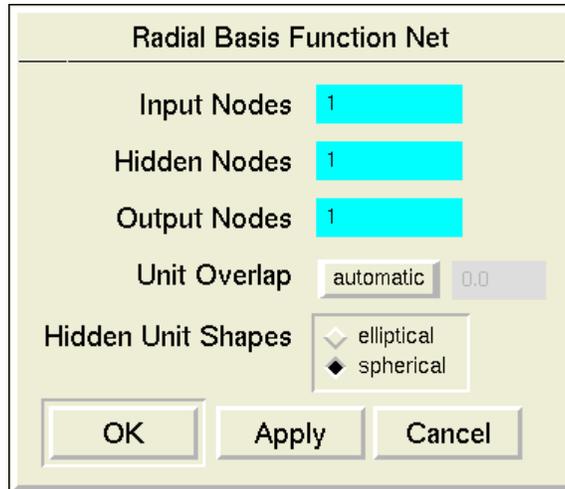
The transfer function of the input and output layer is linear. You can choose whether the transfer functions of the hidden layer are spherical or elliptical Gaussians.

Note NeurOn-Line does not use G2 objects to represent the nodes and connections in a neural network. Instead, NeurOn-Line stores the network internally and lets you change the network's architecture with the configure menu item.

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

When you choose configure from the block's menu, it displays the dialog below.



The image shows a configuration dialog box titled "Radial Basis Function Net". It contains several fields and controls:

- Input Nodes:** A text field containing the number "1".
- Hidden Nodes:** A text field containing the number "1".
- Output Nodes:** A text field containing the number "1".
- Unit Overlap:** A dropdown menu set to "automatic" and a numeric input field set to "0.0".
- Hidden Unit Shapes:** A group box containing two radio buttons: "elliptical" (which is selected) and "spherical".
- Buttons:** "OK", "Apply", and "Cancel" buttons at the bottom.

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.

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 number of nodes in the input layer, enter a number in the Input Nodes field. To set the number of nodes in the hidden layer, enter a number in the Hidden Nodes field. To set the number of nodes in the output layer, enter a number in the Output Nodes field.

To set the unit overlap, choose whether the overlap is automatic or fixed by selecting the toggle button next to the Unit Overlap attribute. 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, enter a positive value in the Unit Overlap attribute edit box. 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.

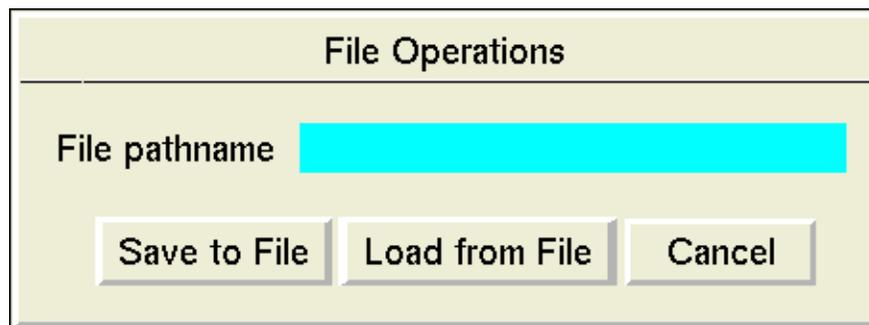
To choose the function shape for the hidden layer, select one of the options under Hidden Unit Shapes: **spherical** or **elliptical**. When data is sparse or the input values are not correlated to each other, spherical units may perform better. When more data is available or the input values are correlated 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. To choose the option for other cases, you may need to perform cross-validation with the Train and Test block or the Five-Fold CV block.

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 block.

To save or load network data, select the file operations menu choice to display this dialog:



Edit the File Pathname attribute to specify the filename. To save the weights, click the Save to File button. To load weights, overwriting the weights that are currently in the network, click the Load from File button.

Making Values Permanent

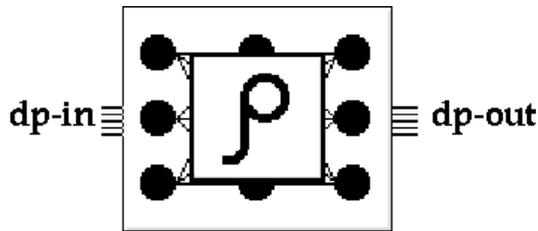
When you choose make permanent from the block's menu, it saves the network's internal configuration and weights.

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>
Rho Net		<i>Reference Manual</i>

Rho Net



The Rho Network is a 3-layer, feed-forward network, whose middle layer uses a multi-variate Gaussian 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.

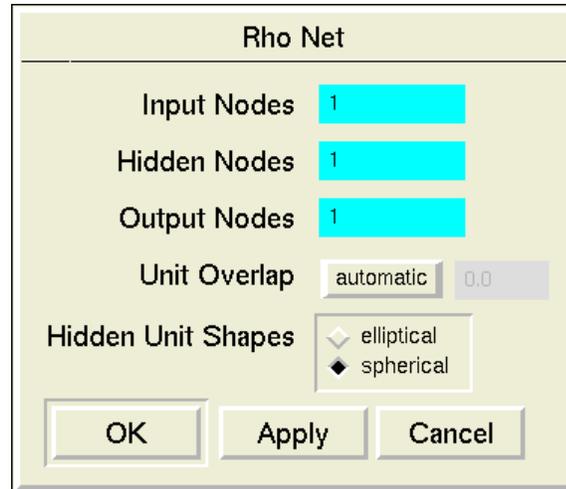
The transfer functions of the input and output layers are linear. You can choose whether the shape for the transfer functions of the hidden layer are spherical or elliptical.

Note NeurOn-Line does not use G2 objects to represent the nodes and connections in a neural network. Instead, NeurOn-Line stores the network internally and lets you change the network's architecture with the configure menu item.

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 connect a Trainer block to a Rho Network and choose **configure** from the Trainer's menu, 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

When you choose configure from the block's menu, it displays the dialog below.



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.

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 number of nodes in the input layer, enter a number in the Input Nodes field. To set the number of nodes in the hidden layer, enter a number in the Hidden Nodes field. To set the number of nodes in the output layer, enter a number in the Output Nodes field.

To set the unit overlap, choose whether the overlap is automatic or fixed by selecting the toggle button next to the Unit Overlap attribute. 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.

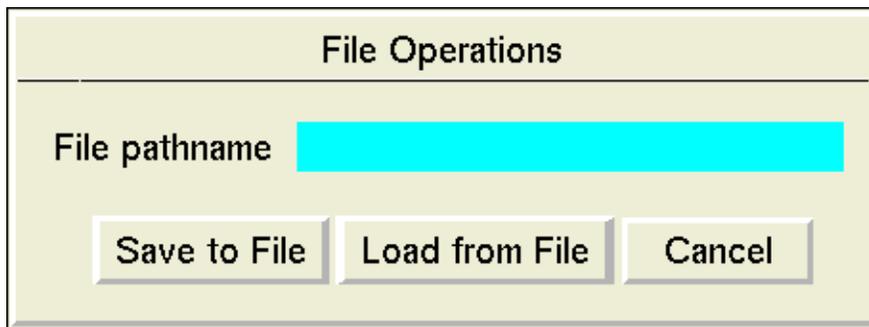
If the overlap is fixed, enter a positive value in the Unit Overlap attribute edit box. The overlap parameter is a multiplicative factor applied to a basic hidden unit width, which is the nearest neighbor distance between the radial units. If the Unit Overlap is 1.0, each hidden unit's width is the distance 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, select one of the options under Hidden Unit Shapes: 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. To choose the option for other cases, you may need to perform cross-validation with the Train and Test block or the Five-Fold CV block.

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 block.

To save or load network data, select the file operations menu choice to display this dialog:



Edit the File Pathname attribute to specify the filename. To save the weights, click the Save to File button. To load weights, overwriting the weights that are currently in the network, click the Load from File button.

Making Values Permanent

When you choose make permanent from the block's menu, it saves the network's internal configuration and weights.

See Also

For general information on how to use this block, see the sections below.

For more information on...

See...

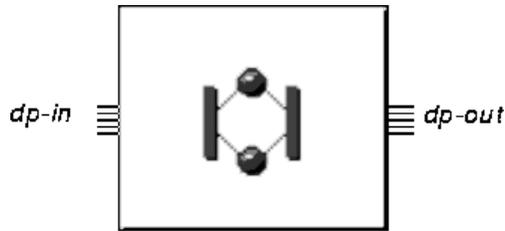
In this book...

Basic Block Behavior

User's Guide

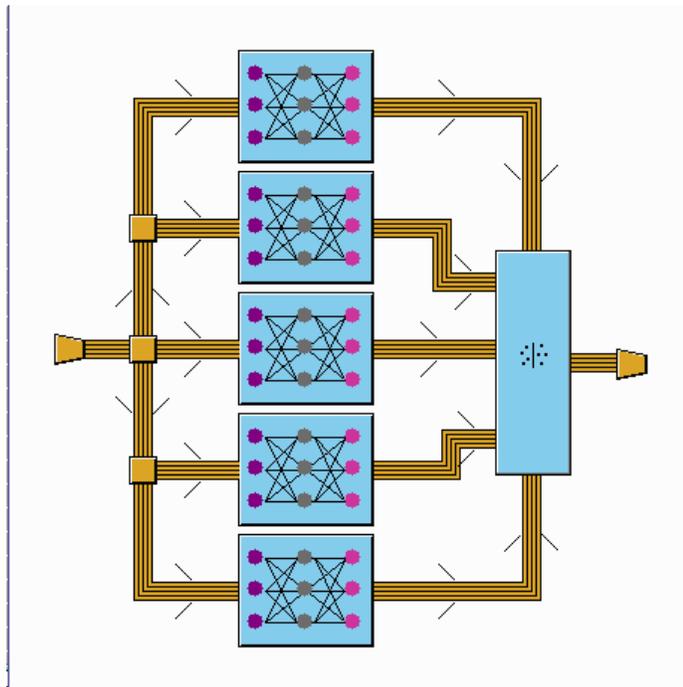
For more information on...	See...	In this book...
Saving a Block's Data After Resetting G2		<i>User's Guide</i>
Radial Basis Function Net (RBFN)		<i>Reference Manual</i>

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 Ensemble Net block is an encapsulation block whose subworkspace is shown below:



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 block, 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*.

Note NeurOn-Line does not let you configure the architecture of an Ensemble Net. The architecture of an Ensemble Net is automatically determined when it is trained within NOL Studio.

Before you can pass data through an Ensemble Net the first time, you must initialize the ENN block 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 block to calculate the median values of the output from all submodels. The output vector of the Ensemble Net is composed of these median values.

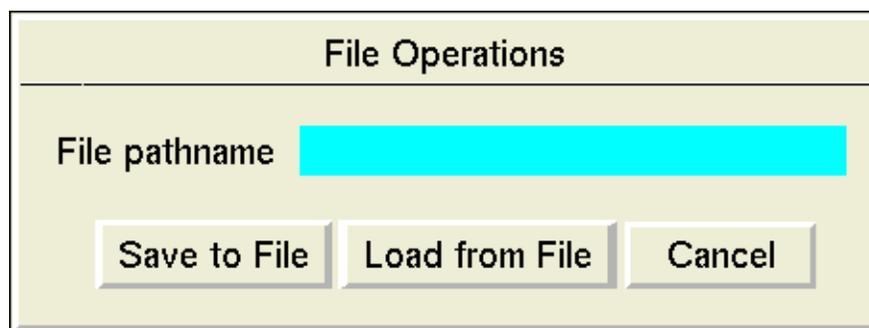
Adjusting Weights

When you first create an ENN block, 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. For more information, see [Chapter 8, Training Blocks](#).

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 block.

To save or load network data, select the file operations menu choice to display this dialog:



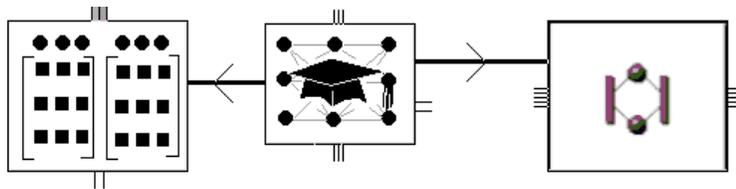
Edit the File Pathname attribute to specify the filename. To save the weights, click the Save to File button. To load weights, overwriting the weights that are currently in the network, click the Load from File button.

Making Values Permanent

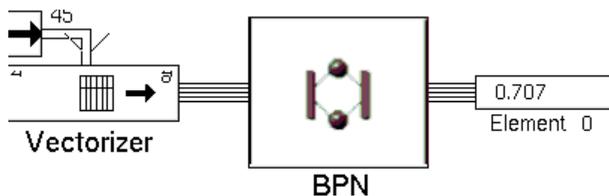
When you choose **make permanent** from the block's menu, it saves the network's internal configuration and weight so that resetting G2 has no effect on their values.

Examples

Below is a simple diagram for training an Ensemble Network. The data set on the left is filled with data collected in real time. The ENN is loaded from a text file exported from NOL Studio. The data set must have the same number of inputs and outputs as specified in the model architecture. The data should be filtered to remove outliers. To train the network, connect a trainer block to the Ensemble Network, then choose **evaluate** from the Training block's menu.



The deployment of ENN is similar to the deployment of BPN. This diagram uses a trained ENN to estimate the sine of a 45-degree angle, just as in the example of BPN.



See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>

Training Blocks

Describes the blocks that test and train networks.

Introduction **241**

Trainer **244**

Fit Tester **251**

Train and Test **255**

Five Fold CV **260**

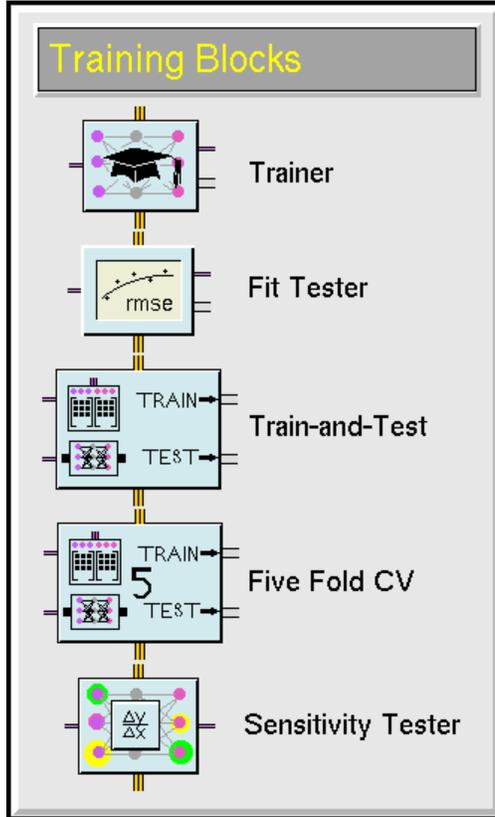
Sensitivity Tester **264**



Introduction

NeurOn-Line comes with several blocks that test and train neural networks.

You will find the Training Blocks palette under the Neural Networks submenu of the Palettes menu:



Usually, you will follow this procedure to find the best neural network and train it:

1 Find the best neural network configuration.

Use either the Train and Test Block or the Five Fold CV block with several different neural networks to find which network solves your problem best. You can test different types of neural networks and different configurations of the same neural network.

You can also create your own testing algorithms with the Trainer and Fit Tester blocks.

2 Train the neural network.

After you find the best neural network configuration, use the Trainer block with that network and all available data to train the network.

You might even add a step after the last step. If you are not sure whether all the input values you have actually affect the output, use a Sensitivity Tester. Removing unimportant inputs and repeating the training cycle may improve the performance of your final network.

Basic Training and Testing

These blocks perform basic training and testing:

- The [Trainer](#) block trains a neural network on a particular data set using a training method that you specify.
- The [Fit Tester](#) block 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 test case.

Finding the Best Network Configuration

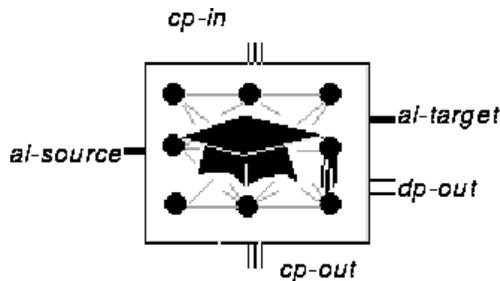
These blocks help you find the best neural network configuration for your problem:

- The [Train and Test](#) block trains and tests your network a specified number of times, each time randomly splitting the data set into two subsets: training and testing.
- The [Five Fold CV](#) block splits the data set into five subsets and then trains and tests your network a total of five times, each time using a different one of the five subsets for training.

Finding Which Inputs are Significant

The [Sensitivity Tester](#) block determines which inputs of a neural network you actually need to predict the output. It accomplishes this by testing the influence of each input on each output by using a trained neural network and a data set of sample data.

Trainer



The Trainer block can train any neural network in NeurOn-Line. Attach one of its action links to a neural network, and attach the other action link to a data set. You cannot attach more than one neural network or data set to the block.

To configure the Trainer block, first, it must be connected to a neural network, then you can choose **configure** from the Trainer block's menu. The configuration panel contains different options depending on what type of neural network is connected to it. For more information, see [Configuring the Trainer for a Backpropagation, Autoassociative, or Ensemble Network](#).

To evaluate the Trainer block, either pass it a control signal or choose **evaluate** from its menu. The Trainer trains the neural network with the data from the data set by adjusting the network's weights and other internal parameters. The Trainer Block does not modify the network's architecture or modify any data in the data set. You may evaluate either the attached neural network or data set during training. If you evaluate the neural network during training, the output is based on the weights before the training started.

To cancel training, choose Controls > Remote Process > Kill and then reset the block. The neural network's weights and other internal parameters are left unchanged and the Trainer outputs nothing.

When the Training block finishes evaluating, it passes a control signal on cp-out and a scalar value on dp-out. The scalar value tells you how well it trained. If you are training a Backpropagation, Autoassociative, or Radial Basis Function Network, it passes the square root of the mean of the squared errors for all the training data. If you are training a Rho Network, it passes the negative mean of the logarithms of the predicted probabilities for all training data. In both cases, lower numbers mean a closer fit of the training data.

Watching the Training Happen

Sometimes you may want to watch the Trainer's progress, especially for long training runs. When you select Enable RPC from the Remote Procedure menu in the Print Configuration Tools menu bar, the Trainer displays an output of its progress. If you launched the NeurOn-Line remote procedure yourself, the table

appears in the window where you launched it. If you let NeurOn-Line launch the remote procedure automatically, the table will be saved in a log file. For more information, see “Launching a G2 Process” in G2 Reference Manual.

Here is a sample of the output for Backpropagation Network training:

<i>GRADIENTS</i>	<i>FUNCTIONS</i>	<i>OBJECTIVE</i>	<i>METHOD</i>
0	1	49.87	
1	4	3.65	<i>steepest descent</i>
2	7	3.56	<i>Broyden</i>
3	11	3.445	<i>Broyden</i>
4	15	3.239	<i>Broyden</i>
6	18	2.81	<i>steepest descent</i>
7	23	2.402	<i>Broyden</i>
8	26	1.54	<i>Broyden</i>
9	30	1.041	<i>Broyden</i>
10	33	0.9416	<i>Broyden</i>
11	36	0.8692	<i>Broyden</i>
12	39	0.8096	<i>Broyden</i>

Maximum number of function calls exceeded

The first column is how many passes, or gradient calculations, have happened so far. Each pass represents a change of direction as the Trainer searches for the best weights.

The second column is how many times the Trainer needed to evaluate the objective function in that pass. Each evaluation of the objective function represents one “step,” and the total is bounded by the maximum iterations (in this case 40). Typically, the trainer takes 3 to 5 steps in a given direction before it calculates a new gradient.

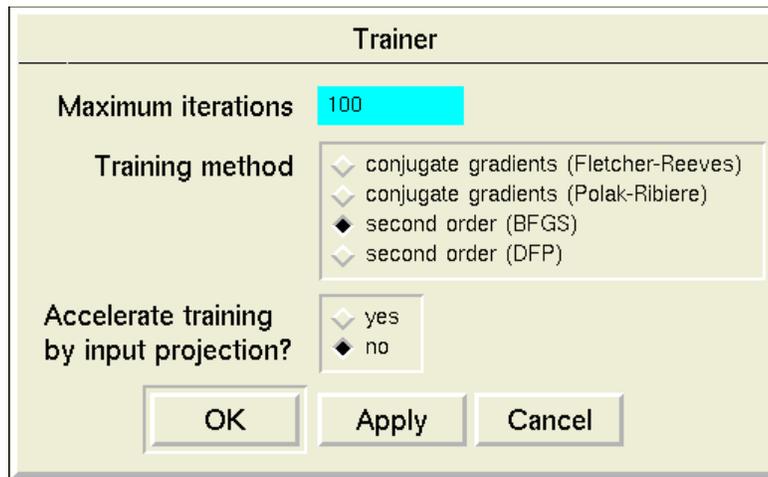
The third column is the value of the least-squares objective function. The lower the value, the closer the Trainer is to a good fit. Typically, when the training begins, the objective decreases greatly with each pass (or gradient). As the training comes to an end, the object decreases much more slowly.

The last column is the training method used on this pass (or gradient). This is usually the method you specified in the configuration panel. However, the trainer may use steepest descent from time to time to accelerate the training.

Configuring the Trainer for a Backpropagation, Autoassociative, or Ensemble Network

The configuration of the trainer depends on the type of the network being trained. To configure the trainer, it must first be connected to a specific network using an action link. The configure menu option will then bring up a configuration panel appropriate to the type of network to be trained.

This is the configuration panel you see when you are training a Backpropagation, Autoassociative, or Ensemble Network.



The image shows a dialog box titled "Trainer" with a light beige background. It contains three main sections: "Maximum iterations" with a text input field containing "100"; "Training method" with a list of four options: "conjugate gradients (Fletcher-Reeves)", "conjugate gradients (Polak-Ribiere)", "second order (BFGS)", and "second order (DFP)", each preceded by a diamond-shaped radio button; and "Accelerate training by input projection?" with two options: "yes" and "no", each preceded by a diamond-shaped radio button. At the bottom of the dialog are three buttons: "OK", "Apply", and "Cancel".

The following headings describe how to configure the block.

Choosing the Maximum Number of Training Iterations

The Trainer improves the weights in a number of individual steps. To specify the maximum number of steps, enter a number in the Maximum Iterations attribute. Typically, values range from 50 to as much as 1000. However, you will usually want to use several short training runs so you can monitor the Trainer's progress. When you evaluate the Trainer again, it continues the training from where it stopped.

Choosing the Training Method

To choose how to train the network, select one of the options below Training Method. The options fall into two main categories:

- Conjugate Gradient options: Conjugate Gradients (Fletcher-Reeves) and Conjugate Gradients (Polak-Ribiere).

- Second Order options: BFGS (Broyden-Fletcher-Goldfarb-Shanno) and DFP (Davidon-Fletcher-Powell).

In general, the Second Order options are more powerful and use significantly more memory, and the Conjugate Gradient options use less memory and take less time per step. The Conjugate Gradient options are generally better for larger networks (over 100 weights), and the Second Order options are better for smaller networks.

Once you choose which category of training methods to use, you may want to experiment with the different methods in each category to find out which is best for your network.

Note Some neural network packages allow the user to choose fixed or scheduled “learning rates” and “momentum factors.” NeurOn-Line automatically optimizes the training parameters and does not require the user to do this. The parameters that NeurOn-Line chooses will always be better than those chosen by hand.

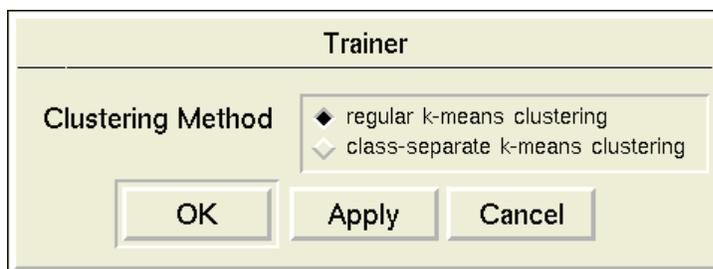
Choosing Whether to Accelerate Training

The Accelerate Training by Input Projection? option can speed up the training time. Set it to yes if your input data has columns that might be correlated.

To speed up training, the block projects your input data vector to a vector with fewer dimensions, trains with the smaller vector, and projects the smaller vector and its training results backwards to obtain results useful for the original vector. In general, this option is recommended if you have more than ten inputs.

Configuring the Trainer for a Radial Basis Function Network

This is the configuration panel you see when you are training a Radial Basis Function Network. When you have selected your option, click the Done button.



Choose an option for Clustering Method. The option you choose depends on what kind of problem you are trying to solve:

- If you are fitting the network to a function, choose regular k-means clustering.

- If you are solving a classification problem, choose **class-separate k-means clustering**.

These methods differ in how they assign locations for unit centers. When Regular clustering assigns locations, it uses all the data in the data set simultaneously. When class-separate clustering assigns locations, it goes through all the members of one class before going through the members of another. This method prevents centers from being placed near the boundaries between classes, where they do not help to discriminate between the classes.

During training, NOL finds and outputs to the background window the cluster centers as shown in this matrix. Each row represents the coordinates of a cluster center. Up to five centers are shown. Each column represents the coordinates for a dimension.

Matrix In cluster, now m = size = 14 by 10

	1	2	3	4	5	6	7	8
1	0.565	0.738	0.374	0.38	0.415	0.257	0.372	0.345
2	0.536	0.532	0.536	0.83	0.509	0.482	0.275	0.356
3	0.603	0.687	0.328	0.243	0.394	0.287	0.604	
4	0.573	0.446	0.337	0.363	0.749	0.656	0.437	0.777
5	0.675	0.774	0.391	0.328	0.3	0.419	0.685	0.399

If the Unit Overlap attribute is automatic, NOL then determines the optimum unit widths by searching for the value of p that minimizes the training error, with the unit centers previously determined by k-means clustering during the first stage of the training, as shown in the following output:

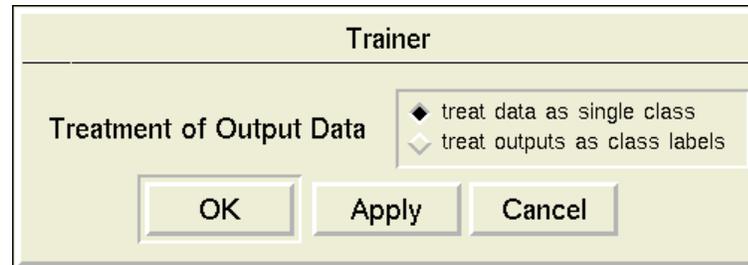
```
Optimizing overlap parameter
Trying p = 1.000000, objective = 19.973553
Trying p = 2.000000, objective = 18.852538
Trying p = 3.618034, objective = 17.939625
Trying p = 4.135448, objective = 17.873745
Trying p = 4.972642, objective = 18.574853
Trying p = 4.455228, objective = 17.979710
Trying p = 3.937814, objective = 17.871113
Trying p = 4.025800, objective = 17.867350
Trying p = 3.985542, objective = 17.868146
Trying p = 4.066058, objective = 17.868195
```

Optimized overlap with $p = 4.025800$

Final objective function value = 17.665130

Configuring the Trainer for a Rho Network

This is the configuration panel you see when you are training a Rho Network. When you have selected your option, click the OK button

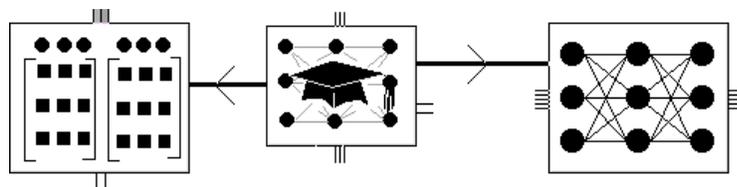


Which option you choose depends on what kind of problem you are trying to solve:

- If the data set contains data that belongs to a single class, choose **treat data as single class**. The Trainer ignores any output values in the data set. When you later evaluate the Rho Network, its output will be the probability that the input data is part of the distribution defined by the training set.
- If the data set contains data that belongs to several classes, choose **treat output data as class label**. The number of output values in the data set corresponds to the number of possible classes. In each data pair, the output that corresponds to the element's class should be one, and the rest of the outputs should be zero. When you later evaluate the Rho Network, its output will be a vector with one element for each class, and each element will be the probability that the input belongs to that class.

Example

Below is a simple diagram for training a network. To train this network, configure the network architecture and the trainer, then choose **evaluate** from the Training block's menu.

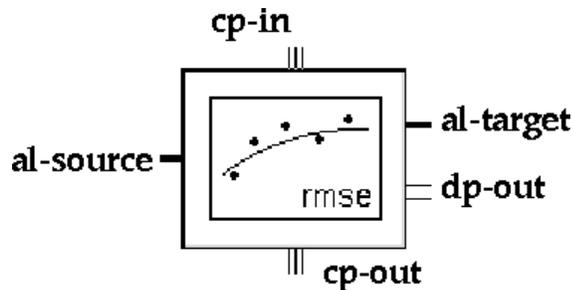


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Neural Network Blocks		<i>Reference Manual</i>
Data Set		<i>Reference Manual</i>
Fit Tester		<i>Reference Manual</i>
Train and Test		<i>Reference Manual</i>
Five Fold CV		<i>Reference Manual</i>

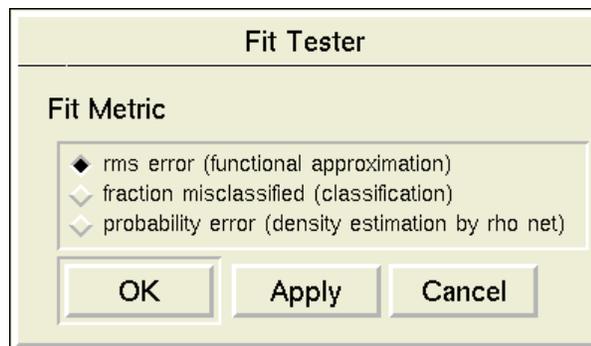
Fit Tester



The Fit Tester block can evaluate the performance of any neural network in NeurOn-Line. Attach one of its action links to a neural network that has been trained, and attach the other action link to a data set for the neural network (not necessarily the data set used for training the network). You cannot attach more than one neural network or data set to the block.

Configuring

To configure the Fit Tester, choose **configure** from its menu. It displays the configuration panel below.



Specify the Fit Metric. Choose an option depending on the type of network you are using and the type of problem you are solving:

- If you are fitting a function, choose **rms (functional approximation)**.
- If you are solving a classification problem, choose **fraction misclassified (classified)**.
- If you are estimating a probability density function, choose **probability error (density estimation by rho net)**.

To evaluate the Fit Tester, either pass it a control signal or choose **evaluate** from its menu. The block applies the neural network to the data set's input data, and

stores the result in the data set as the prediction data. It then compares the data set's prediction data and target data.

When the Fit Tester finishes evaluating, it passes a control signal and a scalar value. The scalar value tells you how well the prediction data matches the target data. The Fit Tester computes that number differently depending on the option you chose in the configuration panel:

- If you chose **rms error**, it passes a positive number. The closer the number is to 0, the better the network fits the function. To compute the number, the Fit Tester subtracts the target value from the predicted value to get the error, squares that error, and takes the square root of the mean of the errors over the entire training set.
- If you chose **fraction misclassified**, it passes a number from 0 to 1, where 0 means all samples were classified correctly, and 1 means no samples were classified correctly. To compute the number, the Fit Tester figures the ratio of misclassified samples to total samples.
- If you chose **probability error**, the Fit Tester figures the negative mean of the logarithm of the probability predictions of the Rho Network. The lower the number, the more accurately the Rho Network matches the probability distribution implied by the examples in the data set.

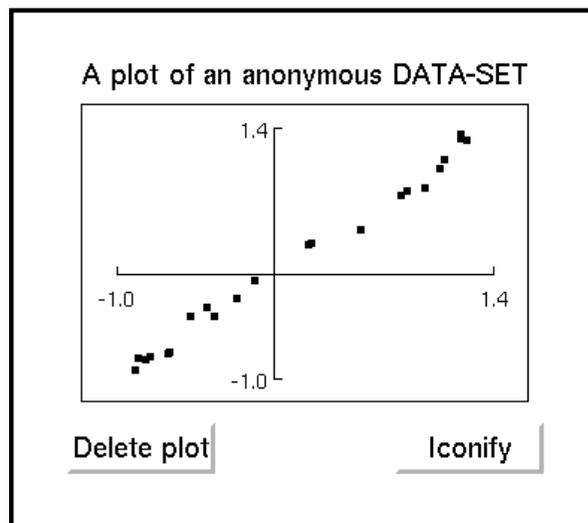
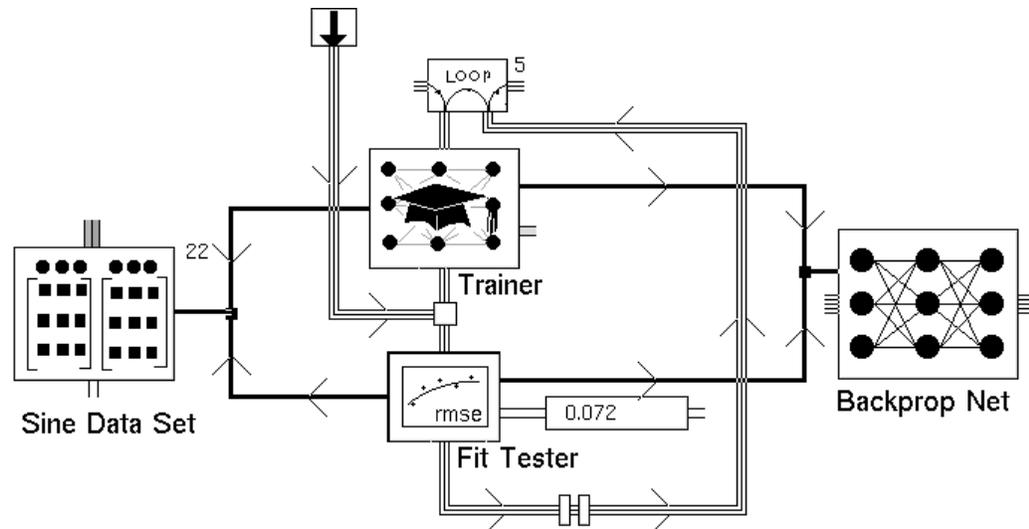
In addition to the calculation of the scalar error, the function of the Fit Tester is to fill up the predictions matrix in the data set; otherwise, the predictions are empty.

To view the predictions stored in the data set, use the data set's configuration panel. For more information, see [Data Set](#).

Example

In this example, a Fit Tester tests a Backpropagation network after it has been trained. The network is trained and tested a total of five times. A Path Display shows you the error number to let you know how well the network fits the data it is training on.

The figure also shows the plot of predicted values against target values. Notice that they lie on a 45 degree line, which means the predicted and target values are equal, i.e., they are a good fit.



See Also

For general information on how to use this block, see the sections below.

For more information on...

See...

In this book...

Basic Block Behavior

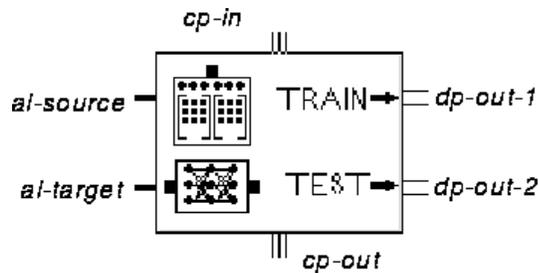
User's Guide

Neural Network Blocks

*Reference
Manual*

For more information on...	See...	In this book...
Data Set		<i>Reference Manual</i>
Trainer		<i>Reference Manual</i>
Train and Test		<i>Reference Manual</i>
Five Fold CV		<i>Reference Manual</i>

Train and Test



The Train and Test block helps you determine whether you have chosen the best neural network configuration for your problem. You can use it to test a data set against different types of neural networks or different configurations of the same type of neural network. It trains and tests your network a specified number of times, each time randomly splitting the data set into two subsets: training and testing.

Note The Train and Test block is an encapsulation block that contains a NeurOn- Line diagram on its subworkspace. For more information on what the subworkspace contains, see [The Train and Test Block's Subworkspace](#).

Attach the Train and Test block's top action link to your data set and the bottom action link to your neural network. You cannot attach more than one neural network or data set to the block.

To configure the Train and Test block, choose **configure** from its menu. It displays three dialogs, one on top of the other. The topmost is for the Train and Test Block. It is described in [Configuring](#). The other dialogs are for the Training and Fit Tester blocks that are on the Train and Test block's subworkspace. For more information on how to configure them, see [Trainer](#) and [Fit Tester](#).

To evaluate the Train and Test block, you must pass it a control signal or choose **evaluate** from the menu. The block trains and tests your neural network a specified number of times. Each time it randomly splits the attached data set into two different subsets using a proportion you specify: one subset for training and one for testing.

To cancel training, choose **Controls > Remote Process > Kill** and then reset the block. The neural network's weights and other internal parameters might be changed if the Train and Test completed one or more iterations before it cancels training. The Train and Test block outputs nothing. If you cancel training, you must restart the remote process.

When the block has finished all the iterations, it passes a control signal and two scalar values. The scalar value from its top output port is the median value of the results from the Trainer block. The scalar value from its bottom port is the median

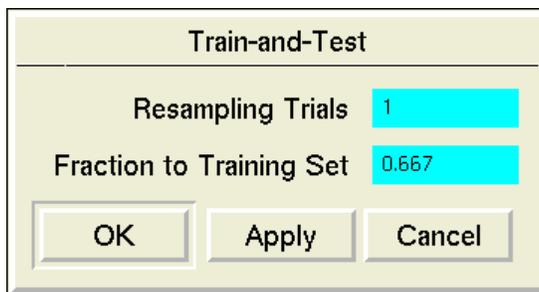
value of the results from the Fit Tester block. The meaning of these values depends on how you configure the Trainer and Fit Tester blocks. For more information on these values, see [Trainer](#) and [Fit Tester](#).

In each case, the training error is an indicator of how well the network fits the training data. The error will generally decrease if you expand the network architecture, even when the network is overfitting. Therefore, do not use the training error to select the optimum network architecture.

The test error indicates how well the network fits data not used in training. In general, you should select the smallest network architecture that minimizes the test error. This value will increase or stay nearly constant when the network is too large.

Configuring

To configure the Train and Test block, choose **configure** from its menu. It displays three dialogs, one on top of the other. The topmost dialog is shown below.



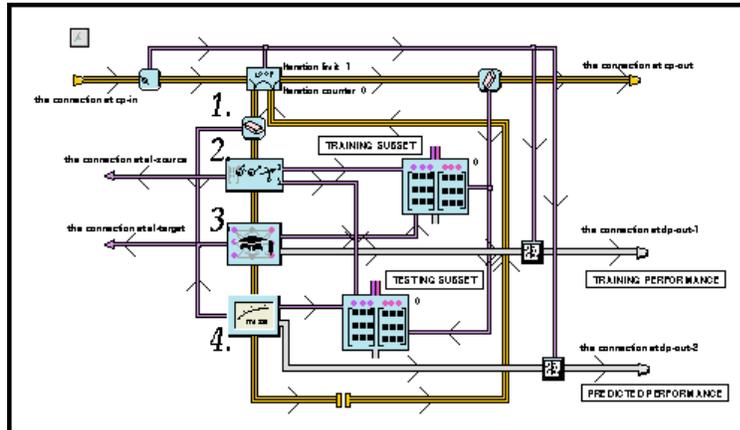
To specify the number of times to train and test your network, enter a number in the attribute Resampling Trials. To specify how much of the data set to use for training and how much for testing, enter the proportion for training in the attribute Fraction to Training Subset. The block uses the rest of the data for testing.

You must also configure the dialog for the Trainer block on the subworkspace of the Train and Test block. For more information on configuring this block, see [Configuring the Trainer for a Backpropagation, Autoassociative, or Ensemble Network](#).

Finally, you must configure the dialog for the Fit Tester on the subworkspace of the Train and Test block. For more information on configuring this block, see [Configuring](#):

The Train and Test Block's Subworkspace

The Train and Test Block has a subworkspace with the following diagram, which you can access by using the view diagram menu choice.



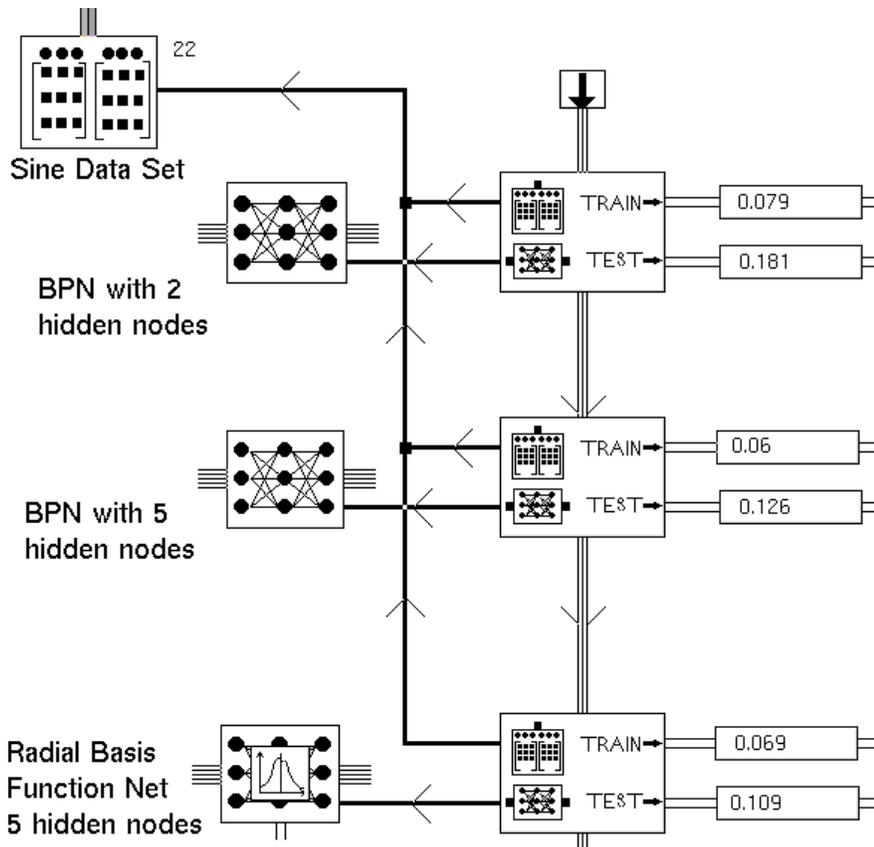
For the number of times that you specified in the configuration panel, the block follows this procedure:

- 1 Clear the neural network to prepare it for a new training session.
- 2 Randomly separate the data set into two smaller data sets: one for training and one for testing. The training set contains the fraction of the data set that you specified in the configuration panel. The testing set contains the rest.
- 3 Train the network with the training set.
- 4 Test the network with the testing set.

Note Do not configure blocks on the subworkspace directly, because this can cause inconsistencies. Use the configuration panels instead.

Example

This example uses Train and Test blocks to see which of three neural network configurations is the best. All networks use exactly the same data set. To determine which configuration fits the data best, look at the number passed from the test output of the Train and Test block. The lower the number, the better the fit for data not seen in the training process.



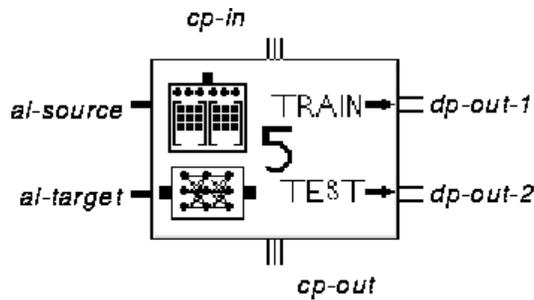
See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Neural Network Blocks		<i>Reference Manual</i>
Data Set		<i>Reference Manual</i>

For more information on...	See...	In this book...
Trainer		<i>Reference Manual</i>
Fit Tester		<i>Reference Manual</i>
Five Fold CV		<i>Reference Manual</i>

Five Fold CV



The Five Fold CV block helps you determine whether you have chosen the best neural network configuration for your problem. You can use it to test a data set against different types of neural networks or different configurations of the same type of neural network. It splits the data set into five subsets and then trains and tests your network a total of five times, each time using a different one of the five subsets for training.

Note The Train and Test block is an encapsulation block that contains a NeurOn-Line diagram on its subworkspace.

Attach the Five Fold CV block's top action link to your data set and the bottom action link to your neural network. You cannot attach more than one neural network or data set to the block.

To configure the Five Fold CV block, choose **configure** from its menu. It displays two dialogs, one on top of the other. They are for the Training and Fit Tester blocks that are on the Five Fold CV block's subworkspace. For more information on how to configure them, see [Trainer](#) and [Fit Tester](#).

To evaluate the Five Fold CV block, you must pass it a control signal. The block randomly divides the data set into five subsets of equal size. It trains and tests your neural network five times. The first time, it trains the network with the first subset and tests it with the other four. The second time, it trains the network with the second subset and tests it with the other four. The block continues until its trained and tested the block five times, each time using a different fifth of the data set for training.

When the block has finished all the iterations, it passes a control signal and two scalar values. The scalar value from its top output port is the median value of the results from the Trainer block. The scalar value from its bottom port is the median value of the results from the Fit Tester block. The meaning of these values depends on how you configured the Trainer and Fit Tester blocks. For more information on these values, see [Trainer](#) and [Fit Tester](#).

In each case, the training error is an indicator of how well the network fits the training data. The error will generally decrease if you expand the network architecture, even when the network is overfitting. Therefore, do not use the training error to select the optimum network architecture.

The test error indicates how well the network fits data not used in training. In general, you should select the smallest network architecture that minimizes the test error. This value will increase or stay nearly constant when the network is too large.

Configuring

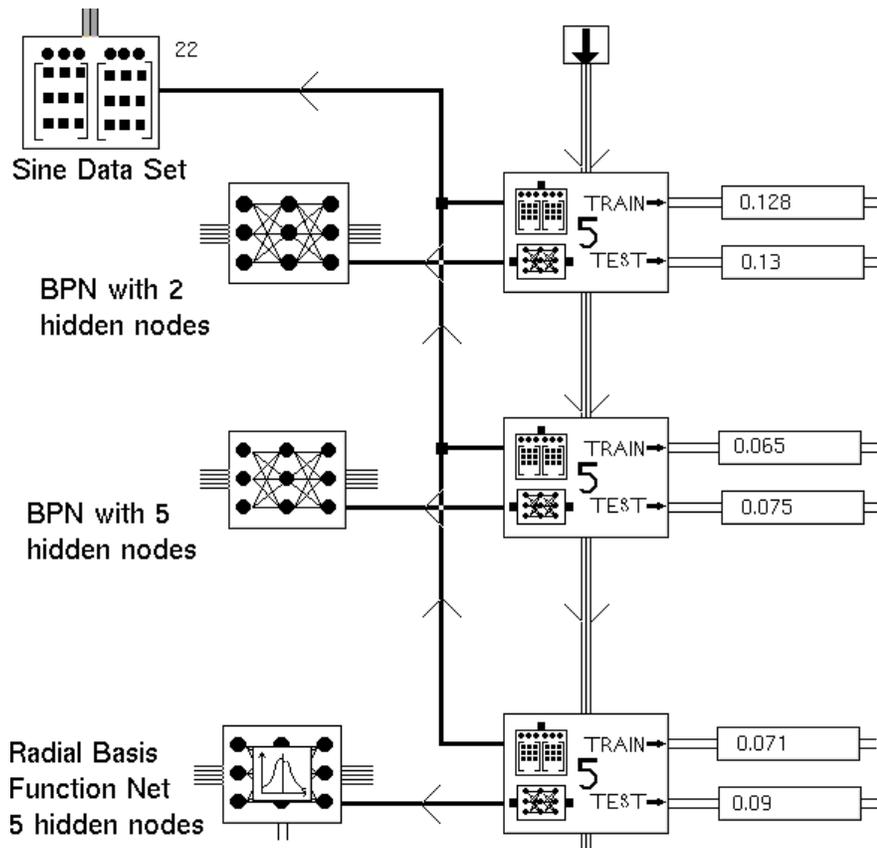
When you choose **configure** from the block's menu, it displays two dialogs, one for configuring the Training block and the other for configuring the Fit Tester block, both of which are on the Five Fold CV block's subworkspace.

For information on configuring the Trainer block, see [Configuring the Trainer for a Backpropagation, Autoassociative, or Ensemble Network](#).

For information on configuring the Fit Tester block, see [Configuring](#).

Example

This example uses Five Fold CV blocks to see which of three neural network configurations is the best. All networks use exactly the same data set. To determine which configuration fits the data best, look at the number passed from the bottom port of the Five Fold CV block. The lower the number, the better the fit for data not seen in the training process.

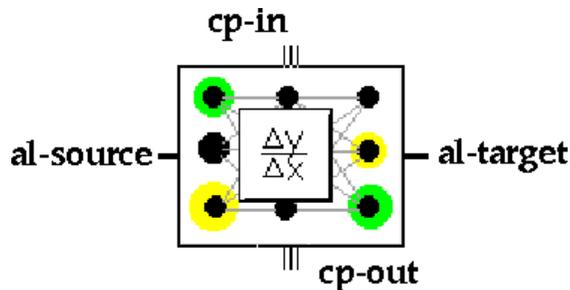


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Neural Network Blocks		<i>Reference Manual</i>
Data Set		<i>Reference Manual</i>
Trainer		<i>Reference Manual</i>
Fit Tester		<i>Reference Manual</i>
Train and Test		<i>Reference Manual</i>

Sensitivity Tester



The Sensitivity Tester helps you determine which inputs of a neural network you actually need to predict the output. When you eliminate unnecessary inputs, you reduce the possibility of a prediction error. Attach one of its action links to a neural network and the other action link to a data set. You cannot attach more than one neural network or data set to the block.

You apply this block after a network is trained, and it gives you an idea about how to improve the fit by finding low-influence inputs.

To evaluate the Sensitivity Tester, either pass it a control signal or choose **evaluate** from its menu. The Sensitivity Trainer performs tests using the data set to determine which inputs most strongly affect the predictions. It creates a matrix that tells you how each input affects each output. When it is done, the Sensitivity Tester passes a control signal.

To view the results, select **view results** from the block's menu. It displays a spreadsheet, described in [Using the GXL Spreadsheet to Edit Data](#) in the *NeurOn-Line User's Guide*. Each column represents the importance of the inputs to one output. The inputs are rated on a scale from 0.0 to 1.0, where 0.0 means that this input has no affect on this output, and 1.0 means that this input is the only input that affects that output. The scores for each column add up to 1.0.

If an entire row contains relatively small values, that input probably does not affect the network's predictions and can be removed from the input set.

Making Values Permanent

When you choose **make permanent** from the block's menu, the block saves the matrix of inputs and outputs.

Configuring

This block has no configuration panel.

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>
Neural Network Blocks		<i>Reference Manual</i>
Data Set		<i>Reference Manual</i>

Action and Other

Chapter 9: Action Utilities

Describes the blocks that perform actions, such as creating loops, evaluating conditions, performing generic actions, and executing rules.

Chapter 10: Inference Blocks

Describes the blocks that create and manipulate truth values.

Chapter 11: Capabilities

Describes the blocks that add features and attributes to other NeurOn-Line blocks, such as charts, graphs, and clocks.

Action Utilities

Describes the blocks that perform actions, such as creating loops, evaluating conditions, performing generic actions, and executing rules.

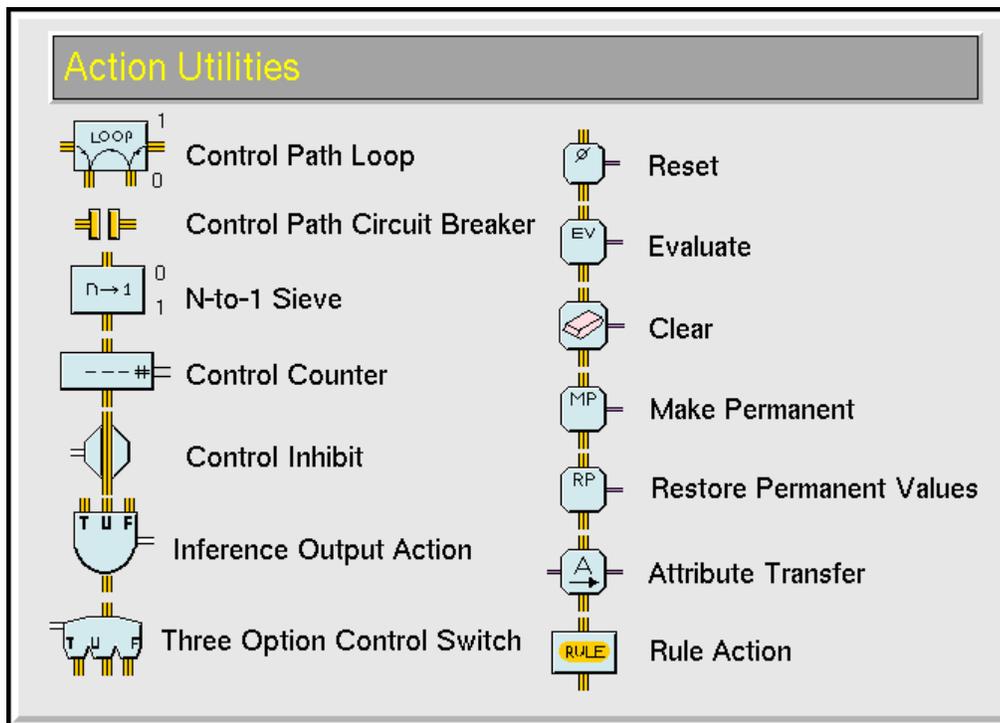
Introduction	270
Control Path Loop	273
Control Path Circuit Breaker	275
N-to-1 Sieve	276
Control Counter	278
Control Inhibit	280
Inference Output	282
Control Switch	285
Reset	286
Evaluate	288
Clear	289
Make Permanent	291
Restore Permanent Values	293
Attribute Transfer	294
Rule Action	296



Introduction

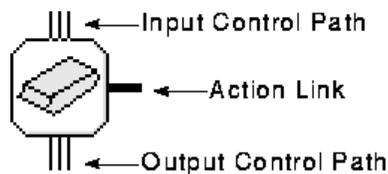
NeurOn-Line provides you with Action blocks, which let you create loops, evaluate conditions, perform actions on other blocks, and execute rules.

You can find the Action Utilities under the Action & Other submenu of the Palettes menu:



You connect Action blocks together with Control Paths, which carry control signals. A block receives a control signal, performs its action, passes the control signal to the next block, and so on. A control signal does not have a value, but it makes sure that an action block is evaluated only after the previous one is done.

Unlike other blocks, an Action block's paths are on the top and bottom: the input path is on top and the output path is on the bottom, as in the figure below.



Some Action blocks also have an Action Link, which you connect to another NeurOn-Line block. It is shown in the figure above. When the Action block is evaluated, it performs its action on any block connected to its Action Link. For

example, the Block Erase erases the data stored in the block connected to its Action Link.

Looping

These blocks let you create loops:

- The [Control Path Loop](#) block lets you execute a loop for a specified number of times or until a specified condition is met.
- The [Control Path Circuit Breaker](#) block lets NeurOn-Line know that a loop in your diagram is intentional. It is fully described in [Circuit Breakers](#).

Stopping Paths

These blocks let you stop the flow of data:

- The [N-to-1 Sieve](#) block discards a specified number of signals before passing one.
- The [Control Inhibit](#) block discards an incoming control signal if an inference input has a given value.

Outputting Data

These blocks let you pass an inference value or a count of control signals:

- The [Control Counter](#) block passes on a data path the number of times it received a control signal. It also passes the control signal on an output control path.
- The [Inference Output](#) block chooses which inference value to pass depending on which of its three input control paths it receives a signal on.

Branching

The [Control Switch](#) block lets an inference path choose on which path to send a control signal.

Performing Actions on Blocks

These blocks perform actions on other blocks. Most of these actions are similar to the menu choices available in a block's menu. Notice that there can be some small but important differences in how these blocks and the menu choices operate.

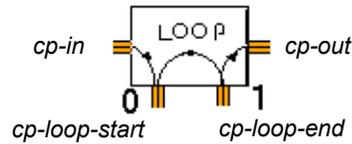
- The [Reset](#) block resets a block.
- The [Evaluate](#) block forces a block to evaluate.
- The [Clear](#) block erases the data a block has stored.

- The [Make Permanent](#) block makes permanent the data that a block has stored.
- The [Restore Permanent Values](#) block restores to a block the data that you have previously made permanent.
- The [Attribute Transfer](#) block transfers attribute values from one block to another.

Invoking a Rule

The [Rule Action](#) block can invoke a G2 rule you have written when it receives a control signal.

Control Path Loop



The Control Path Loop block allows you to add control loops to a diagram. This block iterates from 0 to the value specified by the Iteration Limit attribute, or until the text expression in the Exit If attribute turns `.true`.

For more information on how to specify text expressions, see [Evaluating Expressions in Attributes](#) in the *NeurOn-Line User's Guide*.

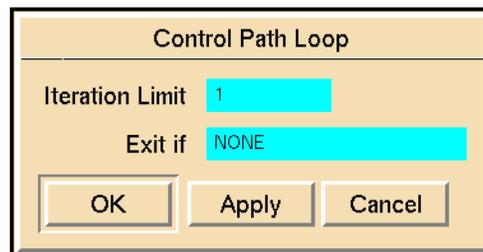
Diagrams that include the Control Path Loop block must contain a circuit breaker somewhere in the cycle.

Resetting

When you reset the Control Path Loop, the block sets its counter back to zero.

Configuring

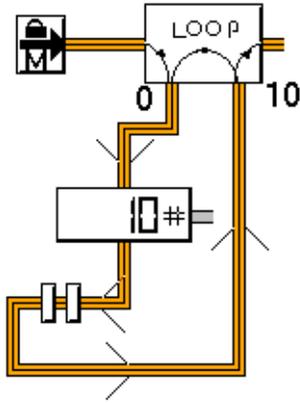
This is the configuration panel for the Control Path Loop block.



Attribute	Description
Iteration Limit	The number of times that the block executes.
Exit If	An expression that when <code>.true</code> , causes the block to stop executing. For information on specifying the expression, see Evaluating Expressions in Attributes in the <i>NeurOn-Line User's Guide</i> .

Example

The following example illustrates how you use this block to create a loop that counts from 0 to 10, inclusive:



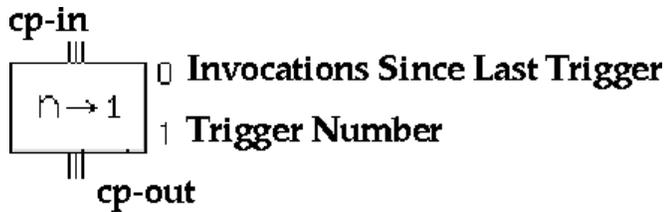
For more information on...	See...	In this book...
Resetting Blocks		<i>User's Guide</i>
Evaluating Blocks		<i>User's Guide</i>
Reading Notes and Errors		<i>User's Guide</i>

Control Path Circuit Breaker



The Control Path Circuit Breaker is also on the Connections palette in the Entry & Paths menu, along with all the other Circuit Breakers. For more information, see [Circuit Breakers](#).

N-to-1 Sieve



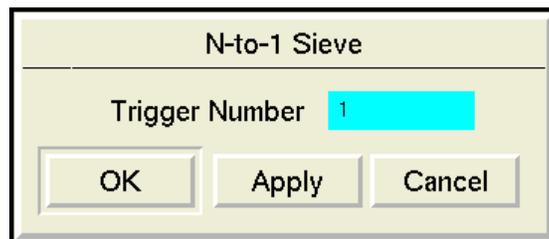
The N-to-1 Sieve block reduces the number of control signals that flow in a path by selectively passing and discarding signals. For every Trigger Number signals that the block receives, it passes one signal. After it passes a signal, the block resets its counter and starts again. For example, if Trigger Number is 5, the block discards the first four signals it receives and passes the fifth signal. Then the block resets its counter, ignores the next four signals, and passes the tenth.

Resetting

When you reset the N-to-1 Sieve block, it resets its counter does not pass a signal until it receives Trigger Number more signals.

Configuring

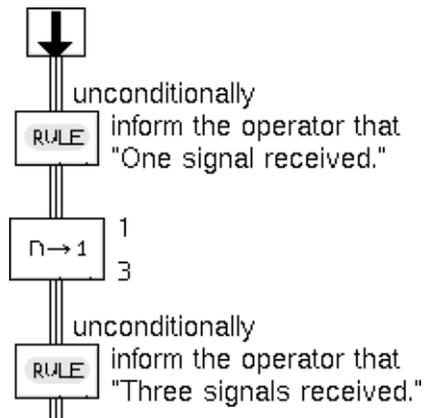
This is the configuration panel for N-to-1 Sieve.



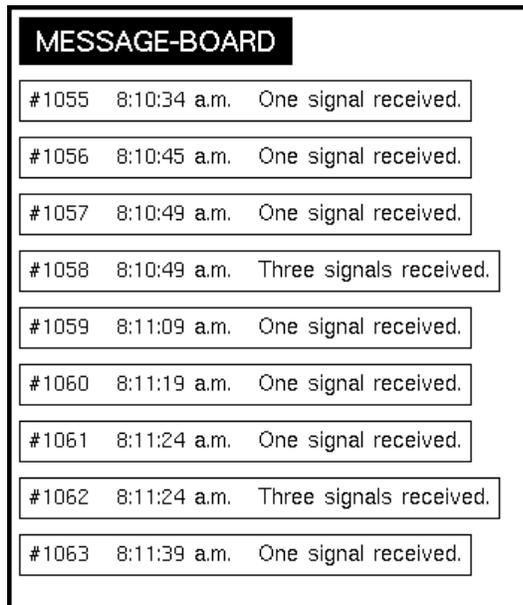
Attributes	Description
Trigger Number	How many signals the block must receive before it passes a signal.

Example

This example contains an N-to-1 Sieve that discards 2 out of every 3 signals received. In the figure below, the Trigger Count is 3 and the block has received one signal since it last passed a signal.



This figure shows what the Message Board contains after the N-to-1 Sieve has received 7 signals.



See Also

For general information on how to use this block, see the sections below.

For more information on...

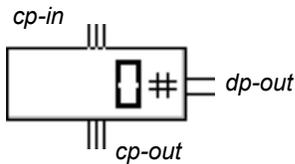
See...

In this book...

Basic Block Behavior

User's Guide

Control Counter



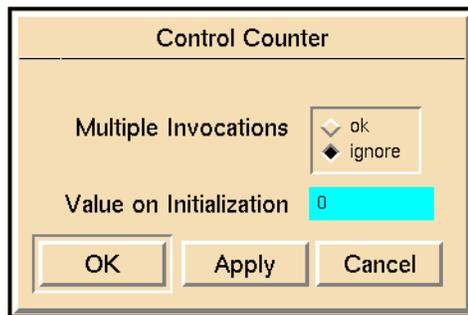
The Control Counter counts the number of times it receives a control signal and displays the count on its icon. It passes the control signal on its output control path and the count on its output data path.

Resetting

When you reset this block, it resets its counter to the value of the attribute Value on Initialization and passes that value on its data path.

Configuring

This is the configuration panel for the Control Counter.

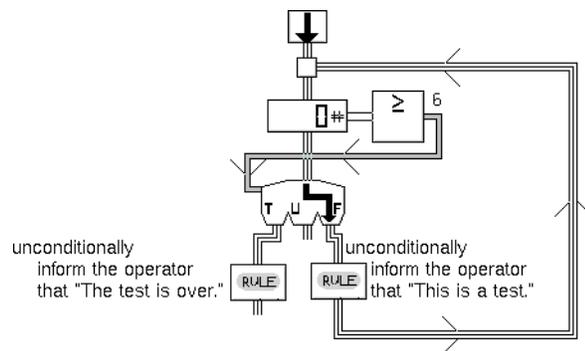
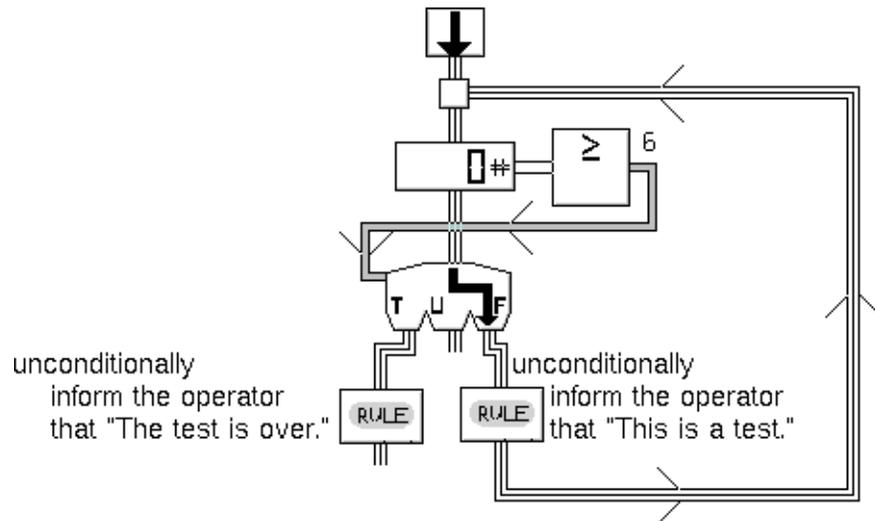


Attribute	Description
Multiple Invocations	See Specifying How to Handle Multiple Values in the <i>NeurOn-Line User's Guide</i> .
Value on Initialization	See Specifying an Initial Data Value in the <i>NeurOn-Line User's Guide</i> .

Example

The following diagram writes “This is a test,” to the Message Board five times and finally writes “The test is over.” The Control Counter counts the number of times “This is a test” has been written. The High Value block then checks whether it has

been written ten times. If the High Value block passes `.false`, the Control Switch passes control to the Rule Action block, which writes “This is a test” and passes control back to the Control Counter. If the High Value block passes `.true`, the Control Switch passes control to the Rule Action block, which displays “The test is over.”



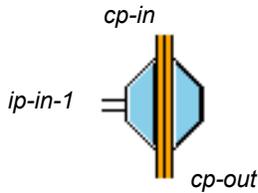
To start this diagram, choose `evaluate` from the Control Counter’s menu.

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User’s Guide</i>
Specifying an Initial Data Value		<i>User’s Guide</i>

Control Inhibit



The Control Inhibit block lets an inference path turn a control path on and off. You can use the block to turn on and off entire sections of a flow diagram.

When the status value of the block's input inference path matches the value of the attribute *Trigger On*, the block discards the input control signal. When the status value of the inference path no longer matches *Trigger On*, the block passes the current value of the input control path and continues to pass the input control signal normally.

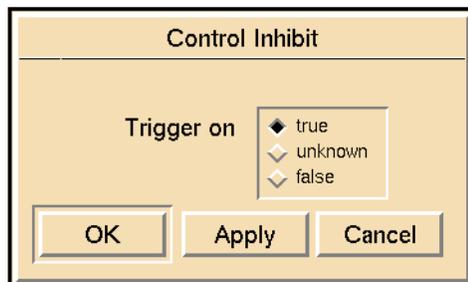
If the block's inference path hasn't received a value yet (that is, it has a **quality of no-value**), the block passes nothing even when it receives a value from its control path.

Resetting

When you reset a Data Inhibit Block, the block does not pass a signal until it receives a value from its inference input path, even if it receives a signal from its input control path.

Configuring

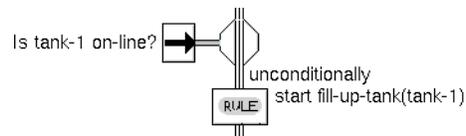
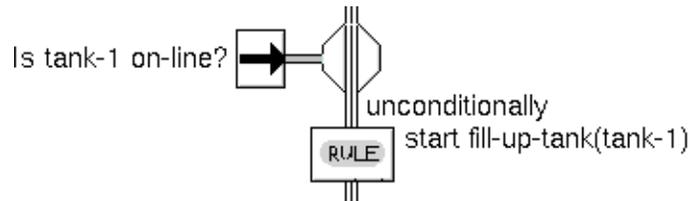
This is the configuration panel for the Control Inhibit block.



Attribute	Description
Trigger On	The truth value that causes the block to discard its input control signal.

Example

The following diagram fills up tank 1 only if the tank is on-line. If the Control Inhibit block receives a control signal while the tank is off-line, the block discards the signal. It must receive a signal while the tank is on-line before it fills up the tank.

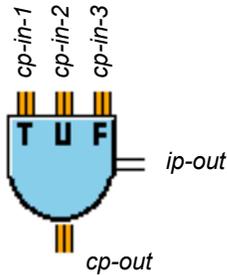


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Specifying How to Handle Multiple Values		<i>User's Guide</i>

Inference Output



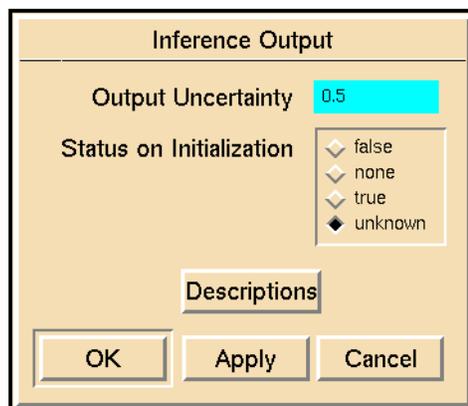
The Inference Output block chooses an inference value to pass depending on which input control path it receives a signal from. Whenever it receives a control signal on one of its input control paths, it passes the inference value for that path on its output inference path and passes the control signal on its output control path.

This table lists the inference values associated with each input control path:

If the block receives a control signal on the path marked...	It passes this inference value...
"T" (cp-in-1)	.true
"U" (cp-in-2)	unknown
"F" (cp-in-3)	.false

Configuring

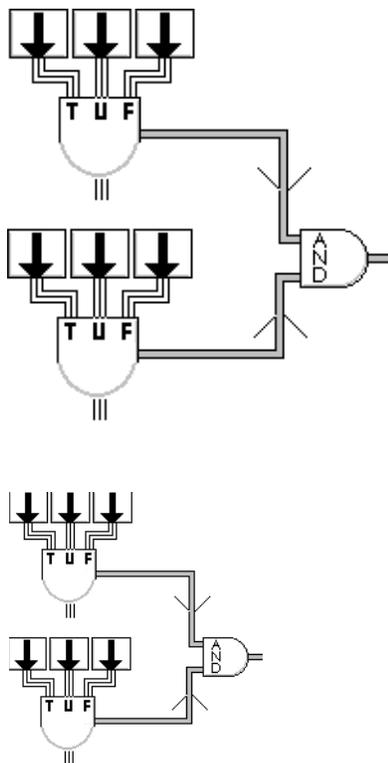
This is the configuration panel for the Inference Output block.



Attribute	Description
Output Uncertainty	This attribute is used with fuzzy logic, which NeurOn-Line does not support.
Status on Initialization	See Specifying an Initial Status Value in the <i>NeurOn-Line User's Guide</i> .
Description when True, Description when False, Description when Unknown, and Description of Input	These attributes allow you to display descriptions, which NeurOn-Line does not support.

Example

The diagram in the following figure passes .true if the two Control Entry Points on the left pass a control signal.

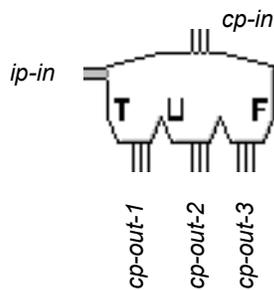


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Specifying an Initial Status Value		<i>User's Guide</i>

Control Switch



The Control Switch lets an inference value choose which path to send a control signal down. It is useful when you need to perform a sequence of control blocks until some condition is met.

The Control Switch block only passes values when it receives a new control signal; it does not pass values when the inference value changes.

The value of the side inference path (*ip-in*) determines where the control signal (*cp-in*) will go. This table shows on which output path the signal will go:

If the side input is...	The left input is passed on the port labeled...
.true	"T" (cp-out-1)
unknown	"U" (cp-out-2)
.false	"F" (cp-out-3)

Configuring

The Control Switch has no configurable attributes.

Example

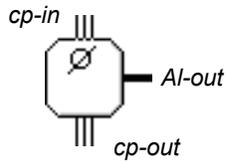
For an example, see the example for [Control Counter](#).

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

Reset



The Reset block resets the blocks connected to its action link. It performs the same action as choosing reset from the blocks' menus.

Note Reset block does not clear the weights in a Neural Network or the Data Pairs in a Data Set. To clear data in those blocks, see [Clear](#).

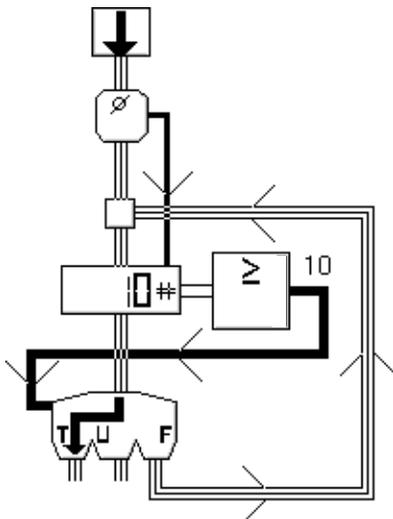
If more than one block is connected to the Reset Blocks's action link, it resets all the blocks.

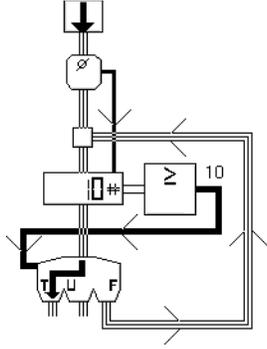
Configuring

The Reset block has no configurable attributes.

Example

In the example below, the Reset block resets a Control Counter before the counter counts up to 10.



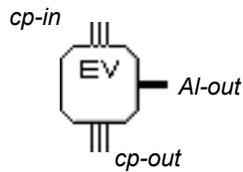


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

Evaluate



The Evaluate block evaluates the blocks connected to its action link. It performs the same action as choosing evaluate from the blocks' menus.

If a connected block has a Clock Capability which has the attribute Allow Intermediate Evaluation set to no, Evaluate block does not evaluate the block. If more than one block is connected to Evaluate Blocks's action link, it evaluates all the blocks.

Configuring

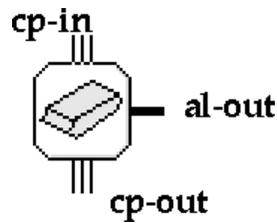
The Evaluate block has no configurable attributes.

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

Clear



The Clear block clears the data of the blocks that are connected to its action link. Clear works with these blocks only.

If it is connected to a...	Clear does this...
Back Propagation Network	Randomizes the network's weights
Autoassociative Network	Randomizes the network's weights
Data Sets	Clears the data in the data set
Data Pair Buffers	Clears the contents of the buffer

If more than one block is connected to the action link, Clear clears all of them.

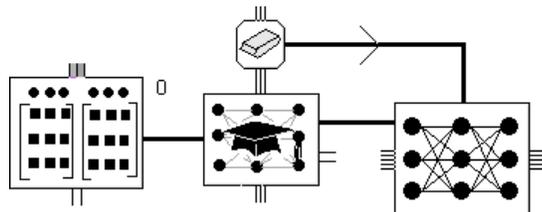
Note Clear does not reset other blocks to their original states. To do that, see Reset block.

Configuring

This block has no configuration panel.

Example

In the figure below, Clear clears a Backpropagation Network before it is trained.

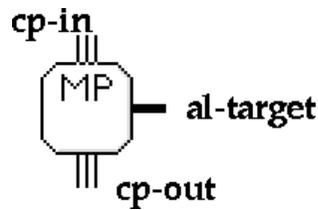


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

Make Permanent



The Make Permanent block makes permanent the data stored in the blocks connected to its action link. It performs the same action as choosing make permanent from the connected blocks' item menus. You can connect it to any block that has restore permanent or make permanent in its menu, including all Neural Networks and Data Sets.

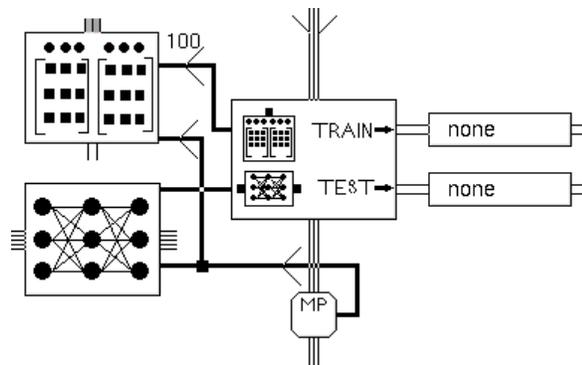
If more than one block is connected to Make Permanent's action link, it makes permanent the data for all the blocks.

Configuring

This block has no configuration panel.

Example

In the example below, a Make Permanent block makes the data in a Neural Network and Data Set permanent after the Neural Network has been trained and tested.

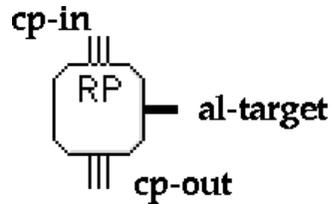


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>

Restore Permanent Values



The Restore Permanent Values block restores the permanent values of the block to which it is connected. It performs the same action as choosing restore permanent values from the connected block's item menu. You can connect it to any block that has restore permanent or make permanent in its menu, including all Neural Networks and Data Sets.

If more than one block is connected to Restore Permanent Values's action link, it restores the data for all the blocks.

Configuring

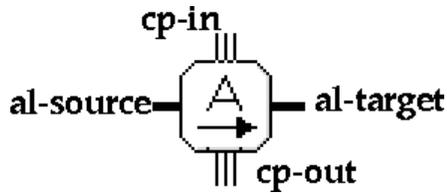
This block has no configuration panel.

See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Saving a Block's Data After Resetting G2		<i>User's Guide</i>

Attribute Transfer



The Attribute Transfer block copies one block's attribute values to another block. It copies the values of user-defined attributes, the internal architecture of a Neural Network, the weights in a Neural Network, and the Data Pairs in a Data Set. It does not copy the values of system-defined attributes, such as names, notes, and user-restrictions. It can copy attributes between blocks if the blocks are the same type or if the block receiving the values is a subclass of the block giving the values.

The table below lists the blocks that can receive values from other types of blocks.

You can copy from this block...	To this block...
Backpropagation Network	Autoassociative Network
Radial Basis Function Network	Rho Network

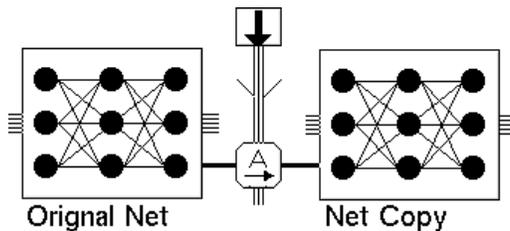
Attach the block to give the values at the left action link, and attach the block to receive the values at the right action link.

Configuring

This block has no configuration panel.

Example

In the example below, the Block Attribute Transfer block copies the user attributes of the Neural Net labelled Original Net to the Neural Net labelled Net Copy.

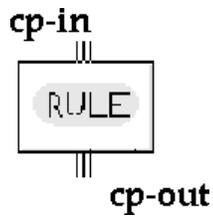


See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

Rule Action



The Rule Action block lets you evaluate one or more rules in your NeurOn-Line diagram. Place the rules on the subworkspace of the block. When you first clone Rule Action Block, it has an empty subworkspace.

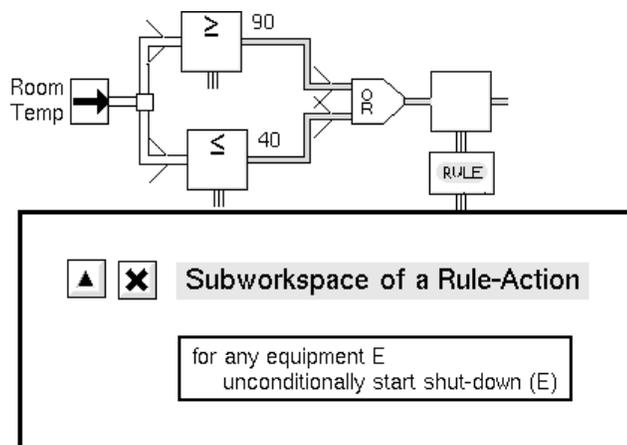
When a Rule Action block evaluates, the rules on its subworkspace are executed in order of priority, from 0 to 10. Rules with the same priority execute in arbitrary order. Like rules in any G2 knowledge base, these rules can forward and backward chain to any other rules in your knowledge base.

Configuring

This block has no configuration panel.

Example

The example below uses a Rule Action block to shutdown all your equipment if the room temperature goes beyond a safe range (40 to 90 degrees Fahrenheit).



See Also

For general information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Arithmetic Function		<i>Reference Manual</i>
Arithmetic Function of Two Arguments		<i>Reference Manual</i>
Vector Function		<i>Reference Manual</i>
Vector Function of Two Arguments		<i>Reference Manual</i>

Click here for more information...

[Basic Block Behavior](#)

[Arithmetic Function](#)

[Arithmetic Function of Two Arguments](#)

[Vector Function](#)

[Vector Function of Two Arguments](#)

Inference Blocks

Describes the blocks that create and manipulate truth values.

Introduction **299**

High Value Observation, Low Value Observation **302**

Equality Observation **305**

Conclusion **308**

AND Gate **311**

OR Gate **314**

NOT Gate **317**

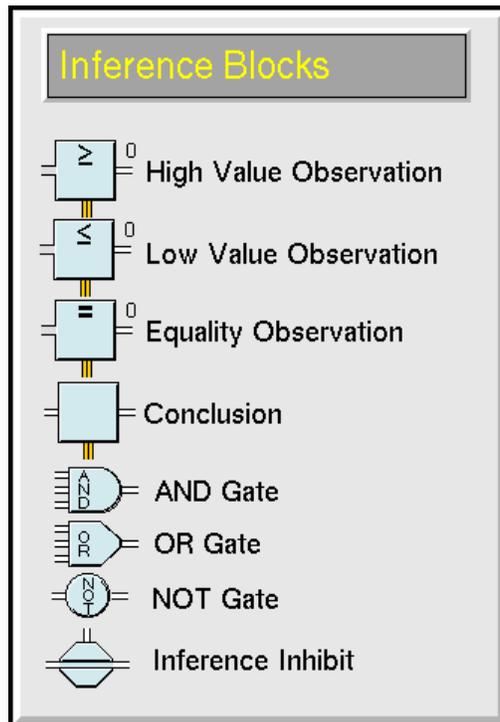
Inference Inhibit **319**



Introduction

NeurOn-Line lets you create and manipulate truth values, which NeurOn-Line calls Inference Values.

You can find the Inference Blocks palette under the Action & Other submenu of the Palettes menu:



Observations

The first three blocks on this palette [High Value Observation](#), [Low Value Observation](#), [Equality Observation](#), and [Conclusion](#) detect features in (or draw "observations" from) your data. These blocks take a data value as input, test it, and pass the inference value that the test produced as output. Since you can state the test as an observation, such as "the input temperature is greater than or equal to 100°," these blocks are called Observations. They mark the transition from the data stage to the inference stage of your diagram.

The Observation blocks and the Conclusion block have special properties that the rest of the blocks on this palette do not have:

- **Triggering action blocks.** Every condition has an output Control Path on the bottom of its icon. The block passes a control signal down that path whenever it passes a .true status value.
- **Triggering capabilities.** You can attach any capability to a Condition, including those dealing with alarms.
- **Overriding.** Every condition has an override menu choice that lets you manually enter the value it passes on.

- Locking. Every condition also has a lock menu choice, described in [Locking and Unlocking Blocks](#), that lets you lock in the value. The output value won't change until you choose the unlock menu choice.

The first three blocks test values against a threshold:

- The High Value Observation block tests whether the input value is higher than a threshold.
- The Low Value Observation block tests whether the input value is lower than a threshold.
- The Equality block tests whether the input value is equal to a reference value.
- The Conclusion block is the simplest of these blocks. It serves no function other than providing you with an Observation block's special properties.

Performing Logical Operations

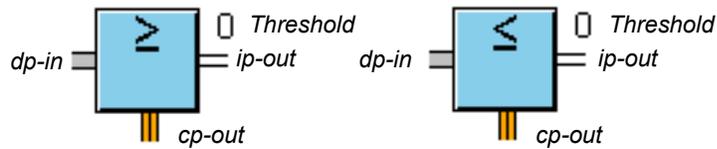
The blocks listed below perform basic logical operations on inference values:

- The [AND Gate](#) block passes `.true` if all its input status values are `.true`.
- The [OR Gate](#) block passes `.true` if any of its input status values are `.true`.
- The [NOT Gate](#) block passes the opposite of its input status value.

Pausing Paths

The [Inference Inhibit](#) block lets you enable or disable an inference path depending on another inference value.

High Value Observation, Low Value Observation



The High Value block and the Low Value block test whether an input value is above or below the Threshold attribute. The High Value block tests whether the input value is greater than or equal to the Threshold. The Low Value block tests whether the input value is less than or equal to the Threshold.

Specifying a Threshold

The block compares the attribute Threshold against the input value.

- In a High Value block, the comparison returns `.true` if the value is greater than or equal to the Threshold and returns `.false` otherwise.
- In a Low Value block, the comparison returns `.true` if the value is less than or equal to the Threshold, and returns `.false` otherwise.

Note When you use High Value or Low Value with NeurOn-Line, make sure Threshold Uncertainty is set to `none`. This field is for fuzzy logic, which NeurOn-Line doesn't support. If you want to use fuzzy logic and have GDA, refer to the GDA documentation.

Configuring

This is the configuration panel for the High Value block. The panel for the Low Value block is identical except for the block name.

Attribute	Description
Threshold	The value against which the block compares its inputs to determine the output inference value.
Threshold Uncertainty	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Output Uncertainty	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Status on Initialization	See Specifying an Initial Status Value in the <i>NeurOn-Line User's Guide</i> .

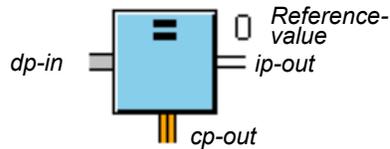
Attribute	Description
Hysteresis When	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Description when True, Description when False, Description when Unknown, and Description of Input	These attributes allow you to display explanations, which NeurOn-Line does not support.

See Also

For more information on attributes and menu choices that are not described in this section, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Overriding Block Values		<i>User's Guide</i>
Editing Attribute Displays		<i>User's Guide</i>

Equality Observation

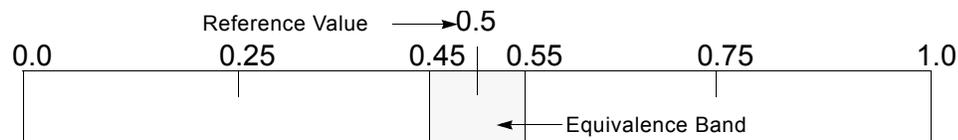


The Equality block checks whether an input value is equal to the attribute Reference Value. It passes `.true` if they are equal and `.false` otherwise. The Reference Value can either be a number, a symbol, or a text string, but its data type must match the data type of the input value.

If you want to check whether the input is approximately equal to the Reference Value, use the Equivalence Band attribute. When the input value is in the following range, the Equality block considers the values equal and passes `.true`:

$$\text{Reference Value} - \text{Equivalence Band}/2 \leq \text{input value} < \text{Reference Value} + \text{Equivalence Band}/2$$

For example, if an Equality block has a Reference Value of 0.5 and an Equivalence Band of 0.1, as shown in the following figure, it passes `.true` for when the input value is less than 0.55 and greater than or equal to 0.45. This attribute is especially useful for comparing floating-point numbers, which can differ by small amounts due to rounding errors but still appear to be equal.



Configuring

This is the configuration panel for the Equality block.

The screenshot shows a configuration window titled "Equality". It contains the following settings:

- Output Uncertainty: 0.5
- Reference Value: 0
- Equivalence Band: none
- Status on Initialization: none (selected)
- Hysteresis When: none (selected)

At the bottom of the window, there is a "Descriptions" button and three buttons: "OK", "Apply", and "Cancel".

Attribute	Description
Output Uncertainty	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Reference Value	The value against which the block compares its inputs to determine equality.
Equivalence Band	The amount of uncertainty that the block applies to the input value to determine equivalency. The number represents an uncertainty band around the input value.
Status on Initialization	See Specifying an Initial Status Value in the <i>NeurOn-Line User's Guide</i> .

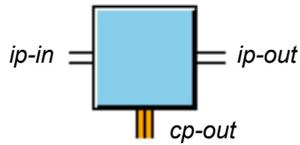
Attribute	Description
Hysteresis When	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Description when True, Description when False, Description when Unknown, and Description of Input	These attributes allow you to display explanations, which NeurOn-Line does not support.

See Also

For more information on attributes and menu choices that are not described in this section, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Overriding Block Values		<i>User's Guide</i>
Editing Attribute Displays		<i>User's Guide</i>
Evaluating Expressions in Attributes		<i>User's Guide</i>

Conclusion



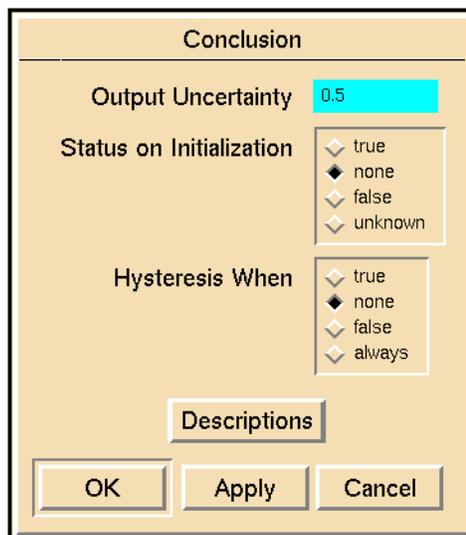
The Conclusion block lets you perform these actions after drawing a conclusion:

- Activate action blocks.
- Override the path's value with the **override** menu choice.

It is especially useful after blocks that do not connect to action blocks, such as the AND, OR, and NOT gates.

Configuring

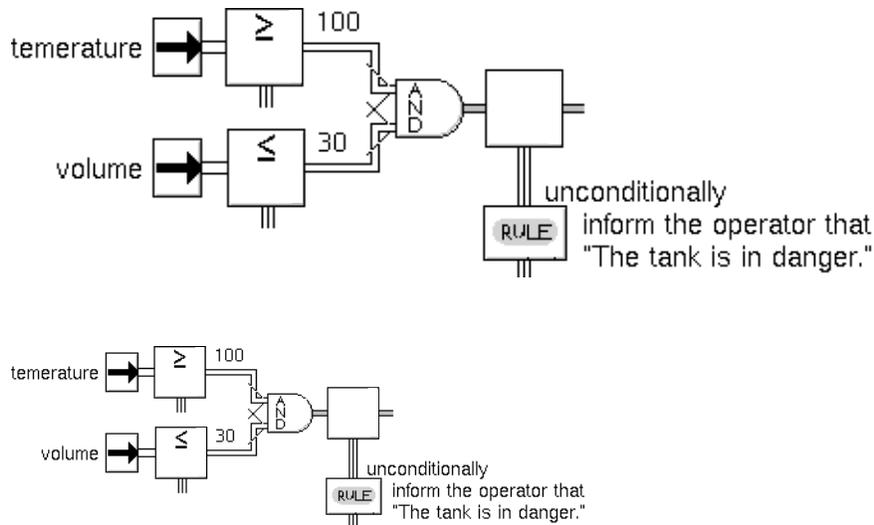
This is the configuration panel for the Conclusion block.



Attribute	Description
Output Uncertainty	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Status on Initialization	See Specifying an Initial Status Value in the <i>NeurOn-Line User's Guide</i> .
Hysteresis When	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Description when True, Description when False, Description when Unknown, and Description of Input	These attributes allow you to display explanations, which NeurOn-Line does not support.

Example

The figure below shows a Conclusion block connected to an AND Gate that concludes whether both the temperature of a tank is high and the volume of the tank is low. The Conclusion block is attached to a Rule Action block that writes a warning to the Message Board.

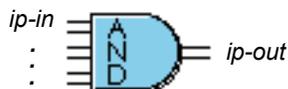


See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Overriding Block Values		<i>User's Guide</i>

AND Gate



The AND Gate passes `.true` if all its input status values are `.true`.

This table lists what the AND Gate passes:

If...	It passes...
Any input is <code>.false</code>	<code>.false</code>
All inputs are <code>.true</code>	<code>.true</code>
Maximum Unknown Inputs or fewer are <code>unknown</code> , and the rest are <code>.true</code>	<code>.true</code>
More than Maximum Unknown Inputs are <code>unknown</code> , and the rest are <code>.true</code>	<code>unknown</code>
All inputs are <code>unknown</code>	<code>unknown</code>

Note If any input is `.false`, the AND Gate always passes `.false`, even if the number of `unknown` inputs is greater than Maximum Unknown Inputs.

How the Block Handles no-value Quality Inputs

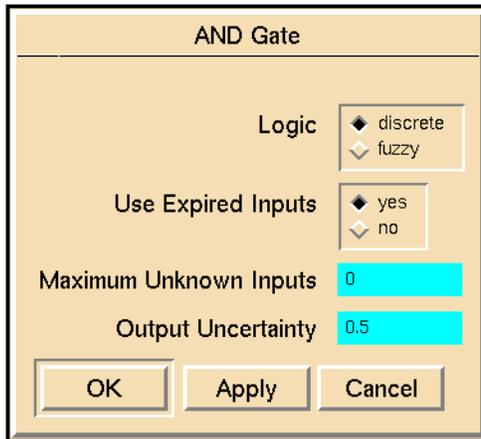
An input path having a value of `unknown` and a Quality of `no-value` is not counted toward the number of Maximum Unknown Inputs.

For example, an AND Gate has 3 inputs and the Maximum Unknown Inputs is 1. If one of the inputs has a value of `true`, another a value of `unknown` and a Quality of `OK`, and the third a value of `unknown` and a Quality of `no-value`, the block output is `true`. When determining whether the number of `unknown` inputs exceeds the Maximum Unknown Inputs, the third path is ignored.

For more information about how blocks handle `no-value` inputs, see the *GDA User's Guide*.

Configuring

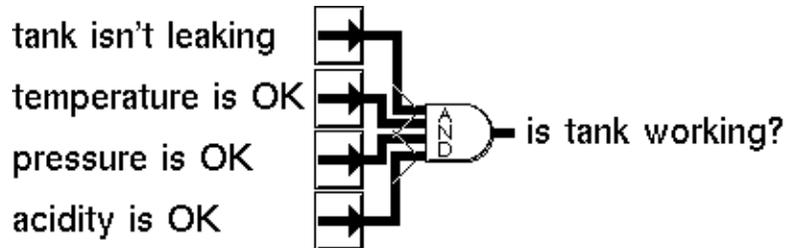
This is the configuration panel for the AND Gate.



Attribute	Description
Logic	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Use Expired Inputs	See Determining Output Path Attributes for Peer Input Blocks in the <i>NeurOn-Line User's Guide</i> .
Maximum Unknown Inputs	The number of inputs that can have a Status-value of unknown when determining whether the output inference value is .true or unknown.
Output Uncertainty	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.

Example

This figure shows an AND Gate connected to four Belief Entry Points:



This table shows some sample input and output values for the AND block in the previous figure, with Maximum Unknown Inputs set to 2:

If the input values are...				The block passes...
.true	.false	.true	.true	.false
.true	.true	.true	.true	.true
unknown	unknown	.false	unknown	.false
unknown	.true	unknown	unknown	unknown
.true	.true	unknown	unknown	.true

See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Connecting to Peer Input Blocks		<i>User's Guide</i>

OR Gate



The OR Gate passes `.true` if any of its input status values are `.true`.

This table lists what the OR Gate passes:

If...	It passes
Any input is <code>.true</code>	<code>.true</code>
All inputs are <code>.false</code>	<code>.false</code>
Maximum Unknown Inputs or fewer are <code>unknown</code> , and the rest are <code>.false</code>	<code>.false</code>
More than Maximum Unknown Inputs are <code>unknown</code> , and the rest are <code>.false</code>	<code>unknown</code>
All inputs are <code>unknown</code>	<code>unknown</code>

Note If any input is `.true`, the OR Gate always passes `.true`, even if the number of unknown inputs is greater than Maximum Unknown Inputs.

Configuring

This is the configuration panel for the OR Gate.

OR Gate

Logic discrete fuzzy

Use Expired Inputs yes no

Maximum Unknown Inputs 0

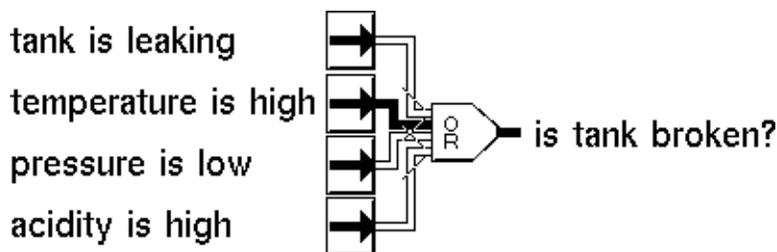
Output Uncertainty 0.5

OK Apply Cancel

Attribute	Description
Logic	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Use Expired Inputs	See Determining Output Path Attributes for Peer Input Blocks in the <i>NeurOn-Line User's Guide</i> .
Maximum Unknown Inputs	The number of inputs that can have a Status-value of unknown when determining whether the output inference value is .false or unknown.
Output Uncertainty	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.

Example

This figure shows an OR Gate connected to four entry points:



This table shows some sample input and output values for the OR block in the previous figure, with Maximum Unknown Inputs set to 2:

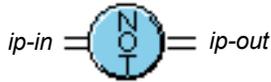
If the input values are...				The block passes...
.true	.false	.true	.true	.true
.false	.false	.false	.false	.false
unknown	unknown	.false	unknown	unknown
unknown	.true	unknown	unknown	.true
.true	.true	unknown	unknown	.true

See Also

For more information on this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Connecting to Peer Input Blocks		<i>User's Guide</i>

NOT Gate

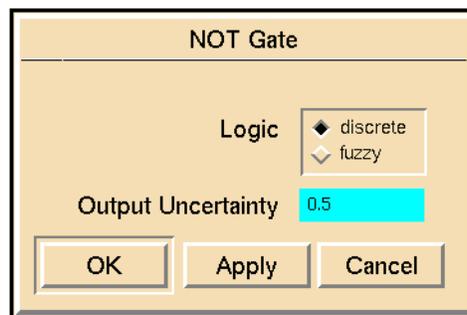


The NOT Gate passes the logical inverse of its input value, as this table shows:

If the input is...	It passes...
.true	.false
.false	.true
unknown	unknown

Configuring

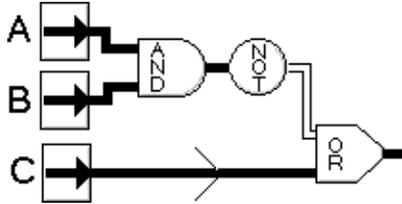
This is the configuration panel for the NOT Gate.



Attribute	Description
Logic	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.
Output Uncertainty	This attribute is for fuzzy logic, which NeurOn-Line does not support. If you want to use fuzzy logic and have GDA, refer to the documentation that came with GDA.

Example

This figure shows a diagram that implements NOT(A AND B) OR C:



See Also

For more information on this block, see the sections below.

For more information on...

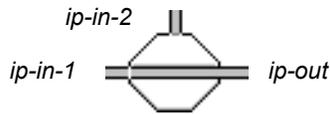
See...

In this book...

Basic Block Behavior

User's Guide

Inference Inhibit



The Inference Inhibit block enables and disables an inference path. You can use it to control entire flow diagrams or an important subsection of one.

When the status value of the top inference path (*ip-in-2*) matches the value of the attribute *Trigger On*, the block inhibits the bottom input inference value (*ip-in-1*). When the block is inhibiting the inference path, it does not evaluate any attached chart or graph capabilities nor does it update their associated charts or graphs.

When the status value of the top inference path does not match *Trigger On*, the block passes the bottom input inference value normally.

If *Status on Initialization* has a value, the block passes that value when it is initialized or reset; otherwise it passes nothing when initialized or reset.

Also, if the block has a *Status on Initialization*, and the top inference path value matches the *Trigger On*, the block passes the *Status on Initialization*.

NeurOn-Line evaluates this block whenever either of the input ports receives a new value.

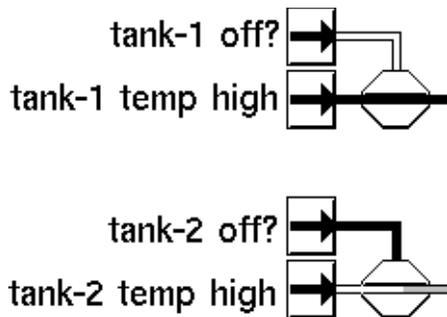
Configuring

This is the configuration panel for the Inference Inhibit block.

Attribute	Description
Trigger On	The status value that determines when the block inhibits the flow of data.
Status on Initialization	See Specifying an Initial Status Value in the <i>NeurOn-Line User's Guide</i> .

Example

This figure shows a portion of a flow diagram that uses two Data Inhibit blocks to test whether a tank is on before analyzing its temperature. Trigger On is `.true`. Tank-1 is on and Tank-2 is off.



See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
The Data Inhibit block		<i>Reference Manual</i>
The Control Inhibit block		<i>Reference Manual</i>

Capabilities

Describes the blocks that add features and attributes to other NeurOn-Line blocks, such as charts, graphs, and clocks.

Introduction **321**

Chart Capability **323**

Clock **334**

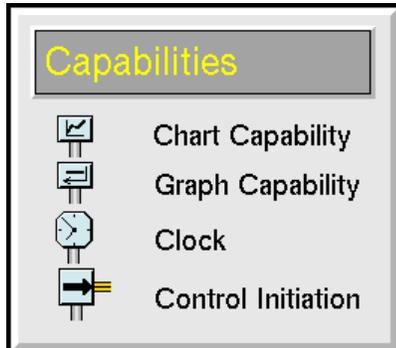
Control Initiation Capability **336**



Introduction

Capabilities are special blocks that let you add features and attributes to other NeurOn-Line Blocks. They attach directly to the top of other blocks. A Capability is a small attachment on top of a block that adds functionality to the block. For example, a Control Initiation Capability lets a block pass a control signal every time its evaluated, and a Graph Capability lets a block display its values on a graph.

You can find the Capabilities palette under the Action & Other submenu of the Palettes menu:



Charting and Graphing Attributes

These Capabilities let you display a NeurOn-Line block's data on a G2 chart or graph:

- The [Chart Capability](#) lets you display a block's data on a chart. It lets you configure the chart from a dialog and change how the data is displayed.
- The Graph Capability lets you display a block's data on a graph. It lets you change how the data is displayed.

Forcing a Block to Evaluate

The [Clock](#) block forces the attached block to evaluate at specified times.

Starting a Control Signal

The [Control Initiation Capability](#) block passes a control signal whenever the attached block receives a value.

Chart Capability



A Chart Capability is a link between the attribute of a block you want to display and the chart on which it is displayed. The Capability lets you choose where and how the attribute's data is displayed. In addition, it adds a menu choice to the chart that lets you modify the chart.

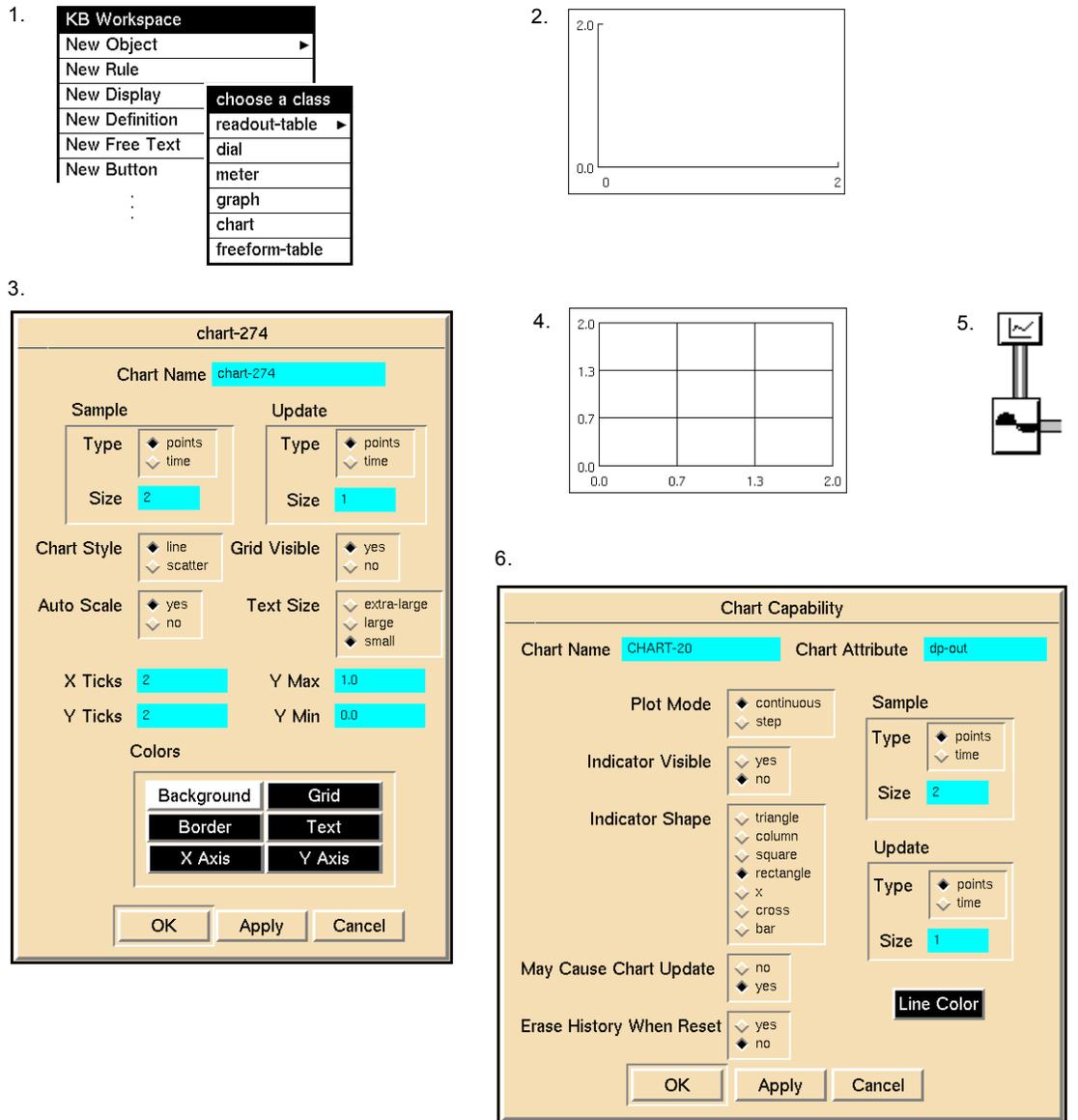
You can attach a Chart capability to any block that has a named data or inference input or output path, for example, dp-out or ip-in.

Setting Up a Chart

To set up a chart to display data from a block:

- 1 Create a new chart using KB Workspace > New Display. This attaches a new display to your mouse.
- 2 Position the display on the workspace and click with the mouse to place it.
- 3 Select **configure** from the chart's menu to display the configuration panel for the display.
- 4 Configure the chart and select the OK button to update the display.
NOL creates a Chart Capability with a system-defined chart name and places it next to the display.
- 5 Connect the Chart Capability to the block whose values you want to plot.
- 6 Select **configure** from the capability's menu to display its configuration panel. Configure the capability and select the OK button. For more information, see [Configuring a Chart](#).

The following figure illustrates these steps for a chart:



Caution When there is a single capability associated with a particular chart, NOL prevents you from configuring the chart and its associated capability at the same time. When there are multiple capabilities that plot data on a single chart, however, you should avoid simultaneous editing of the display and its capability, because NOL cannot prevent this.

Configuring a Chart

You configure the chart associated with its capability by using a configuration panel. The panel lets you set how much data the chart displays, how frequently the chart is updated, and what its axes look like.

To display the configuration panel for a chart, select **configure** from the chart's menu as shown:

chart-274

Chart Name

Sample		Update	
Type	<input checked="" type="radio"/> points <input type="radio"/> time	Type	<input checked="" type="radio"/> points <input type="radio"/> time
Size	<input type="text" value="2"/>	Size	<input type="text" value="1"/>
Chart Style	<input checked="" type="radio"/> line <input type="radio"/> scatter	Grid Visible	<input checked="" type="radio"/> yes <input type="radio"/> no
Auto Scale	<input checked="" type="radio"/> yes <input type="radio"/> no	Text Size	<input type="radio"/> extra-large <input type="radio"/> large <input checked="" type="radio"/> small
X Ticks	<input type="text" value="2"/>	Y Max	<input type="text" value="1.0"/>
Y Ticks	<input type="text" value="2"/>	Y Min	<input type="text" value="0.0"/>
Colors			
<input type="button" value="Background"/>		<input type="button" value="Grid"/>	
<input type="button" value="Border"/>		<input type="button" value="Text"/>	
<input type="button" value="X Axis"/>		<input type="button" value="Y Axis"/>	
<input type="button" value="OK"/>		<input type="button" value="Apply"/>	
<input type="button" value="Cancel"/>			

Setting the Amount of Data Displayed

The field Sample Type specifies how points are displayed on the chart and how the field Sample Size is interpreted.

If Sample Type is...	Data points are displayed...	And Sample Size is...
Points	At equal intervals, regardless of when they arrived.	Number of points to display.
Time	According to the time they arrived.	Time interval to display.

For example, if Sample Type is Points and Sample Size is 60, the chart displays the last 60 points. If Sample Type is Time and Sample Size is 60 seconds, the chart displays the points received in the 60 second interval before the block last evaluated.

Setting How Frequently a Chart is Updated

When a block attached to a Chart Capability receives a value, the Chart Capability checks the setting of the attribute May Cause Chart Update. If that attribute is **yes**, the capability signals the chart to let it know it has received a new value. The chart then decides whether to update itself by checking the settings of Update Type and Update Size. If Update Type is Points, the chart is updated if it has received a total of Update Size signals since the last update. If Update Type is Time, the chart is updated if Update Size seconds have passed since the last update.

If Update Type is...	The chart is updated when a capability signals it and ...
Points	The chart has received a total of Update Size signals since the last update.
Time	Update Size seconds have passed since the last update.

Note Updating a chart can consume a significant amount of your computer's time.

Specifying the Axes

To set how many tickmarks are on the chart's axes, edit X Ticks and Y Ticks. To display the tickmarks on the chart, set Grid Visible to **true**. To turn off the tickmarks, set Grid Visible to **false**.

To set the upper and lower limits for the Y axis, edit Y Max and Y Min, and set Auto Scale to **no**. To cause the chart to set its own upper and lower limits, set Auto Scale to **yes**; the chart ignores the values of Y Max and Y Min.

Determining How the Plot is Displayed

To chart values using a line between plot points, set Chart Style to **line**.

To chart values using points only, with no line between points, set Chart Style to **scatter**. When using a scatter-type plot, you must also set Indicator Visible to **yes** in the Chart Capability's configuration panel as shown in [Choosing How a Block's Data is Displayed](#).

Setting the Chart Colors

You can set the following colors for a chart:

The color labeled...	Sets...
Background	The background color of the entire chart
Grid	The line color of the grid
Border	The line color of the border
Text	The text color of all text in the chart
X Axis	The line color of the x axis
Y Axis	The line color of the y axis

To set a color:

- 1 Click on the color you want to set.
- 2 Choose a color from the list of colors in the scroll area.
- 3 Select the OK button in the scroll area.
- 4 Select the Apply or OK button in the configuration panel.

Choosing How a Block's Data is Displayed

By default, a Chart Capability displays a block's data as a black line on the chart. To change how a Chart Capability displays its data on its chart, edit the capability's attribute table.

Specifying the Name of the Chart

The name of the chart must correspond to the Chart Name specified in the capability. NOL automatically inserts a system-generated name in the configuration panel for the Chart Capability when you configure the associated chart; thus, there is no need to update these attributes in the capability unless you want to change the name. For a description, see step 4 under [Setting Up a Chart](#).

You can also enter an expression that evaluates to the name of a chart. For example, you can use an expression such as “[chart-variable]” to refer to a variable whose value is the name of a chart, or you can use an expression such as “[the chart-name of chart-object]” to refer to an attribute of an object whose value is the name of a chart.

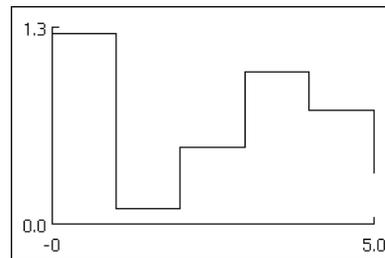
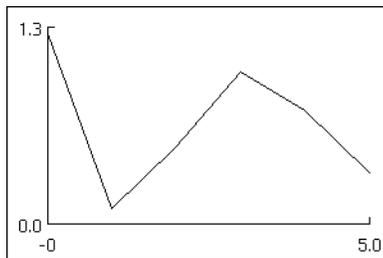
Specifying The Attribute to Plot

The attribute Chart Attribute determines which attribute is plotted in the chart. The default value of this attribute is `dp-out`, which plots the data output value of the block to which the capability is connected. For example, when plotting entry points and signal generators, you do not need to edit this attribute.

Specifying the Type of Connection Between Points

To choose how the display connects the points in a block’s data set, set the attribute Plot Mode to one of the values in the following table. The figure shows you what a chart that uses these modes looks like.

If Plot Mode is...	Then the display...
continuous	Directly connects the points in the block’s dataset.
discrete	Inserts extra points into the block’s dataset so the transitions are square.



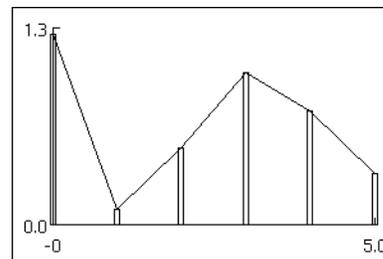
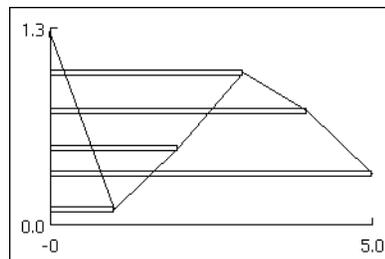
This attribute is ignored by charts when Chart Style is `scatter` in the Chart Capability configuration panel as discussed in [Determining How the Plot is Displayed](#).

Specifying a Marker

In a Chart Capability, you can mark each point in a block's data set. Set the attribute Indicator Visible to **yes**. To choose a marker, set the attribute Indicator Shape to one of these values:

Attribute	Picture
rectangle	□
square	■
triangle	△
cross	+
x	×
bar	see below
column	see below

The bar and column indicators are a little different from the rest. They draw a line from the point to an axis. A bar goes to the Y axis, and a column goes to the X axis, as shown in this figure:



Specifying Whether a Capability Can Update a Display

In a Chart Capability, you can choose whether getting a new value can force the chart to update. If the attribute May Cause Chart Update is **yes**, the capability signals the chart every time it receives a new value, and the chart may update if the time is right. (You determine when the right time is by setting the fields Update Type and Update Size in the chart's **configure** dialog, described in [Setting How Frequently a Chart is Updated](#).) If May Cause Chart Update is **no**, the capability does not let the chart know it has received a new value.

If several Chart Capabilities use a single chart, it is good practice to let only the one furthest downstream update the chart. This practice lessens the time NOL spends updating charts and makes sure that the displayed values are consistent.

Specifying the Line Color

To specify the color of the line used to plot data in a chart:

- 1 Click on Line Color in the configuration panel for the Chart Capability.
- 2 Choose a color from the color palette.
- 3 Select the OK button in the scroll area.
- 4 Select the Apply or OK button in the configuration panel.

Going to a Chart

Sometimes you need to put a chart on a different workspace from a Chart Capability. To see the display for a capability, choose **go to chart** from the capability's menu. NOL brings the workspace containing the display to the front.

Resetting

What happens when you reset a Chart Capability depends on the setting of the attribute Erase History When Reset. If Erase History When Reset is **yes**, the capability deletes its history of values. The next time it plots a point, it erases the line before plotting the new point.

Configuring

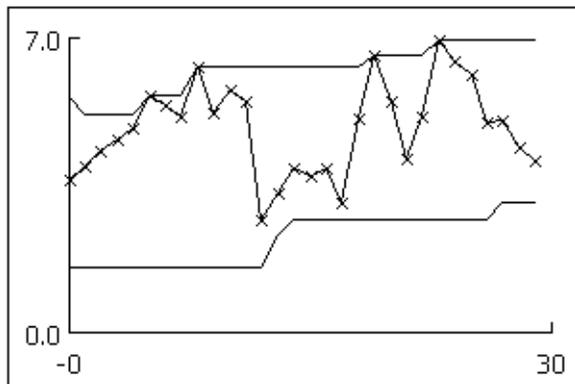
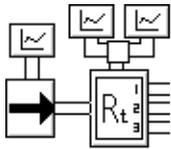
This is the configuration panel for the Chart Capability.

Attribute	Description
Chart Name	The name of the chart on which the capability plots its data.
Chart Attribute	The attribute of the block to which the capability is attached whose data will plot.
Plot Mode	Whether the capability connects the data points to form a continuous plot, or whether the capability inserts additional points to create a step-like plot.
Indicator Visible	Whether the data points are visible or not.
Indicator Shape	When Indicator Visible is yes , the shape of the indicators that are the data points of the plot.

Attribute	Description
May Cause Chart Updating	Whether the chart automatically updates each time it receives a new data value or not.
Erase History When Reset	See Specifying What Happens to History Upon Reset in the <i>NeurOn-Line User's Guide</i> .
Sample Type and Sample Size	See Specifying the Size of the History in the <i>NeurOn-Line User's Guide</i> .
Update Type and Update Size	See Specifying When to Propagate Data in the <i>NeurOn-Line User's Guide</i> .
Line Color	The color of the plot.

Examples

The Chart Capabilities in the following figure display the values from an Entry Point, in addition to the minimum and maximum values from the entry point for the past fifteen points. The Entry Point's values are marked with an X.



The following table lists some of the attribute values for the Chart Capabilities. Notice that there are two capabilities attached to the Moving Range block: one charts the maximum and one charts the minimum.

Attribute Name	Entry Point	Maximum	Minimum
Chart Name	chart-1	chart-1	chart-1
Chart Attribute	dp-out	dp-out-1	dp-out-3
Line Color	black	black	black
Indicator	x	rectangle	rectangle
Indicator Visible	yes	no	no
Plot Mode	continuous	continuous	continuous
May Cause Chart update	no	no	yes
Erase History When Reset	no	no	no

See Also

For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>
Evaluating Expressions in Attributes		<i>User's Guide</i>

Clock



The Clock capability forces the attached block to evaluate at a specified interval and not necessarily when the block receives new input data. You can attach a Clock capability to any block.

To specify the interval, set the attribute Evaluation Period. For example, if Evaluation Period is 1 minute, the block will evaluate once every minute.

If you do not want the block to evaluate when it receives new data, set Allow Intermediate Evaluation to no. The block will evaluate only at the specified interval.

To turn the Clock capability on and off, use the menu choices enable evaluation and disable evaluation.

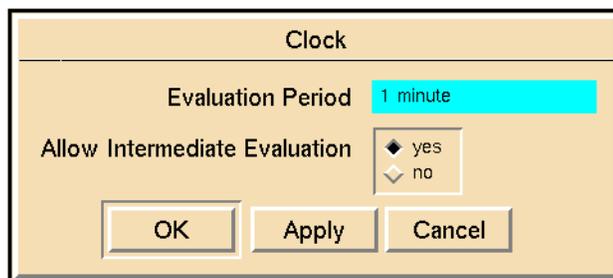
When you turn the clock off, the block to which the Clock capability is attached evaluates whenever it receives a new value, as opposed to every Evaluation Period. When you turn the clock on, the attached block evaluates once every Evaluation Period.

If the Clock capability has Allow Intermediate Evaluation set to no, the attached block does not fire when the clock is turned off.

You can attach a Clock capability to a Graph Capability to cause the attached capability to update at regular intervals.

Configuring

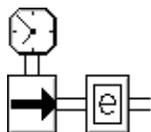
This is the configuration panel for the Clock capability.



Attribute	Description
Evaluation Period	The time interval at which the clock evaluates.
Allow Intermediate Evaluation	Whether the clock also evaluates when it receives new data (yes) or only every Evaluation Period (no).

Example

The Clock capability in this figure forces the Data Entry Point to pass a value at a regular interval, even if its value does not change:

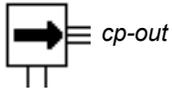


See Also

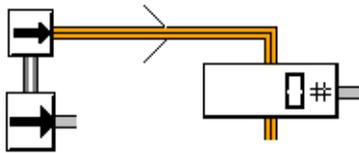
For more information on how to use this block, see the sections below.

For more information on...	See...	In this book...
Basic Block Behavior		<i>User's Guide</i>

Control Initiation Capability



The Control Initiation Capability initiates a control action whenever its associated block receives a value. You connect the capability to any block via the link, and you connect the control path to any control action. For example, you can use a Control Initiation Capability to increment a counter each time a block evaluates, as shown in the following diagram:



Configuring

The Control Initiation Capability has no configurable attributes.

See Also

For more information on how to use this block, see the sections below.

For more information on...

See...

In this book...

Basic Block Behavior

User's Guide

Application Programmer's Interface

Chapter 12: Application Programmer's Interface

Describes the NeurOn-Line Application Programmer's Interface.

Application Programmer's Interface

Describes the NeurOn-Line Application Programmer's Interface.

Introduction	339
Accessing the NOL API Procedures	340
Path Displays	341
Vector Blocks	342
Data Set Blocks	351
Neural Networks	360
Action Utilities	367
File Operations	369



Introduction

This section describes how to use the NeurOn-Line Application Programmer's Interface (API), and it serves as a quick reference guide for application developer.

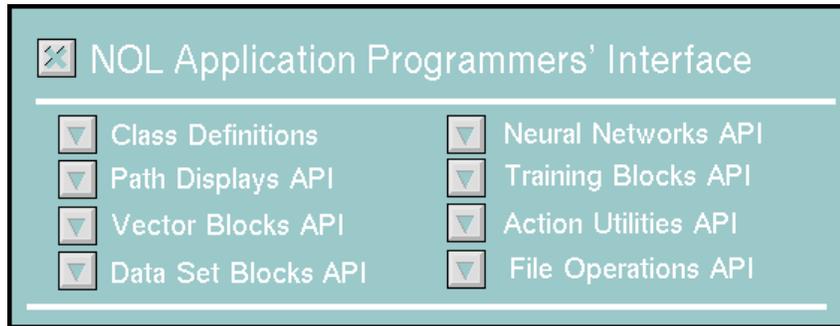
You can control NOL objects from within a G2 procedure or function by using the API procedures. The API includes procedures that perform the same actions as executing NeurOn-Line blocks.

Accessing the NOL API Procedures

To see the declarations of the NeurOn-Line functions:

→ Choose Help > Programmer's Interface menu choice.

NeurOn-Line displays this workspace:



Click the subworkspace buttons to look at the declarations. Many of the API procedures require the following arguments:

- The block to execute.
- The mode of execution, which can be one of two symbols: **system** or **manual**. Typically, you use **system**, although most NeurOn-Line procedures do not use this argument.
- The input value or values that the block requires.
- A list or lists of objects or other blocks that the block uses. These are equivalent to the blocks that would be connected to the executing block with an action link, if the block were on a diagram.
- The output value or values that the block produces.

If a block has an input or output data path, the API procedure for the block requires an object of class **data-path-value**. Your calling procedure must create the **data-path-value** objects. If the argument represents an input to the block, assign the input value to the **data-value** attribute of the **data-path-value** object. If the block has vector inputs or outputs, the block's procedure passes a **vector-path-value** object. If the block has data pair inputs or outputs, the block's procedure passes a **data-pair** object.

To see the class definitions for **data-path-value**, **vector-path-value** and **data-pair**, click the Class Definitions button. The other APIs are listed according to the sequence of the layout in individual subworkspace.

Path Displays

nol-execute-data-path-display

(*path-display*: class `gdl-data-path-display`, *mode*: symbol,
input-data: class `data-path-value`, *output-data*: class `data-path-value`)

Display the value, quality, or collection time of the input data in a Data Path Display block.

Parameter	Description
<i>path-display</i>	The Data Path Display block whose data is to be displayed.
<i>mode</i>	The mode of execution, which is not used in this case. You can use the symbol <code>system</code> to be consistent with other APIs.
<i>input-data</i>	The input data path value passed into the block.
<i>output-data</i>	The output data path value produced by the block.

nol-execute-vector-path-display

(*path-display*: class `vector-path-display`, *mode*: symbol,
input-vector: class `vector-path-value`, *output-vector*: `item-or-value`)

Display the vector values, quality, or collection time of the input vector in a Vector Path Display block.

Parameter	Description
<i>path-display</i>	The Vector Path Display block whose data is to be displayed.
<i>mode</i>	The mode of execution, which is not used in this case. You can use the symbol <code>system</code> to be consistent with other APIs.
<i>input-vector</i>	The input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.

Vector Blocks

nol-execute-windower

(*blk*: class windower, *mode*: symbol, *input-data*: class data-path-value, *output-vector*: class vector-path-value)

Pushes the data path value into a Windower block to create a vector from consecutive input values. The block combines the specified number of previous input values into a vector, with the most recently received value being the first element of the vector. The block does not pass an output vector until it has received enough input to fill its window.

Parameter	Description
<i>blk</i>	The Windower block used to process data into a vector.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-data</i>	The input data path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.

nol-execute-classifier-input-converter

(*blk*: class classifier-input-converter, *mode*: symbol, *input-data*: class data-path-value, *output-vector*: class vector-path-value)

Converts an integer to a vector, using a Classifier Input Converter block. You can use the output vector as a training target for a classification problem.

Parameter	Description
<i>blk</i>	The Classifier Input Converter block used to process data into a vector.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .

Parameter	Description
<i>input-data</i>	The input data path value passed into the block. The data should be an integer, which specifies which class element of output vector should be 1. If it is a floating-point number, it will be rounded to the nearest integer value.
<i>output-vector</i>	The output vector path value produced by the block. The vector contains all zeros except the element specified by the input integer.

nol-execute-classifier-output-converter

(*blk*: class classifier-output-converter, *mode*: symbol,
input-vector: class vector-path-value, *output-data*: class data-path-value)

Outputs the index of the largest element in the input vector, using a Classifier Output Converter block. The largest value in the output vector usually corresponds to the most likely class as predicted by the network.

Parameter	Description
<i>blk</i>	The Classifier Output Converter block used to process vector into data.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector</i>	The input vector path value passed into the block.
<i>output-data</i>	The output data path value produced by the block. The data will be an integer, which points to the largest element of the input vector.

nol-execute-vector-combiner

(*blk*: class vector-combiner, *mode*: symbol,
input-vector1: class vector-path-value, *input-vector2*: class vector-path-value,
output-vector: class vector-path-value)
-> *truth-value*

Combines two vectors together by appending the second vector to the end of the first, using a Vector Combiner block.

Parameter	Description
<i>blk</i>	The Vector Combiner block used to process the vectors.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector1</i>	The first input vector path value passed into the block.
<i>input-vector2</i>	The second input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block. The vector will be the combination of the two input vectors.

Return Value	Description
<u><i>truth-value</i></u>	Returns false if the quality of either input path is no-value , and no value will be passed. Otherwise returns true .

nol-execute-vector-splitter

(*blk*: class vector-splitter, *mode*: symbol,
input-vector: class vector-path-value, *output-vector1*: class vector-path-value,
output-vector2: class vector-path-value)
-> *truth-value*, *truth-value*

Splits the input vector into two smaller vectors, using a Vector Splitter block. Specify the size of the top input vector with the attribute Output 1 Dimension of the block. The bottom vector contains what remains of the input vector after removing the specified number of elements.

If the size of the input vector is less than the Output 1 Dimension, the block generates an error.

Parameter	Description
<i>blk</i>	The Vector Splitter block used to process the vectors.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector</i>	The input vector path value passed into the block.
<i>output-vector1</i>	The first output vector path value produced by the block.
<i>output-vector2</i>	The second output vector path value produced by the block.

Return Value	Description
<u><i>truth-value</i></u>	Returns false if the first vector dimension is less than zero. Otherwise returns true .
<u><i>truth-value</i></u>	Returns false if the second vector dimension is less than zero. Otherwise returns true .

nol-execute-vector-order-swapper

(*blk*: class vector-order-swapper, *mode*: symbol, *input-vector*: class vector-path-value, *output-vector*: class vector-path-value)

Reorders the elements of a vector, using a Vector Order Swapper block.

If the dimension on the input vector does not equal the dimension of the block's vector, the block generates an error.

Parameter	Description
<i>blk</i>	The Vector Order Swapper block used to process the vectors.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector</i>	The input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.

nol-execute-vector-rescaler

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

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

Parameter	Description
<i>blk</i>	The Vector Rescaler block used to process the vectors. The block contains scale factors for each element.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector</i>	The input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.

nol-execute-vector-sum

(*blk*: class vector-sum, *mode*: symbol, *input-vector1*: class vector-path-value,
input-vector2: class vector-path-value, *output-vector*: class vector-path-value)
-> truth-value

Adds the elements of the two input vectors, using a Vector Sum block. The procedure passes a vector in which each element is the sum of the input vectors' corresponding elements.

If the two input vectors have different lengths, the block generates an error.

Parameter	Description
<i>blk</i>	The Vector Sum block used to process the vectors.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector1</i>	The first input vector path value passed into the block.
<i>input-vector2</i>	The second input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.

Return Value	Description
<u><i>truth-value</i></u>	Returns false if the quality of either input path is no-value , and no value will be passed. Otherwise returns true .

nol-execute-vector-difference

(*blk*: class vector-difference, *mode*: symbol,
input-vector1: class vector-path-value, *input-vector2*: class vector-path-value,
output-vector: class vector-path-value)
-> *truth-value*

Subtracts the elements of the second input vector from the elements of the first input vector, using a Vector Difference block.

If the two input vectors have different lengths, the block generates an error.

Parameter	Description
<i>blk</i>	The Vector Difference block used to process vectors.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector1</i>	The first input vector path value passed into the block.
<i>input-vector2</i>	The second input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.

Return Value	Description
<u><i>truth-value</i></u>	Returns false if the quality of either input path is no-value , and no value will be passed. Otherwise returns true .

nol-execute-vector-product

(*blk*: class vector-product, *mode*: symbol,
input-vector1: class vector-path-value, *input-vector2*: class vector-path-value,
output-vector: class vector-path-value)
-> *truth-value*

Multiplies the elements of the two input vectors, using a Vector Product block. The procedure passes a vector in which each element is the product of the input vectors' corresponding elements.

If the two input vectors have different lengths, the block generates an error.

Parameter	Description
<i>blk</i>	The Vector Product block used to process the vectors.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector1</i>	The first input vector path value passed into the block.
<i>input-vector2</i>	The second input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.
Return Value	Description
<u><i>truth-value</i></u>	Returns false if the quality of either input path is no-value , and no value will be passed. Otherwise returns true .

nol-execute-vector-quotient

(*blk*: class vector-quotient, *mode*: symbol,
input-vector1: class vector-path-value, *input-vector2*: class vector-path-value,
output-vector: class vector-path-value)
 -> *truth-value*

Divides the elements of the first input vector by the elements of the second input vector, using a Vector Quotient block.

If the two input vectors have different lengths, the block generates an error.

Parameter	Description
<i>blk</i>	The Vector Quotient block used to process the vectors.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector1</i>	The first input vector path value passed into the block.
<i>input-vector2</i>	The second input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.
Return Value	Description
<u><i>truth-value</i></u>	Returns false if the quality of either input path is no-value , and no value will be passed. Otherwise returns true .

nol-execute-vector-function

(*blk*: class vector-function, *mode*: symbol,
input-vector: class vector-path-value, *output-vector*: class vector-path-value)

Applies a function or procedure to the input vector and passes the result, using a Vector Function block. Specify the name of the function in the Arithmetic Function attribute of the block. The function can be a built-in G2 function, user-defined function, procedure, or tabular-function.

Parameter	Description
<i>blk</i>	The Vector Function block used to process the vector.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector</i>	The input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.

nol-execute-vector-function-of-two-args

(*blk*: class vector-function-of-two-args, *mode*: symbol,
input-vector1: class vector-path-value, *input-vector2*: class vector-path-value,
output-vector: class vector-path-value)
-> truth-value

Applies a function or procedure to the two input vectors and passes one vector as the result, using a Vector Function of Two Arguments block. Specify the name of the function in the Arithmetic Function attribute of the block. The function can be a built-in G2 function, user-defined function, or procedure.

Parameter	Description
<i>blk</i>	The Vector Function of Two Arguments block used to process the vector.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector1</i>	The first input vector path value passed into the block.
<i>input-vector2</i>	The second input vector path value passed into the block.
<i>output-vector</i>	The output vector path value produced by the block.

Return Value	Description
<u><i>truth-value</i></u>	Returns false if the quality of either input path is no-value , and no value will be passed. Otherwise returns true .

Data Set Blocks

nol-execute-data-pair-buffer

(*blk*: class data-pair-buffer, *mode*: symbol,
input-vector1: class vector-path-value, *input-vector2*: class vector-path-value,
output-dp: class data-pair)
 -> truth-value

Stores the input vectors into a data pair buffer, using a Data Pair Buffer block. The first input becomes the X vector, and the second input becomes the Y vector. If new data is output, that data will be placed into the output data pair. The concurrency of input values is determined by the timestamps of the two input vectors.

Parameter	Description
<i>blk</i>	The Data Pair Buffer block whose data is to be stored.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector1</i>	The first input vector path value passed into the block. This is the X vector.
<i>input-vector2</i>	The second input vector path value passed into the block. This is the Y vector.
<i>output-dp</i>	The output data pair object produced by the block.

Return Value	Description
<u>truth-value</u>	Returns false if no data are passed into the data pair object. Otherwise returns true .

nol-clear-data-pair-buffer

(*blk*: class data-pair-buffer)

Clears the data pair buffer for later storage, using a Data Pair Buffer block. The stored X and Y vectors are reset to zero.

Parameter	Description
<i>blk</i>	The Data Pair Buffer block whose data is to be cleared.

nol-execute-data-pair-converter

(*blk*: class data-pair-converter, *mode*: symbol,
input-vector: class vector-path-value, *output-dp*: class data-pair)

Forms a data pair by dividing its input vector into X and Y vectors, using a Data Pair Converter block. The X Dimension attribute of the block specifies how many elements to place into the data pair's X vector. The rest of the elements are placed into the data pair's Y vector.

Parameter	Description
<i>blk</i>	The Data Pair Converter block whose data is to be converted.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector</i>	The input vector path value passed into the block.
<i>output-dp</i>	The output data pair object produced by the block.

nol-execute-data-pair-divider

(*blk*: class data-pair-divider, *mode*: symbol,
input-dp: class data-pair, *output-vector1*: class vector-path-value,
output-vector2: class vector-path-value)

Splits the input data pair into two separate vectors, using a Data Pair Divider block. The left output port is the data pair's X vector, and the right vector is the data pair's Y vector.

Parameter	Description
<i>blk</i>	The Data Pair Divider block whose data is to be divided.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-dp</i>	The input data pair passed into the block.
<i>output-vector1</i>	The first output vector path value produced by the block. This is the X vector.
<i>output-vector2</i>	The second output vector path value produced by the block. This is the Y vector.

nol-execute-data-pair-outlier-filter

(*blk*: class data-pair-outlier-filter, *mode*: symbol, *input-dp*: class data-pair)
-> *truth-value*

Tests the input data pairs whose elements do not fall within specified bounds, using a Data Pair Outlier Filter block. The procedure returns true if all data elements within the data pair are within the specified bounds; otherwise it returns false.

Parameter	Description
<i>blk</i>	The Data Pair Outlier Filter block whose data is to be filtered.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-dp</i>	The input data pair passed into the block.
Return Value	Description
<u><i>truth-value</i></u>	Returns false if any element of the data pair is out of the bounds specified in the block. Otherwise returns true.

nol-execute-data-set

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

Inserts the data contained in the input data pair into a new row at the end of the Data Set block. 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
<i>blk</i>	The Data Set block to which data is to be added.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .

Parameter	Description
<i>input-dp</i>	The input data pair that contains the row of data to be added to the data set.
<i>output-data</i>	An object that contains the number of rows in the data set.

nol-read-data-set-from-file

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

Populates the contents of a Data Set block with the contents of a file. If the procedure named by the File Load Procedure attribute of the Data Set block 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
<i>blk</i>	The Data Set block to populate.
<i>file-name</i>	The name of the file containing the data values, including a path appropriate for the file system type.

nol-write-data-set-to-file

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

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

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

nol-configure-data-set

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

Configures a Data Set block to a given specification.

Parameter	Description
<i>blk</i>	The Data Set block to be configured.
<i>number-of-samples</i>	The number of rows for the data set.
<i>width-of-input</i>	The number of input variables for the data set.
<i>width-of-output</i>	The number of output variables for the data set.

nol-clear-data-set

(*blk*: class data-set)

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

Parameter	Description
<i>blk</i>	The Data Set block to be cleared.

nol-execute-data-set-reader

(*blk*: class data-set-reader, *mode*: symbol, *input-ds*: class data-set, *output-dp*: class data-pair, *output-data*: class data-path-value)

Reads the contents of a data set and places the contents into the output data pair, using a Data Set Reader block. The procedure maintains the line pointer during and after each procedure call.

Parameter	Description
<i>blk</i>	The Data Set Reader block used to read data from the data set.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-ds</i>	The data set to read.
<i>output-dp</i>	The output data pair into which the reader reads the data.
<i>output-data</i>	The line pointer after the procedure call.

nol-execute-random-divider

(*blk*: class random-divider, *mode*: symbol, *obj-list*: class item-list, *ds1*: class data-set, *ds2*: class data-set)

Randomly copies all the data pairs from one or more data sets to two output data sets, using a Random Divider block. 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
<i>blk</i>	The Random Divider block used to read data from the data set.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>obj-list</i>	The list of data sets to read.
<i>ds1</i>	The first data set into which the data is copied.
<i>ds2</i>	The second data set into which the data is copied.

nol-execute-s-fold-divider

(*blk*: class s-fold-divider, *mode*: symbol, *obj-list1*: class item-list, *obj-list2*: class item-list)

Copies all the data pairs from one or more input data sets to one or more output data sets, using a S-Fold Divider block. The procedure randomly divides the input data between the output data sets.

Parameter	Description
<i>blk</i>	The S-Fold Divider block used to read data from the data sets.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>obj-list1</i>	The list of data sets to read.
<i>obj-list2</i>	The list of data sets into which the data is copied.

nol-execute-data-set-copier

(*blk*: class gdl-block, *mode*: symbol, *obj-list1*: class item-list ,
obj-list2: class item-list)

Copies all of the data sets in the first object list to each of the data sets in the second object list, using a Data Set Copier block.

Parameter	Description
<i>blk</i>	The Data Set Copier block used to read data from the data sets.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>obj-list1</i>	The list of data sets to read.
<i>obj-list2</i>	The list of data sets into which the data is copied.

nol-execute-data-set-rescaler

(*blk*: class data-set-rescaler, *mode*: symbol,
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 block.

Parameter	Description
<i>blk</i>	The Data Set Rescaler block used to read data from the data sets.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>obj-list1</i>	The list of data sets containing the data to be scaled.
<i>obj-list2</i>	The list of data sets into which the scaled data is copied.

nol-execute-maximum-age-filter

(*filter*: class maximum-age-filter, *ds*: class data-set, *obj-list*: class item-list,
output-data: class 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, using a Maximum Age Filter block.

Parameter	Description
<i>filter</i>	The Maximum Age Filter block whose data is to be filtered.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>ds</i>	The data sets to be filtered.
<i>obj-list</i>	The list of data sets used to archive the filtered data pairs.
<i>output-data</i>	The data set size after the procedure call.

nol-execute-size-limitation-filter

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

Limits the number of data pairs stored in the data set, using a Size Limitation Filter block. 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
<i>filter</i>	The Size Limitation Filter block to whose data is to be filtered.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>ds</i>	The data set to be filtered.
<i>obj-list</i>	The list of data sets used to archive the filtered data pairs.
<i>output-data</i>	The data set size after the procedure call.

nol-execute-novelty-filter

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

-> truth-value

Filters the data set, using a Novelty Filter block. You configure the filtering algorithm in the block. 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
<i>filter</i>	The Novelty Filter block that filters the data.
<i>ds</i>	The primary data set to be filtered. In a diagram, this would be the data set to which the capability link on the Novelty Filter block would attach.
<i>obj-list</i>	The list of data sets in which the removed data points are stored.
<i>output-data</i>	The new number of rows in the data set.
Return Value	Description
<u><i>truth-value</i></u>	Return true if the new data point is none.

Neural Networks

nol-execute-bpn

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

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

Parameter	Description
<i>net</i>	The Backpropagation Network block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: <code>system</code> or <code>manual</code> . Usually, you use <code>system</code> .
<i>x</i>	The input vector X for the network.
<i>y</i>	The output vector Y from the network.

nol-configure-bpn

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

Configures a Backpropagation Network block with a given specification.

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

nol-clear-bpn

(*net*: class bpn)

Clears a Backpropagation Network block of its weights.

Parameter	Description
<i>net</i>	The Backpropagation Network block to be cleared.

nol-write-bpn-to-file*(net: class bpn, stream: class g2-stream)*

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

Parameter	Description
<i>net</i>	The Backpropagation Network block whose parameters are to be saved.
<i>stream</i>	The G2 file stream into which the parameters are saved.

nol-read-bpn-from-file*(net: class bpn, stream: class g2-stream)*

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

Parameter	Description
<i>net</i>	The Backpropagation Network block whose parameters are to be read.
<i>stream</i>	The G2 file stream from which the parameters are read.

nol-execute-rbfn*(net: class rbfn, mode: symbol, x: class vector-path-value, y: class vector-path-value, maximum-activation-path: class data-path-value)*

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

Parameter	Description
<i>net</i>	The Radial Basis Function Network block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>x</i>	The input vector X for the network.
<i>y</i>	The output vector Y from the network.
<i>maximum-activation-path</i>	The maximum hidden node activation value.

nol-configure-rbf

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

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

Parameter	Description
<i>net</i>	The Radial Basis Function Network block to be configured.
<i>inputs</i>	The number of inputs.
<i>hidden</i>	The size of the hidden layer.
<i>outputs</i>	The number of outputs.
<i>overlap</i>	The type of overlap.
<i>unit-type</i>	The unit type.

nol-write-rbf-to-file

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

Save the specification and weights of a Radial Basis Function Network block into a file stream.

Parameter	Description
<i>net</i>	The Radial Basis Function Network block to be saved.
<i>stream</i>	The G2 file stream into which the data is saved.

nol-read-rbf-from-file

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

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

Parameter	Description
<i>net</i>	The Radial Basis Function Network block whose data is to be read.
<i>stream</i>	The G2 file stream to read.

nol-execute-rho-net

(*net*: class rho-net, *mode*: symbol, *x*: class vector-path-value, *y*: class vector-path-value)

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

Parameter	Description
<i>net</i>	The Rho Network block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>x</i>	The input vector X for the network.
<i>y</i>	The output vector Y from the network.

nol-execute-ensemble-model

(*net*: class ensemble-network , *mode*: symbol,
x: class vector-path-value, *y*: class vector-path-value)

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

Parameter	Description
<i>net</i>	The Ensemble Network block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>x</i>	The input vector X for the network.
<i>y</i>	The output vector Y from the network.

nol-read-ensemble-model-from-file

(*net*: class ensemble-network, *stream*: class g2-stream)

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

Parameter	Description
<i>net</i>	The Ensemble Network block whose data is to be read.
<i>stream</i>	The G2 file stream to read.

nol-write-ensemble-model-to-file

(*net*: class ensemble-network, *stream*: class g2-stream)

Save the specification of an Ensemble Network block into a file stream.

Parameter	Description
<i>net</i>	The Ensemble Network block to be saved.
<i>stream</i>	The G2 file stream into which the data is saved.

nol-execute-autoassociative-net

(*net*: class autoassociative-net, *mode*: symbol, *input-vector*: class vector-path-value, *output-vector*: class vector-path-value)

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

Parameter	Description
<i>net</i>	The Autoassociative Network block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-vector</i>	The input vector X for the network.
<i>output-vector</i>	The output vector Y from the network.

nol-write-autoassociative-net-to-file

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

Save the specification of an Autoassociative Network block into a file stream.

Parameter	Description
<i>net</i>	The Autoassociative Network block to be saved.
<i>stream</i>	The G2 file stream into which the data is saved.

nol-read-autoassociative-net-from-file

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

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

Parameter	Description
<i>net</i>	The Ensemble Network block to be loaded into.
<i>stream</i>	The g2 file stream.

nol-execute-trainer

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

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

Parameter	Description
<i>blk</i>	The Trainer block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>net</i>	The network block to be trained.
<i>ds</i>	The data set to use for training.
<i>output-data</i>	The RMSE error value after the network has been trained.

nol-execute-fit-tester

(*blk*: class fit-tester, *mode*: symbol, *net*: class neural-network, *ds*: class data-set, *output-data*: class data-path-value)

Uses a Fit Tester block to train a neural network, using a data set.

Parameter	Description
<i>blk</i>	The Fit Tester block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>net</i>	The network block to be tested.
<i>ds</i>	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.
<i>output-data</i>	A data path value into which the result of the fitness test is placed.

nol-execute-sensitivity-tester

(*blk*: class sensitivity-tester, *mode*: symbol, *net*: class neural-network, *ds*: class data-set)

Uses a Sensitivity Tester block to calculate the sensitivity of a neural network, given a data set.

Parameter	Description
<i>blk</i>	The Sensitivity Tester block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>net</i>	The network block to be tested.
<i>ds</i>	The data set to be tested.

Action Utilities

nol-execute-block-attribute-transfer

(*blk*: class block-attribute-transfer, *mode*: symbol, *input-obj*: class object, *obj-list*: class item-list)

Uses the Block Attribute Transfer block to copy one block's attribute values to another block.

Parameter	Description
<i>blk</i>	The Block Attribute Transfer block to be executed.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .
<i>input-obj</i>	The block from which to copy attributes.
<i>obj-list</i>	The list of blocks into which to copy the attributes.

nol-execute-block-make-permanent

(*blk*: class block-make-permanent, *mode*: symbol, *obj-list*: class item-list, *client-obj*: class object)

Uses the Makes Permanent block to make a list of objects permanent. This procedure can invoke the G2 make permanent action, or it can make permanent any array or matrix data associated with the object. The NOL blocks that can have permanent data are: Novelty Filter, Data Set, Data Pair Outlier Filter, Vector Order Swapper, Vector Rescaler, Data Set Rescaler, Vector Path Entry Point, Data Set Plot, Sensitivity Tester, and any Neural Network block. If an error occurs during the execution of this procedure, the error is posted to the error queue.

Parameter	Description
<i>blk</i>	The Make Permanent block to be executed. To make the API call, an instance must be supplied; however it has no affect on the execution of this procedure.
<i>mode</i>	The mode of execution, which can be one of two symbols: system or manual . Usually, you use system .

Parameter	Description
<i>obj-list</i>	The list of objects to be made permanent. If an object in the list is a subclass of <code>nol-block-with-permanent-array</code> , then the procedure makes permanent the NOL data associated with the block. If the object is any other class of object, the procedure calls the G2 <code>make permanent</code> action on the object. Non-objects in the list are ignored. The procedure must manage the creation and deletion of this list; otherwise, the procedure can leak items.
<i>client-obj</i>	A G2 window or client object to be used for posting errors. If no window is available, use the default window <code>gfr-default-window</code> .

nol-execute-block-restore-permanent-values

(*blk*: class `block-restore-permanent-values`, *mode*: symbol, *obj-list*: class `item-list`, *client-obj*: class `object`)

Uses the Restore Permanent Values block to restore the permanent values of a list of blocks.

Parameter	Description
<i>blk</i>	The Restore Permanent Values block to be executed. To make the API call, an instance must be supplied; however, it has no affect on the execution of this procedure.
<i>mode</i>	The mode of execution, which can be one of two symbols: <code>system</code> or <code>manual</code> . Usually, you use <code>system</code> .
<i>obj-list</i>	The list of objects whose values are to be restored. If an object in the list is a subclass of <code>nol-block-with-permanent-array</code> , then the procedure restores the NOL data associated with the block. If the list contains objects that are not of this class, the objects are ignored. The procedure must manage the creation and deletion of this list; otherwise, the procedure can leak items.
<i>client-obj</i>	A G2 window or client object to be used for posting errors. If no window is available, use the default window <code>gfr-default-window</code> .

File Operations

nol-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
<i>stream</i>	The G2 file stream to read.
<i>number-of-values</i>	The number of values to read.
<i>destination-array</i>	The G2 array in which to store the values.
<i>delimiter</i>	The delimiter string.

nol-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
<i>stream</i>	The G2 file stream to write.
<i>array</i>	The G2 array from which to write the values.
<i>delimiter</i>	The delimiter string.
<i>comment</i>	Comment text to write with the file stream.

nol-read-matrix

(*stream*: class g2-stream, *x*: class 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
<i>stream</i>	The G2 file stream to read.
<i>x</i>	The matrix in which to store the values.

Parameter	Description
<i>rows</i>	The number of rows for the two-dimensional array.
<i>cols</i>	The number of column for the two-dimensional array.
<i>delimiter</i>	The delimiter string.

nol-write-matrix

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

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

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

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

A

action block: Performs actions on other blocks or on the environment when an inference value becomes true.

action link: Serves as a pointer when a block is to perform an action on a target object. For example, you can reset a block by using a control signal by attaching any block to the action link associated with the Reset block. When the block receives a control signal, NOL resets the block. Action links appear on many blocks on the Action Utilities palette.

API procedure: Enables you to control NeurOn-Line objects from within a G2 procedure or function. The Application Programmer's Interface (API) includes procedures that perform the same actions as executing NeurOn-line blocks.

attributes: Specify the particular behavior of a block. You specify block attributes in the configuration panel.

B

block menu: Enables you to perform G2 operations on blocks, such as cloning, transferring, deleting, and configuring.

C

capability link: Adds various types of features to blocks, such as charts, graphs, and clocks.

configuration panel: A dialog that enables you to specify the attributes of a block.

configure: To specify the attributes of a block in a configuration panel or in the attribute display of a block.

connection post: Enables a block on one workspace to pass data to a block on another workspace.

control path: A type of path that passes control signals, which cause downstream blocks to execute.

D

data block: Operates on numeric values.

data pair path: Carries Data Pairs, which contain two vectors: the input vector and the target vector. Generally you add Data Pairs to a Data Set and use the Data Set to train and test a neural network. The input vector represents the input data for a neural network, and the target vector represents the data that the neural network should output.

Data pair paths have the attributes **Collection-time**, **Quality**, and **Timestamp**.

data path: An input or output path to a block that contains numeric values.

data set blocks: Enable you to store and manipulate the data with which you train a neural network. The Data Set Blocks palette under the Data Processing submenu of the Palettes lists the various data set blocks.

developer mode: A user mode that provides access to all the basic functionality required for building schematic diagrams.

disable evaluation: Stops a block from passing its output value and responding to new values. Disabling evaluation enables you to “turn off” entire portions of a NOL diagram.

discrete logic: A form of inferencing whereby a block passes discrete inference values, for example, a **Status-value** of `.true`, `.false`, or `unknown`, and a **Belief-value** of 1.0, 0.0, or 0.5.

E

enable data input: To run a diagram, you must enable data input, which:

- Starts data flowing into entry points.
- Evaluates signal generators.
- Evaluates clock capabilities.

enable evaluation: Allows a block to pass its output value and respond to new values.

entry point: Receives data externally from a variable, a GSI (G2 Standard Interface) variable, G2 procedure, or from an embedded variable in the table for the block. Entry points are the starting point of a NOL diagram. NOL supports four kinds of entry points: numeric, belief, control, and vector.

evaluate: To execute the procedure for a block. For example, when you evaluate an entry point, the current value is propagated with a new timestamp. When you evaluate a block with a single input control path, the block acts as if it has received a new control signal.

F

filter: A category of block that you use after data entry blocks in a diagram to filter out noise and find trends in data.

G

G2 Main Menu: Controls whether G2 is running or paused, and allows you to load and save applications.

G2 menus: Provide all the functionality of G2 within the NOL environment, for example, the creation of class definitions, variables, parameters, rules, and procedures.

H

history: A store of past input values. Numerous NOL blocks operate on the stored values, such as computing the average of the last 25 input values.

HTML: Hypertext Markup Language. Online documentation is a collection of HTML files, which you can display in any HTML browser.

I

inference block: Translates data values to truth values and operates on truth values. For example, observations observe data values and pass inference values, and logic gates use Boolean logic to combine reference values.

inference path: A type of path that carries truth values. Inference paths carry two values:

- Belief value - a number between 0.0 to 1.0, where 0.0 is false and 1.0 is true.
- Status value - one of the symbols `.true`, `.false`, or `unknown`. NOL derives status values from belief values.

initial value: The value a block passes when you first start G2 or when you reset the block.

input port: Carries data to a block. A block has one or more input ports depending on the type of block.

invoke: To execute the procedure for a block. The block is invoked when:

- An entry point receives a value from its data source.
- A value is propagated onto the input path of the block due to the behavior of an upstream block.
- You evaluate a block manually.

K

KB Workspace menu: Enables you to create G2 definitions and objects on a workspace and set up applications.

L

link: A special-purpose type of connection that you use to add features or behaviors to a block, or to perform actions on a block. For example, you use links to add a graph or chart capability to a block.

There are two types of links:

- Action
- Capability

lock: When locked, a block does not respond to input data or pass its output values. NOL locks a block when you manually override its value.

M

multiple values: Simultaneous control signals that a block receives. A block can ignore or use multiple values as needed.

neural network blocks: Enable you to save and load what the blocks have learned during training. NeurOn-Line saves the information to text files, which you can examine. The Neural Networks palette under the Neural Networks submenu of the Palettes menu lists the four neural network blocks:

- Backpropagation Net (BPN)
- Autoassociative Net
- Radial Basis Function Net (RBFN)
- Rho Net (Density Estimation)

N

no-value inputs: Occur when stubs are unattached or when connected paths never receive a value. Peer input data blocks and peer input logic blocks ignore input paths with a Quality of no-value. Non-peer input blocks require all of their inputs to evaluate, and, therefore, never place a value onto an output path if the block has a no-value input.

O

observation: A category of blocks that detect features in your data. Observation blocks take data as input, test it against a threshold, and pass as output the inference value that the test produced.

output port: Carries data from a block. A block has one or more output ports depending on the type of block.

override: To manually change a block's output value for testing purposes. Overriding a block locks the block and propagates a **Quality of manual** onto the output path.

P

parameter: A G2 object that stores a data value and keeps a history of it over a specified time. A parameter can also initiate forward chaining.

path: The connection between two blocks. NOL supports data, inference, control, vector, and data pair paths.

path attributes: Attributes that provide information about the value and status of the data on one path. Each type of path defines slightly different path attributes.

path quality: A path attribute that specifies the status of a path's data. There are three types:

- Manual
- No-value
- Expired

path splitter: Connects the input stub from one block to the path between two other blocks so that more than one downstream block can get input from the same upstream block.

peer input block: A category of block that can have any number of inputs and does not evaluate the inputs in a specific order. The inputs are treated all alike and are therefore peers.

port: Connection stub to which another stub can attach. Ports are either named or unnamed, depending on the type of block and whether the port is input or output.

Q

queue: A special workspace that displays information about errors.

remote process: A concurrent process that is separate from G2, the NeurOn-Line uses for numerically intensive tasks such as neural network training, fit testing, sensitivity testing, and running autoassociative networks.

R

reset: Causes the following to happen:

- Sets the block to its initial state, which propagates the block's initial value.
- Clears any error conditions.
- Unlocks the block, if it was locked.
- Erases the blocks history if the block maintains a history.

S

scalar blocks: Enable you to perform operations on scalar values, such as arithmetic, averaging, filtering, delaying, and applying your own functions. The Scalar Blocks palette under the Data Processing submenu of the Palettes menu lists the various scalar blocks.

signal generator: Generates a continuous signal to a diagram, to simulate real-time data. Examples of signal generators include the Sine Wave signal and White Noise signal.

snapshot: A file that contains the current state of the running application as backup. You can configure NOL to take snapshots automatically at regular intervals. When you restore a snapshot, NOL resumes running the application from the point at which you took the snapshot.

spreadsheet editor: Many blocks that operate on vectors, Data Pairs, and Data Sets let you edit data by using the Gensym Spreadsheet System or GXL. GXL is a G2 module that provides spreadsheet editing capabilities for editing vector and data set blocks.

stub: A connection port to which another connection port can attach.

sweep: An internal mechanism where NOL searches for invoked blocks, evaluates them, and continues until no more blocks are left to evaluate.

system administrator: A category of NOL users who works in Administrator mode, which enables access to additional attributes and menu choices used for debugging.

T

top menu bar: Provides access to basic functionality, including controlling the diagram, cloning blocks from palettes, accessing queues, and customizing the environment.

training block: Trains a neural network when you attach one of its action links to a neural network block and the other action link to a data set block. Training blocks train the following neural networks:

- Backpropagation
- Autoassociative
- Radial Basis Function Network
- Rho Network

U

uncertainty: Defines a band around 0.5 that determines the status value unknown. For example, if the attribute Output Uncertainty is 0.25, then the Belief-value is .true above 0.625 and .false below 0.375 and Unknown between .65 and .35.

user modes: There are four user modes:

- Administrator - enables system administrators to access attributes and menu choices used for debugging.
- Developer - enables developers to access all the basic functionality used for building schematic diagrams.
- User and Browser - enables end users to view diagrams, display menus, and display configuration panels of objects but not to move blocks, clone blocks, or edit attributes. Browser mode is slightly more restrictive than User mode.

V

variable: You use variables as starting points in a NOL diagram, either by referring to the variable in an entry point, or by connecting blocks directly to the variable. *See also* parameter.

vector blocks: Enable you to create, manipulate, and operate on vectors. The Vector Blocks palette under the Data Processing submenu lists the various vector blocks.

vector path: Carries vectors, which are one-dimensional arrays of numeric data. Vectors can be of any size. Vector paths have the attributes Collection-time and Quality.

vertex: An 90° bend in the connection between blocks.

@	A	B	C	D	E	F	G	H	I	J	K	L	M
#	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

A

- abs function
 - using with data values
- Action Utilities palette
- actions
 - API procedures for
 - branching
 - introduction to
 - invoking a rule
 - looping
 - outputting data
 - performing on blocks
 - stopping paths
- adding noise
- Additive Noise block
- Allow Intermediate Evaluation attribute
 - Clock
- Amplitude attribute
 - Sine Wave
- AND Gate
- Application Programmer? Interface (API)
 - accessing
 - action utilities
 - data set blocks
 - file operations
 - introduction
 - neural networks
 - path displays
 - vector blocks
- applications
 - connecting to G2
 - using Data Output block
 - using entry points
- arctan function
 - using with data values
- Arithmetic Function attribute
 - Arithmetic Function
- Arithmetic Function block
- arithmetic operations
 - adding
 - constants
 - inputs
 - changing sign
 - computing inverse
 - dividing
 - multiplying
 - by constants
 - inputs
 - performing
 - subtracting
- Attribute Transfer block
- Attribute, Set
- attributes, transferring
- Autoassociative Net block
- average
 - of a history
 - of input values
- Average Input Value block
- Average, Moving
- averaging values

- B**
- Backpropagation Net (BPN) block
- Band Center attribute
 - Outlier Filter
- Band Range attribute
 - Outlier Filter
- Band Type attribute
 - Outlier Filter
- Belief Entry Point
- Bias attribute
 - Bias
 - Sine Wave
- Bias block
- blocks
 - evaluating
 - using Evaluate block
 - overriding
 - using Set Attribute block
 - resetting
 - using Reset block
- branching
- Breakers, Circuit
- buttons
 - specifying entry point data, using

C

- capabilities
 - charting
 - evaluating blocks at specific time intervals
 - forcing a block to evaluate
 - initiating control actions
 - introduction to
 - plotting data
 - starting control paths
- Capabilities palette
- ceiling function
 - using with data values
- Change Sign block
- Chart Attribute attribute
 - Chart Capability
- Chart Capability
- Chart Name attribute
 - Chart Capability
- charting
- charts
 - configuring
 - creating
 - determining how data is displayed in
- Circuit Breakers
- Classifier Input Converter block
- Classifier Output Converter block
- classifiers, using with vectors
- Clear block
- Clock Capability
- clocks
 - evaluating blocks at specified time intervals, using
- close breaker menu choice
- computing statistical properties
- Conclusion block
- configure menu choice
 - for charts
- Connection Posts
- connections
- Connections palette
- Connectors
- Control Counter block
- Control Entry Point
- Control Inhibit block
- Control Initiation Capability
- Control Path Loop block
- control signals
 - choosing paths for
 - based on inference value
 - converting to inference values

- counting
- evaluating blocks
- generating
- initiating when blocks receive values
- looping
- resetting blocks
- starting
- stopping
- Control Switch block
- converting
 - inference values from control signals
- cos function
 - using with data values
- Counter, Control
- cp-out attribute
 - reading and displaying
- customer support services
- cycles, unmanaged

D

- Data Delay block
- Data Inhibit block
- Data Output block
- Data Pair Buffer block
- Data Pair Converter block
- Data Pair Divider block
- Data Pair Outlier Filter block
- Data Pair Quality Filter block
- Data Pair Random Gate block
- data pairs, creating
- Data Path Display block
- Data Set block
- Data Set Blocks palette
- Data Set Copier block
- Data Set Plot block
- Data Set Reader block
- Data Set Rescaler block
- data sets
 - adding data pairs to
 - API procedures for
 - choosing which points to keep
 - clearing
 - copying data to
 - customizing text format of
 - deciding whether a data pair is novel
 - editing
 - entering and viewing data
 - filtering data from
 - introduction to

- making values permanent
- plotting data
 - choosing how to display the data
 - choosing what to display
 - choosing where to display the data
 - creating and deleting data series
 - how to
 - reading data from
 - saving and loading data
 - scaling data in
 - text format of
- Data Shift block
- Data Source attribute
- data sources
 - of entry points
 - embedded
 - external
 - external datapoints
- data values
 - displaying, using path display
 - generating
 - external
 - plotting
- Delay, Data
- descriptions
 - Conclusion
 - Entry Points
 - Equality
 - High and Low Value
 - Inference Output
- deviation
 - computing
 - using Variance
- Difference block
- Display, Data Path
- displays, path
- Distributed Control Systems (DCS)
- Disturbance Mean attribute
 - White Noise
- Disturbance Variance attribute
 - White Noise
- Dp-out attribute
 - configuring for entry points

E

- Ensemble Net (ENN) block
- entry points
 - choosing data source for
 - enabling data input for

- obtaining data from
 - external datapoints
 - variables
 - reading output values for
 - specifying embedded variables for
 - vector
- Entry Points palette
- Equality block
- Equivalence Band attribute
 - Equality
- Erase History When Reset attribute
 - Chart Capability
 - Data Shift
 - Moving Average
 - Sample Median
 - Variance
- Evaluate block
- evaluating blocks
 - at specified time intervals
 - using Evaluate block
- Evaluation Period attribute
 - Clock
- Exit If attribute
 - Control Path Loop
- exp function
 - using with data values
- exponential filters
 - first-order
- external data
 - connecting to
 - using Data Output block
 - using entry points

F

- file operations, API procedures for
- Filter Constant attribute
 - First-Order Exponential Filter
- filtering noise
- filters
 - low-pass
 - exponential
 - outlier
- First-Order Exponential Filter
- Fit Tester block
- Five Fold CV block
- floor function
 - using with data values
- forcing blocks to evaluate
- formulas, specifying entry points, using

functions
 average
 defining your own
 median
 using in blocks

G

Gain attribute
 Gain
Gain block
go to chart menu choice
go to sensor menu choice

H

High Value block
Hysteresis When attribute
 Conclusion
 Equality
 High and Low Value

I

Indicator Shape attribute
 Chart Capability
Indicator Visible attribute
 Chart Capability
Inference Blocks palette
Inference Inhibit block
Inference Output block
inference values
 converting
 from control signals
 generating
 inhibiting
inferencing
 introduction to
 making observations
 pausing paths
 performing logical operations
Inhibit, Control
Inhibit, Data
Inhibit, Inference
int function
 using with data values
Inverse block
inverting
 data values
lp-out attribute

 reading and displaying
Iteration Limit attribute
 Control Path Loop

L

Line Color attribute
 Chart Capability
ln function
 using with data values
log function
 using with data values
Logic attribute
 AND Gate
 entry points
 NOT Gate
 OR Gate
logical operations
 and
 not
 or
logical operations, performing
looping actions
loops, creating
Low Value block
low-pass filters
 exponential

M

Make Permanent block
Maximum Age Filter block
Maximum Unknown Inputs attribute
 AND Gate
 OR Gate
May Cause Chart Updating attribute
 Chart Capability
mean
 generating values around
 white noise
median
 of a history of data values
 of input values
Median Input Value block
Moving Average block
Multiple Invocations attribute
 Control Counter
Multiplication block

N

- Name attribute
 - Connection Posts
- Name of Sensor attribute
 - entry points
- neural networks
 - adjusting weights
 - API procedures for
 - autoassociative networks
 - backpropagation networks
 - choosing the run mode
 - ensemble networks
 - introduction to
 - radial basis function networks
 - rho networks
 - saving and loading network weights
 - saving and loading weights
- Neural Networks palette
- New Display menu choice
- noise
 - adding
 - filtering
- Noise, White
- NOT Gate
- no-value inputs, AND gate and Maximum
 - Unknown Inputs
- Novelty Filter block
- Numeric Entry Point

O

- observations
 - equality
 - high and low
 - values
- open breaker menu choice
- OR Gate
- Outlier Filter
- Outlier Replacement attribute
 - Outlier Filter
- Output as Std Deviation attribute
 - Variance
- Output Uncertainty attribute
 - AND Gate
 - Conclusion
 - entry points
 - Equality
 - High and Low Value
 - Inference Output
 - NOT Gate

- OR Gate
- Output, Data
- Output, Inference
- outputting data

P

- parameters
 - putting data into, using block
- path
 - API procedures for displays
- path display blocks
- Path Displays palette
- path splitters
- paths
 - combining
 - connecting
 - using circuit breakers
 - using connectors
 - stopping
- pausing data
- Period attribute
 - Sine Wave
- Phase Angle attribute
 - Sine Wave
- Plot Mode attribute
 - Chart Capability
- plotting
 - data values
 - in data sets
 - using charts
- Posts, Connection
- procedures
 - specifying entry point data, using
 - using
 - in blocks

Q

- Quantization attribute
 - First-Order Exponential Filter
 - Outlier Filter
- Quotient block

R

- Radial Basis Function Net (RBFN) block
- Random Divider block
- random function
 - using with data values

- ranges
 - filtering values within
 - outlier filter
- RBFN block
- Reference Value attribute
 - Equality
- Require Full History attribute
 - Moving Average
 - Sample Median
 - Variance
- Reset block
- Reset Phase attribute
 - Sine Wave
- resetting
 - blocks
 - using Reset block
- Restore Permanent Values block
- Rho Net block
- Rule Action block
- rules
 - invoking
 - specifying entry point data, using

S

- Sample Median block
- Sample Period attribute
 - Sine Wave
 - specifying how often to generate values,
 - using
 - White Noise
- Sample Size attribute
 - Chart Capability
 - Data Shift
 - Moving Average
 - Sample Median
 - Variance
- Sample Type attribute
 - Chart Capability
 - Moving Average
 - Sample Median
 - Variance
- scalar blocks
- Scalar Blocks palette
- Scalarizer block
- scalars, creating
- Sensitivity Tester block
- Set Attribute block
- S-Fold Divider block
- Shift, Data

- show collection time menu choice
- show quality menu choice
- show value menu choice
- signal generators
 - periodic
 - sine wave
 - random
 - white noise
 - specifying how often to generate values
- sin function
 - using with data values
- Sine Wave block
- Size Limitation Filter block
- sqrt function
 - using with data values
- standard deviation
- Standard Deviation attribute
 - Additive Noise block
- statistical properties, computing
- Status on Initialization attribute
 - Conclusion
 - entry points
 - Equality
 - High and Low Value
 - Inference Inhibit
 - Inference Output
 - Inference Path Circuit Breaker
- stopping data
- Summation block
- Superior-connection attribute
 - in Connection Posts
- Switch, Control
- Symbolic Entry Point
- symbolic values, generating

T

- tabular-function-of-1-arg function
 - using with data values
- tan function
 - using with data values
- Target Attribute attribute
 - Set Attribute
- Target Variable attribute
 - Data Output
- text
 - values, generating
- Text Entry Point
- Threshold attribute
 - High and Low Value

- Threshold Uncertainty attribute
 - High and Low Value
- time
 - evaluating blocks at specified intervals
- Train and Test block
 - description
 - subworkspace
- Trainer block
- training
 - and testing
 - basic
 - choosing
 - maximum number of iterations
 - training method
 - whether to accelerate
 - configuring
 - for Radial Basis Function network
 - for Rho network
 - finding
 - the best network configuration
 - which inputs are significant
 - introduction to
 - watching the training happen
- Training Blocks palette
- transferring attributes
- Trigger On attribute
 - Control Inhibit
 - Data Inhibit
 - Inference Inhibit
- truncate function
 - using with data values

U

- Update Size attribute
 - Chart Capability
 - Moving Average
 - Sample Median
 - Variance
- Update Type attribute
 - Chart Capability
 - Moving Average
 - Sample Median
 - Variance
- Use Expired Inputs attribute
 - AND Gate
 - Average Input Value
 - Median Input Value
 - Multiplication
 - OR Gate

Summation

V

- Validity Interval attribute
 - entry points
- Value on Initialization attribute
 - Control Counter
 - Data Inhibit
 - Data Path Circuit Breaker
 - Data Shift
 - entry points
 - Moving Average
 - Sample Median
 - Variance
- variables
 - obtaining data from
 - external
 - using entry points
 - putting data into, using block
 - specifying entry point data, using your own
 - viewing data source for entry points
- Variance block
- Vector Blocks palette
- Vector Combiner block
- Vector Difference block
- Vector Function block
- Vector Function of Two Arguments block
- Vector Inhibit block
- Vector Order Swapper block
- Vector Path Display block
- Vector Product block
- Vector Quotient block
- Vector Rescaler block
- Vector Splitter block
- Vector Sum block
- vector values
 - displaying, using path display
- Vectorizer block
- vectors
 - API procedures for
 - blocks for
 - choosing when to evaluate blocks
 - creating
 - inhibiting
 - manipulating
 - operating on elements of
 - using with classifiers
- Vpv-out attribute

reading and displaying

W

Wave, Sine

White Noise block

Windower block