Custom Solution Wizard

Program version:	**4.00**
Help version:	**4.00**
Help Release Date:	**01/18/01**
Download the latest release from:	www.nd.com/support/help.htm

The Custom Solution Wizard is a tool that will take an existing neural network created with NeuroSolutions and automatically generate and compile a Dynamic Link Library (DLL). This allows you to easily incorporate neural network models into your own applications and into other NeuroDimension products, such as TradingSolutions (Developers level only).

While using the wizard to create the DLL, you are also given the option of creating a shell for any of the following programming environments:

- Visual Basic®
- Visual C++®
- Microsoft Excel®
- Microsoft Access®
- Active Server Pages (Developers level only)

Each shell provides a sample application along with source code to give you a starting point for integrating the generated DLL into your application.

The generated neural network DLL provides a simple protocol for assigning the network input and producing the corresponding network output. Furthermore, the Developers level of the Custom Solution Wizard supports learning. This allows you to train the generated neural network and/or retune the network after gathering new data. Embedding a custom neural network into your application could not be any easier!

There are five levels of the Custom Solution Wizard, each providing different capabilities as discussed below.

Educator

The Educator level of the Custom Solution Wizard can generate **recall** dynamic link libraries (DLLs) for neural networks that fall within the Educator level restrictions of NeuroSolutions.

Users

The Users level of the Custom Solution Wizard can generate **recall** DLLs for neural networks that fall within the Users level restrictions of NeuroSolutions.

Consultants

The Consultants level of the Custom Solution Wizard can generate **recall** DLLs for neural networks that fall within the Consultants level restrictions of NeuroSolutions.

Developers Lite

The Developers Lite level of the Custom Solution Wizard can generate **recall** DLLs for neural networks built with any level of NeuroSolutions. This level specifically adds the ability to generate DLLs for networks that include embedded user-defined DLLs (Note: Networks with embedded user-defined DLLs can be created with the Developers Lite or Developers levels of NeuroSolutions).

Developers

The Developers level of the Custom Solution Wizard can generate **recall** and **learning** DLLs for neural networks built with any level of NeuroSolutions. Learning DLLs can be used in your custom application to retune the neural network weights when new data becomes available.

The latest pricing for all NeuroDimension products can be found on our web site at: www.nd.com/pricing.htm.

Note: If you do not have access to the internet, see the Contacting NeuroDimension topic for information on how to contact NeuroDimension via phone, fax and mail.

NeuroDimension products can be ordered using any of the following 3 methods:

1. Place the order on-line using our SECURE order entry system.
2. Download and print the latest order form then fax the completed form to 352-377-9009, or mail it to:

   NeuroDimension, Inc.
   Order Processing Department
   1800 N. Main Street, Suite #D4
   Gainesville, FL 32609-8606

3. Phone in your order (800-634-3327 or 352-377-5144) Monday through Friday between the hours of 8:30 AM and 5:00 PM EST.

We accept payment by credit card (Visa, MasterCard or American Express), wire transfer, or prepayment by check or money order.

Note: In order to use the links within this topic, you must be connected to the internet.

NeuroDimension, Inc.
1800 N. Main Street, Suite D4
Gainesville, FL 32609
www.nd.com/

Sales and Information

Sales	**1-800-634-3327 (Option 3)**
Product Literature and Evaluation Software	**1-800-634-3327 (Option 2)**
Fax	**352-377-9009**
Email	**info@nd.com**
Calls Outside U.S.	**352-377-5144**

Technical Support

Bug Reports, Installation Problems, and Priority Support	**1-800-634-3327 (Option 0)**
All Other Technical Support	**352-377-1542**
Fax	**352-377-9009**
Email	**csw_support@nd.com**
Calls Outside U.S.	**352-377-1542**

Before contacting technical support, please attempt to answer any questions by first consulting the following resources:

- The printed manual (if applicable)
- The on-line help
- The Frequently Asked Questions (FAQ)

The latest versions of the on-line help and FAQ can always be found at the NeuroDimension web site: www.nd.com

Included with Evaluation/Demo Software

§ Toll-free line for bug reports and installation problems

Included with Purchased Software

§ Toll-free line for bug reports and installation problems
§ 1 year unlimited email, fax and phone support (toll line)

Contact Information

Bug Reports, Installation Problems and Priority Support	**1-800-634-3327 (Option 0)**
All other Technical Support	**(352) 377-1542**
Fax	**(352) 377-9009**
Email	**csw_support@nd.com**
Calls Outside U.S.	**(352) 377-1542**

Please have your invoice number ready when calling and include your invoice number in all fax, email, or written correspondence.

Note that the technical support described above is for questions regarding the software package. Neural network experts are on staff and available for consulting on an hourly basis. Consulting rates are dependent on the specifics of the problem.

NeuroSolutions

NeuroSolutions is the most powerful and flexible neural network simulator available for the Windows market. This leading edge software combines a modular, icon-based network design interface with advanced learning procedures, such as recurrent backpropagation and backpropagation through time. NeuroSolutions can be used to solve real-world problems such as financial forecasting, speech recognition, process control, and many more. Notable features include C++ code generation, user-defined algorithms and integration with Excel.

NeuroSolutions for Excel

NeuroSolutions for Excel is a revolutionary product that allows you access to all the powerful features of NeuroSolutions, but with the convenience and ease of use of a spreadsheet program. This Microsoft Excel add-in gives you the ability to visually tag your data as Training, Cross Validation, Testing, or Production, train a neural network, and test the neural network's performance directly from within a Microsoft Excel worksheet. Reports are automatically generated showing the results. Working with neural networks could not be any easier.

NeuroSolutions for Excel is not just for the novice, however. There are also powerful built-in features that help you to find the optimum neural network for your problem. These include the ability to train a neural network multiple times, vary any neural network parameters across multiple runs, genetically optimize network parameters, and create your own custom batches.

Neural and Adaptive Systems: Fundamentals Through Simulations

This interactive electronic book published by John Wiley and Sons combines the hypertext and searching capabilities of the Windows help system with the highly graphical simulation environment of NeuroSolutions to produce a revolutionary learning tool. The book contains over 200 interactive experiments in NeuroSolutions to elucidate the fundamentals of neural networks and adaptive systems.

TradingSolutions

TradingSolutions is a comprehensive financial analysis software package that helps you make better trading decisions. It combines traditional technical analysis with state-of-the-art artificial intelligence technologies. Now you can use any combination of financial indicators in conjunction with advanced neural networks and genetic algorithms to create models that are remarkably effective, especially in today's volatile markets.

TradingSolutions' user-friendly interface allows you to perform complex financial forecasting without leaving you lost in the technology. Simple wizards guide you step-by-step through each task, while optional advanced panels give you the flexibility to adjust parameters behind the scenes.

Genetic Server

Genetic Server is a general-purpose genetic algorithm ActiveX component that can be used to easily build custom genetic applications in Visual Basic (or any other environment that supports Active-X components). This software provides two core genetic algorithms: Generational and Steady State. Furthermore, the user can select between several types of selection, crossover, and mutation. Floating point and integer data types are supported in addition to the traditional binary data type. Genetic Server also provides several methods for terminating the genetic search.

Genetic Library

Genetic Library is a general-purpose genetic algorithm library written in ANSI C++. The library provides two core genetic algorithms: Generational and Steady State. Furthermore, the user can select between several types of selection, crossover, and mutation, or even write their own. Floating point and integer data types are supported in addition to the traditional binary data type. The library also provides several methods for terminating the genetic search.

Consulting

Neural network and genetic algorithm experts are on staff and available for consulting on an hourly basis. Consulting rates are dependent upon the specifics of the problem. To obtain an estimate for consulting, please email your problem specifics to info@nd.com.

Priority Support Package

The priority support package is a valuable service that can be added to the purchase of any NeuroDimension

product anytime within the first 30 days of product purchase. This service provides preferred status on all support issues as well as the following benefits:

§ Toll-free line for all calls
§ One FREE major upgrade
§ FREE minor upgrades

The cost for the priority support package is 30% of the purchase price of the product to be covered.

In order to run the Custom Solution Wizard, you must meet the following system requirements:

1. Windows 95/98/Me/NT4/2000.
2. NeuroSolutions 4
3. Visual C++ 5.0 or 6.0

Since the Custom Solution Wizard requires NeuroSolutions, its installation is included as part of the NeuroSolutions installation. During the NeuroSolutions installation, choose the **Typical** or the **Custom** option in order to make sure the Custom Solution Wizard gets installed. The Custom Solution Wizard installation will install the following files onto your hard drive:

File	Location	Description
CustomSolutionWizard.exe	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard	Custom Solution Wizard Main Executable
CustomSolutionWizard.hlp	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard	Custom Solution Wizard Help
CustomSolutionWizard.cnt	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard	Table of Contents for the Custom Solution Wizard Help
Xor.asc	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\Demo	Exclusive-Or data used by the Custom Solution Wizard demo
DemoMLP.dll	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\Demo	Pre-existing MLP dll used by the demo if the user does not have Visual C++ 5.0 or 6.0
DemoMLP.nsw	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\Demo	Pre-existing MLP weights file used by the demo if the user does not have Visual C++ 5.0 or 6.0
Train.bas	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\Demo	Sample of the code used by the Train procedure in the demo
GetResponse.bas	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\Demo	Sample of the code used by the GetResponse procedure in the demo
MSAccess8Shell.mdb, MSAccess9Shell.mdb	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\ProjectShells	Microsoft Access 97/2000 project shells
MSExcel8Shell.xls, MSExcel9Shell.xls	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\ProjectShells	Microsoft Excel 97/2000 project shells
VB5Shell.vbp, VB6Shell.vbp ,VBShell.bas, VBShellDlg.frm, VBShellDlg.frx, Schema.ini	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\ProjectShells	Visual Basic 5.0/6.0 project shell files
VCPP5Shell.dsw, VCPP5Shell.dsp, VCPP6Shell.dsw, VCPP6Shell.dsp, VCPPShell.clw, VCPPShell.cpp, VCPPShell.h, VCPPShell.ico, VCPPShell.rc, VCPPShell.rc2, NSNetwork.cpp, NSNetwork.h, resource.h, StdAfx.cpp, StdAfx.h	C:\Program Files\NeuroSolutions 4\Wizards \CustomSolutionWizard\ProjectShells	Visual C++ 5.0/6.0 project shell files
NeuroSolutionsOL.dll	C:\Windows\System or C:\Winnt\System32	NeuroSolutions 4.00 Object Library

The file locations shown are the default locations and may vary depending upon the options chosen during installation and/or the location of your Windows/Windows NT directory.

Below is a list of the limitations/restrictions of the Custom Solution Wizard:

1. The NeuroSolutions breadboard that the Custom Solution Wizard uses to generate a DLL must have a single input source and a single output destination. Furthermore, the input must be injected into the *Pre-Activity* access point of an *Axon* component and the output must be retrieved from the *Activity* access point on an *Axon* component. If these conditions are not met, the generated DLL may produce erroneous results.
2. Only the Developers level of the Custom Solution Wizard can generate learning DLLs. All other levels can only generate recall DLLs.
3. The following NeuroSolutions components are not supported by the Custom Solution Wizard and will automatically be removed before DLL generation:
   - § DataStorageTransmitters
   - § Input palette components
   - § Probes palette components (except for DLLPostProcessor)
   - § Dialog palette components
4. DLLs created with the Developers Lite or Developers Levels of NeuroSolutions will be removed if they are used to override any of the components listed in the previous item.
5. Once a DLL has been created using the Custom Solution Wizard, its network type, component settings, component interconnections, number of inputs, and number of outputs cannot be changed.
6. The Custom Solution Wizard does not support the generation of DLLs for NeuroSolutions breadboards that contain Gradient Search components with their *Individual* switch checked.
7. The Custom Solution Wizard only supports the generation of DLLs for supervised or hybrid networks (a hybrid network is a network that combines an unsupervised pre-processing stage with a supervised post-processing stage). Unsupervised networks are not supported at this time.

The Custom Solution Wizard can be run by clicking on the *Custom Solution Wizard* menu item within the *NeuroSolutions 4*|*Custom Solution Wizard* program group of the Windows *Start* menu or by clicking on the *CustomSolutionWizard* menu item within the *Tools* menu of NeuroSolutions. If the *Necessities* toolbar is displayed in NeuroSolutions (the default), you can also click the CSW button to run the Custom Solution Wizard.

Portion of the Necessities toolbar containing the CSW button

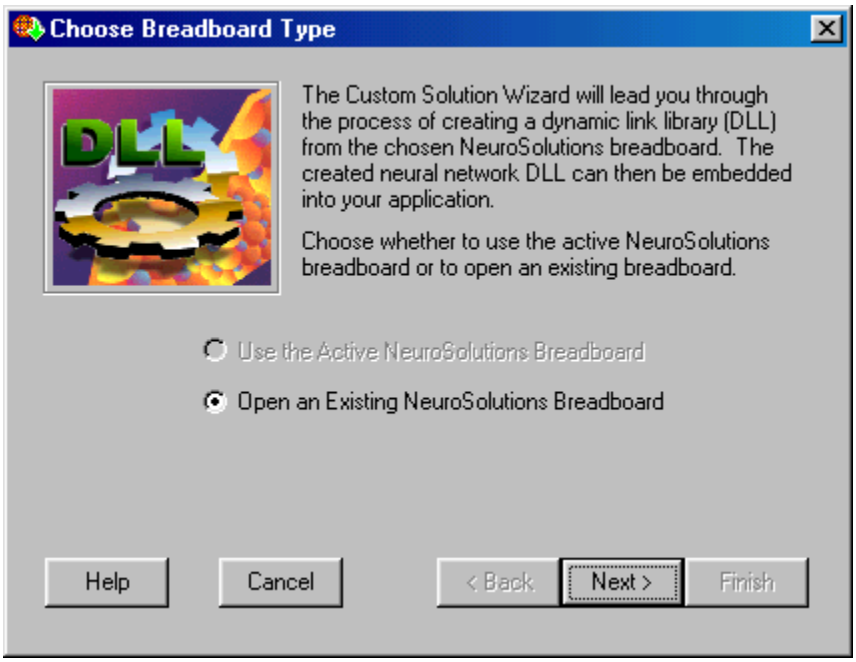
The Wizard will display a series of panels that will guide you step-by-step through the creation of a neural network DLL and a project shell (optional). Depending upon the settings within the NeuroSolutions breadboard being used for the DLL creation, some panels may or may not be displayed. Below is a list of the Custom Solution Wizard Panels (Note: The panels that are only displayed under certain conditions are marked *Conditional*):

- Choose Breadboard Type
- Select Existing Breadboard - *Conditional*
- Data Flow - *Conditional*
- Select Input Axon - *Conditional*
- Select Output Axon - *Conditional*
- Choose Project Type
- Choose Project Location

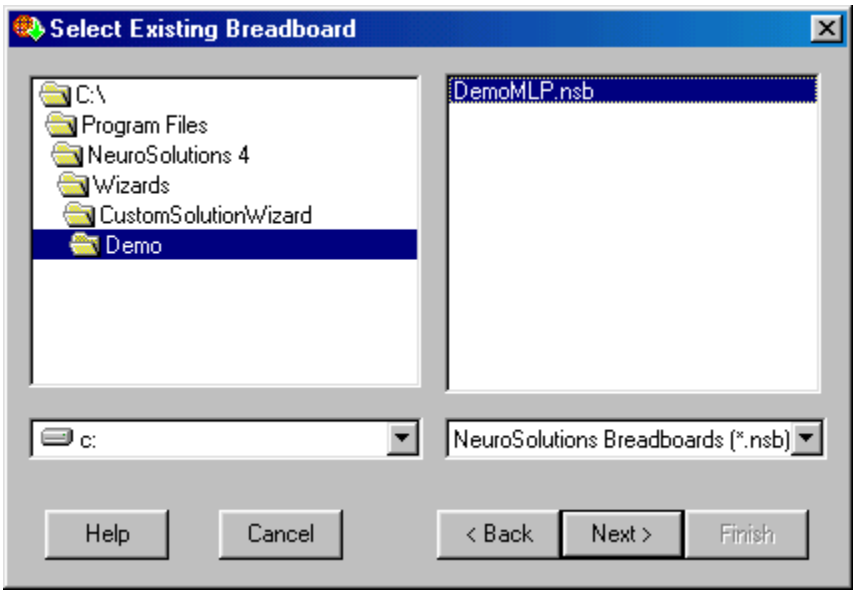
Click on the panel name to find out more information about the purpose of a particular panel.

Upon completion, the Custom Solution Wizard will create a neural network DLL named [Breadboard Name].dll and a weights file named [Breadboard Name].nsw, where [Breadboard Name] is the name of the corresponding NeuroSolutions breadboard without an extension. The created DLL can be accessed from C/C++ or from Visual Basic (Note: The DLL is accessed from Visual Basic using the NeuroSolutions Object Library). The weights file holds the weights for each component and the input and output normalization coefficients contained within the NeuroSolutions breadboard at the time of creation.

The *Choose Breadboard Type* panel is the first panel that you will see when you run the Custom Solution Wizard. This panel gives you the option of using the active NeuroSolutions breadboard or allows you to open an existing NeuroSolutions breadboard for generating the neural network DLL. If there is not an active NeuroSolutions breadboard, the *Use the Active NeuroSolutions Breadboard* option will be disabled.

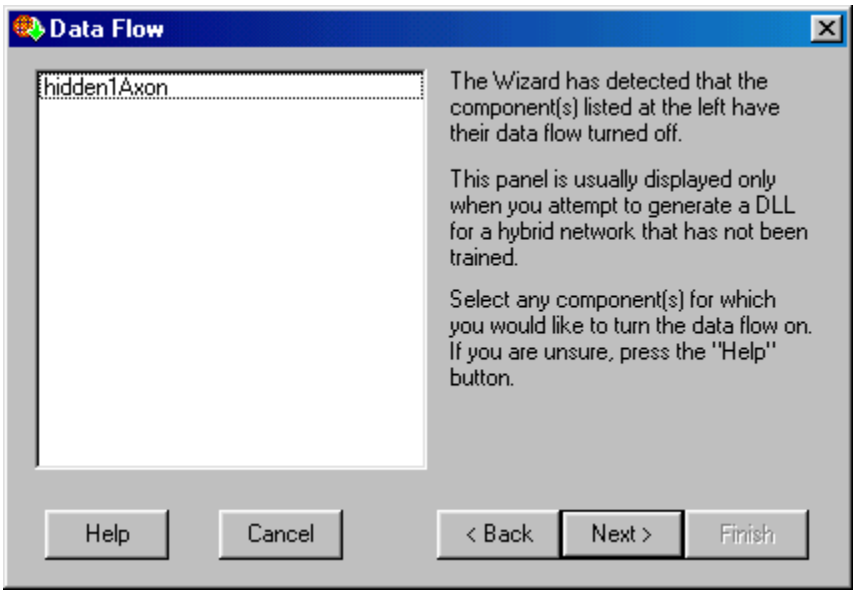


The *Select Existing Breadboard* panel will only be shown if the *Open an Existing NeuroSolutions Breadboard* option was chosen in the *Choose Breadboard Type* panel. This panel allows the user to choose the NeuroSolutions breadboard that will be used to create the network DLL. The Next button on this panel will be disabled until a breadboard has been selected.

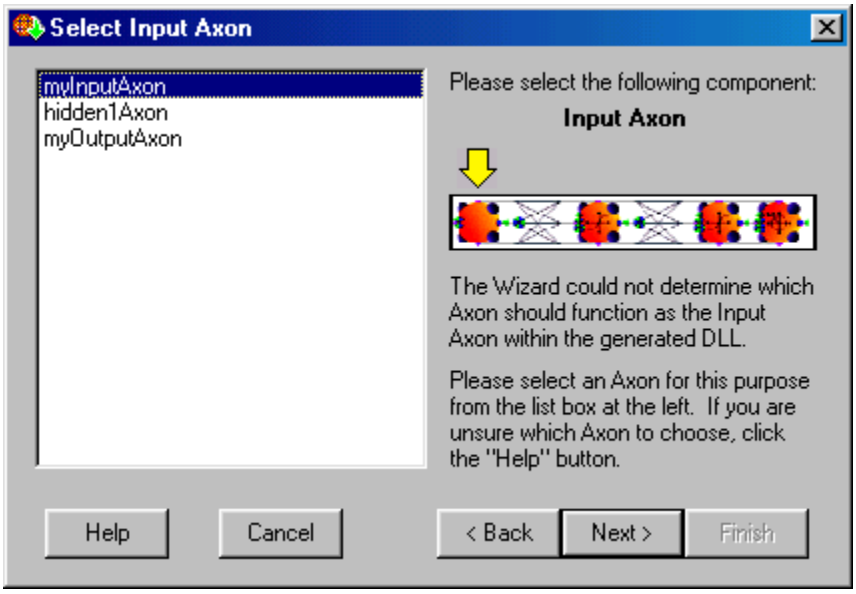


This panel appears only if the Custom Solution Wizard locates axons on the NeuroSolutions breadboard with their data flow turned off. The panel lists each of these axons and you are given the opportunity to turn the axon's data flow on by selecting its name.

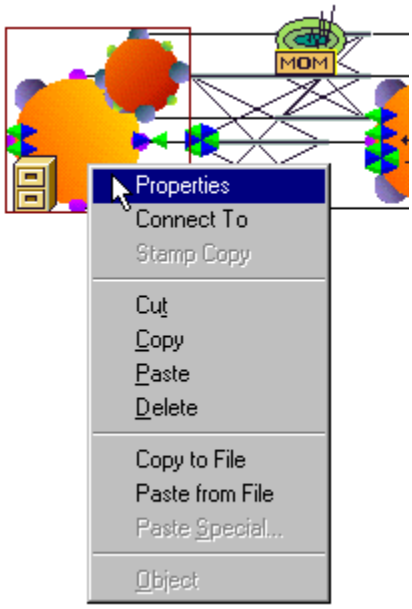
This panel is most commonly displayed when you attempt to build a DLL for a hybrid network that has not been trained. Its purpose is to act as a warning to let you know that the network will need to be trained before attempting to get the network response (output). If you do not have the Developers level of the Custom Solution Wizard, this training will have to be done in NeuroSolutions before generating the network DLL. If you do have the Developers level, simply click *Next* to continue on to the next panel, but make sure you train the network in your application before getting the network response.



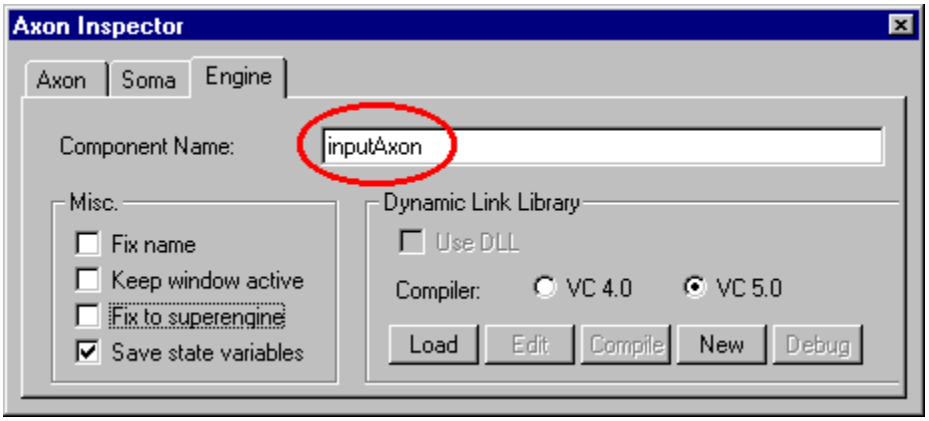
This panel appears only if the wizard could not determine which component is the input axon. The names of all of the axons on the chosen NeuroSolutions breadboard will be displayed and you will be asked to specify which of these axons is the input axon. The input axon is usually located to the far left of the NeuroSolutions breadboard and it usually has a file component  attached to it. It is very important to correctly specify which axon is the input axon. If the wrong axon is chosen, the generated DLL will produce incorrect results.



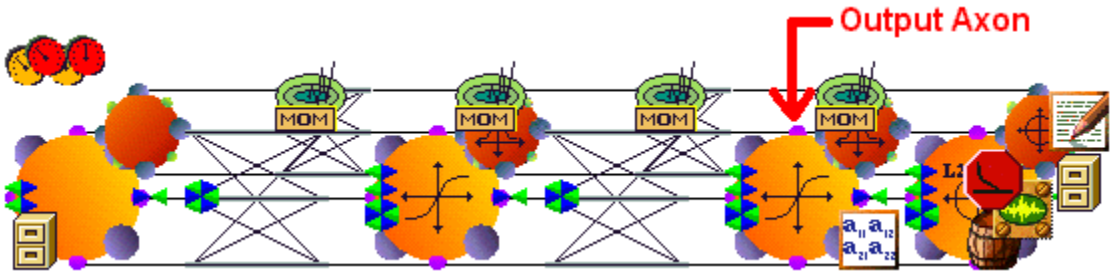
Note: A component's name can be viewed within NeuroSolutions by right-clicking on the component followed by left-clicking on the *Properties* menu item within the resulting shortcut menu.

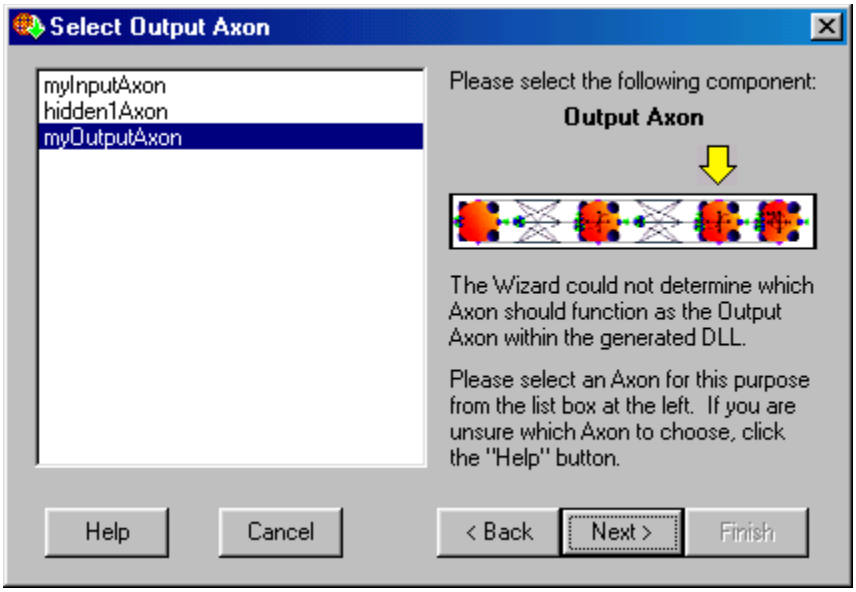


This action will display the inspector for the chosen component. Within the inspector, click on the *Engine* tab to view the component's name. The name can also be edited from here, but never attempt to rename a component while the Custom Solution Wizard is running.



This panel appears only if the wizard could not determine which component is the output axon. The names of all of the axons on the chosen NeuroSolutions breadboard will be displayed and you will be asked to specify which of these axons is the output axon. The output axon is usually the right-most axon on the NeuroSolutions breadboard. Be careful not to confuse the output axon with the criterion component. This is easy to do because a criterion component is similar in appearance to an axon (same size, shape, and color). The output axon is usually located between a hidden axon and a criterion component. The figure below illustrates the location of the output axon on a typical NeuroSolutions breadboard:





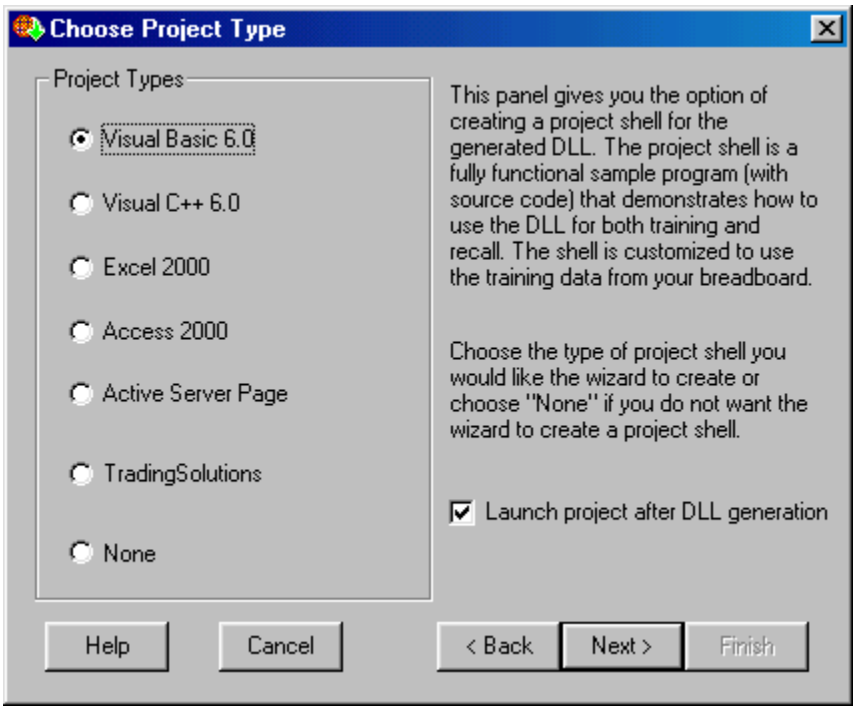
This panel allows you to choose the type of project with which you would like to use the generated DLL. From this panel you can choose the following types of projects:

- § Visual Basic 5.0/6.0
- § Visual C++ 5.0/6.0
- § Excel 97/2000
- § Access 97/2000
- § Active Server Page
- § TradingSolutions
- § None

Note: Excel and Access project shells cannot be created unless one of the versions of Excel or Access listed above is installed on your machine (the options will appear grayed and will not be selectable). Furthermore, only the latest available version of a particular project type can be created. For example, if you have Excel 97 and Excel 2000 installed on your machine, only Excel 2000 will be listed in *Project Type* options.

After the network DLL has been created, the Custom Solution Wizard will create a project shell in the format of the project type you select on this panel (Note: Instead of creating a project shell, the *TradingSolutions* project type simply creates a neural network DLL compatible with TradingSolutions). If the *Launch project after DLL generation* switch is checked, the Wizard will also open this project shell within the appropriate software package. The shell is provided as a guide to help you get started with developing a custom application using the generated neural network DLL.

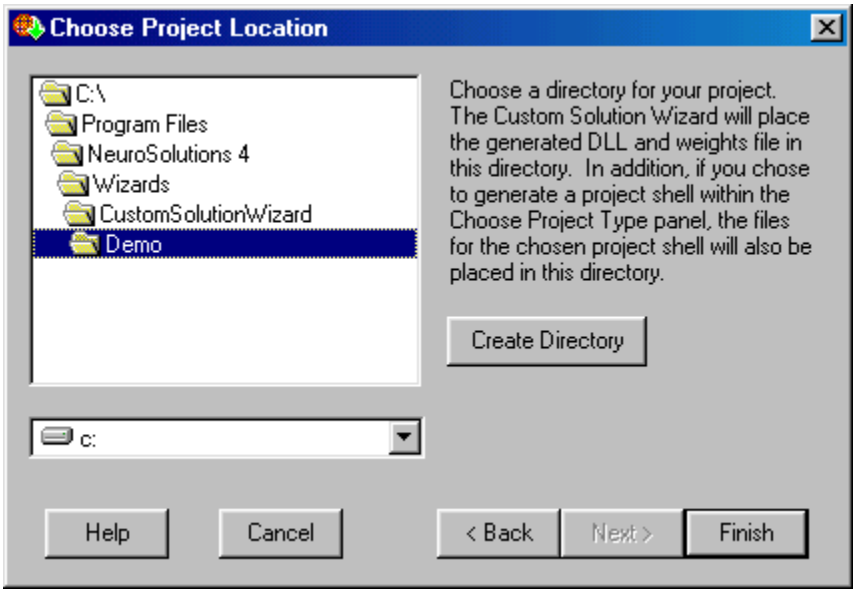
If you choose the *None* option, the Wizard will not create a shell.



This panel is always the last panel displayed by the Custom Solution Wizard. It allows the user to choose the directory in which to place the generated neural network DLL, weights file, and project shell (if a project type was selected in the Choose Project Type panel).

Note: The chosen directory cannot be the Custom Solution Wizards' *ProjectShells* directory. If you choose this directory, you will get a warning telling you to choose another directory.

Click the *Create Directory* button to create a new directory within the currently selected directory. The created directory will automatically become the selected directory.



The NeuroSolutions Object library is an In-Process COM Server implemented through a dynamic link library (DLL) named NeuroSolutionsOL.dll. This DLL was installed in your Windows\System or Winnt\System32 directory (depending upon your operating system) during the Custom Solution Wizard installation. The NeuroSolutions Object Library provides a simple protocol (made up of properties and methods) for communicating with neural network DLLs generated by the Custom Solution Wizard. This protocol makes it extremely easy to use the generated network DLLs from within your application.

The object library allows you to create the following two types of neural network objects:

§ NSLearningNetwork
§ NSRecallNetwork

The protocol for NSRecallNetwork objects is a subset of the protocol for NSLearningNetwork objects. The properties and methods for each of these objects are listed below. Those properties and methods that only apply to NSLearningNetwork objects are marked (NSLearningNetwork only).

Properties:

§ dllPathName
§ inputData
§ desiredData (NSLearningNetwork only)
§ crossValidationEnabled (NSLearningNetwork only)
§ crossValidationInputData (NSLearningNetwork only)
§ crossValidationDesiredData (NSLearningNetwork only)
§ saveBestWeightsEnabled (NSLearningNetwork only)
§ saveBestWeightsForTraining (NSLearningNetwork only)
§ bestWeightsPathName (NSLearningNetwork only)
§ bestCost (NSLearningNetwork only)
§ epochOfBestCost (NSLearningNetwork only)
§ costData (NSLearningNetwork only)
§ crossValidationCostData (NSLearningNetwork only)
§ numberOfEpochsTrained (NSLearningNetwork only)
§ autoComputeInputNormCoeff (NSLearningNetwork only)
§ inputNormMin (NSLearningNetwork only)
§ inputNormMax (NSLearningNetwork only)
§ normalizeInputByChannel (NSLearningNetwork only)
§ autoComputeOutputNormCoeff (NSLearningNetwork only)
§ outputNormMin (NSLearningNetwork only)
§ outputNormMax (NSLearningNetwork only)
§ normalizeOutputByChannel (NSLearningNetwork only)
§ messageErrors

Methods:

§ loadWeights
§ saveWeights
§ seedRandom (NSLearningNetwork only)
§ randomizeWeights
§ resetNetwork
§ train (NSLearningNetwork only)
§ getResponse
§ getSensitivity

§ removeInputNormalization (NSLearningNetwork only)
§ removeOutputNormalization (NSLearningNetwork only)

Note: NSLearningNetwork objects can only be used with DLLs generated by the Developers level of the Custom Solution Wizard.

Using the NeuroSolutions Object Library

The properties and methods of the NeuroSolutions Object library can be used in your application by creating a reference to the object library within your programming environment. The ability to create references is available in Component Object Model (COM)-enabled programming environments like Visual Basic and Visual Basic for Applications (the programming environment for Microsoft Office applications such as Microsoft Access, Microsoft Excel, etc.). Below is a description of how to enable the NeuroSolutions Object Library within a number of different programming applications:

Excel 97/2000

§ Open an existing workbook or create a new workbook
§ Open the Visual Basic Editor by selecting *Tools*|*Macro*|*Visual Basic Editor* from the menu bar of Excel
§ Open the *References* dialog by selecting *Tools*|*References* from the menu bar of the Visual Basic Editor
§ Check the box next to *NeuroSolutions 4.00 Object Library* then click *OK*

Access 97/2000

§ Open an existing database or create a new database
§ Open an existing module or create a new module (by selecting the *Modules* tab then clicking *New*)
§ Open the *References* dialog by selecting *Tools*|*References* from the menu bar
§ Check the box next to *NeuroSolutions 4.00 Object Library*

Visual Basic 5.0/6.0

§ Open an existing project or create a new project
§ Open the *References* dialog by selecting *Project*|*References* from the menu bar
§ Check the box next to *NeuroSolutions 4.00 Object Library*

Visual C++

If you are programming in Visual C++, it is probably best to make calls to the generated neural network DLL directly. The protocol for communicating with the generated network DLL is fully documented. You will find that it is very similar to the NeuroSolutions Object Library protocol. The easiest way to create a Visual C++ application is to configure the Custom Solution Wizard to generate a Visual C++ 5.0/6.0 shell from within the Choose Project Type panel and start with the generated shell.

Note: If you cannot find the *NeuroSolutions 4.00 Object Library* item within the *References* dialog, you will need to register the file *NeuroSolutionsOL.dll* using regsvr32.exe (located in your Windows\System or Winnt\System32 directory). This should not be necessary in most cases, because the object library is automatically registered during the installation of the Custom Solution Wizard. If the object library was unregistered on your machine or you manually copied it to another machine, you will need to register it in order to use it.

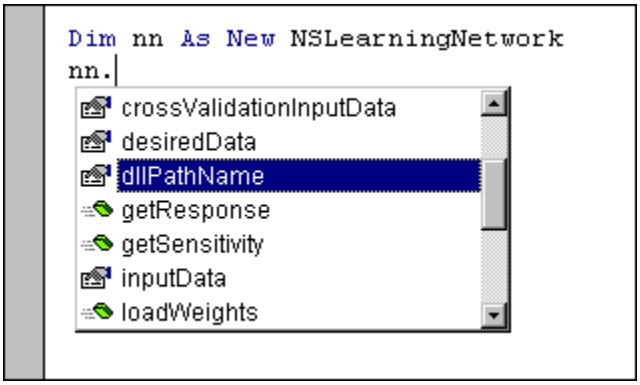
Once you have created a reference to the object library within your programming environment, you begin using the object library by creating one of the two objects discussed above. Here is how you would create an *NSLearningNetwork* object in Visual Basic or any application with VBA:

```
Dim nn As New NSLearningNetwork
```

"nn" is now an *NSLearningNetwork* object and has available all of the properties and methods discussed above. To use a property or method, you use the dot operator. For example, to set the path to the generated DLL (this must always be done), you would type the following line of code:

```
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
```

The really nice thing about Visual Basic and applications with VBA (such as Microsoft Excel and Microsoft Access) is that when you type "nn.", all of the properties and methods of the object will automatically pop up in a list box. You can simply select the desired property or method from the list box using the arrow keys (or by typing the first few letters) then press the *Tab* key.



For information on using each of the properties and methods, see the help for the property or method you are interested in.

An NSLearningNetwork is one the two object types contained within the NeuroSolutions Object Library. To create an NSLearningNetwork within Visual Basic or any application with VBA, you simply enter the following line of code:

```
Dim nn As New NSLearningNetwork
```

This sets *nn* equal to a new instance of an NSLearningNetwork object. NSLearningNetwork objects have the following Properties and Methods at their disposal:

Properties:

§ dllPathName
§ inputData
§ desiredData
§ crossValidationEnabled
§ crossValidationInputData
§ crossValidationDesiredData
§ saveBestWeightsEnabled
§ saveBestWeightsForTraining
§ bestWeightsPathName
§ bestCost
§ epochOfBestCost
§ costData
§ crossValidationCostData
§ numberOfEpochsTrained
§ autoComputeInputNormCoeff
§ inputNormMin
§ inputNormMax
§ normalizeInputByChannel
§ autoComputeOutputNormCoeff
§ outputNormMin
§ outputNormMax
§ normalizeOutputByChannel
§ messageErrors

Methods:

§ loadWeights
§ saveWeights
§ seedRandom
§ randomizeWeights
§ resetNetwork
§ train
§ getResponse
§ getSensitivity
§ removeInputNormalization
§ removeOutputNormalization

An NSRecallNetwork is one the two object types contained within the NeuroSolutions Object Library. To create an NSRecallNetwork within Visual Basic or any application with VBA, you would simply enter the following line of code:

```
Dim nn As New NSRecallNetwork
```

This sets *nn* equal to a new instance of an NSRecallNetwork object. NSRecallNetwork objects have the following Properties and Methods at their disposal:

Properties:

§ dllPathName
§ inputData
§ messageErrors

Methods:

§ loadWeights
§ saveWeights
§ randomizeWeights
§ resetNetwork
§ getResponse
§ getSensitivity

Description

Returns or sets the pathName of the generated neural network DLL. The initial value of this property is an empty string (“”).

Note: This property must be set before using most of the other properties and methods.

Syntax

object.**dllPathName** [= *string*]

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.
string	A string that specifies the location of the generated neural network DLL.

Example:

```
Dim nn As New NSLearningNetwork
nn.dllPathName = “C:\MyDirectory\MyDLL.dll”
```

Error Numbers	Description
-1	DLL initialization failed.

Possible cause

§ One or more of the calls for retrieving the function pointers from DLL failed. The registered version of the NeuroSolutions Object Library may not be compatible with the DLL you are attempting to load. DLLs generated with a particular version of the Custom Solution Wizard must use the NeuroSolutions Object Library included with that version.

-2 Unable to load DLL.

Possible cause

§ The DLL does not exist at the specified location.

2 This DLL does not support learning.

Possible causes

§ The DLL was created with a version of the Custom Solution Wizard that is not licensed for generating learning DLLs. Only the Developers level of the Custom Solution Wizard can generate learning DLLs.

§ The NeuroSolutions breadboard that was used to generate the DLL was a recall network or it did not have one or more of the components necessary for learning.

Description

Sets the neural network input data that is used by the network DLL for the train, getResponse, and getSensitivity methods. This property should be set to a two dimensional variant array of the form (inputs, exemplars). This variant array should contain numeric elements of the type: Integer, Long, Single, or Double.

Syntax

object.**inputData** = *variant*

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.
variant	A 2-D variant array (inputs, exemplars) containing numeric elements of the type: Integer, Long, Single, or Double.

Microsoft Access Example:

This example gets data from a table within an access database and assigns it to the inputData property. Note: GetRows transposes the data as it is read from the table.

```
Dim rstData As Recordset
Set rstData = CurrentDb.OpenRecordset("My Data Table", dbOpenSnapshot)
Dim inputData As Variant
inputData = rstData.GetRows(numberOfExemplars)

Dim nn As New NSLearningNetwork
nn.inputData = inputData
```

Microsoft Excel Example:

This example gets data from a worksheet, transposes it to get it in the form (inputs, exemplars), and assigns it to the inputData property.

```
Dim rawData As Variant
rawData = ActiveWorkbook.Sheets("Sheet1").Range("$A$2:$B$5").Value
Dim inputData() As Variant
ReDim inputData(LBound(rawData, 2) To UBound(rawData, 2), LBound(rawData, 1) To
   UBound(rawData, 1))
Dim rowNum As Long
For rowNum = LBound(rawData, 1) To UBound(rawData, 1)
   Dim colNum As Long
   For colNum = LBound(rawData, 2) To UBound(rawData, 2)
      inputData(colNum, rowNum) = dataArray(rowNum, colNum)
   Next colNum
Next rowNum

Dim nn As New NSLearningNetwork
nn.inputData = inputData
```

General Example:

```
Dim inputData(0 to 1, 0 to 3)  As Variant
inputData(0, 0) = -1!
inputData(0, 1) = -1!
```

```
inputData(0, 2) = 1!
inputData(0, 3) = 1!
inputData(1, 0) = -1!
inputData(1, 1) = 1!
inputData(1, 2) = -1!
inputData(1, 3) = 1!

Dim nn As New NSLearningNetwork
nn.inputData = inputData
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.
-2	The number of inputs expected by the network DLL does not match the number of inputs defined by the size of the training input data array.
-3	The variant data must be of type Array.
-4	Cannot determine data type.
-5	The variant array must be 2-dimensional.

Description

Sets the neural network desired data that is used by the network DLL for the train method. This property should be set to a two dimensional variant array of the form (outputs, exemplars). This variant array should contain numeric elements of the type: Integer, Long, Single, or Double.

Syntax

object.**desiredData** = *variant*

Part	Description
object	An NSLearningNetwork object.
variant	A 2-D variant array (outputs, exemplars) containing numeric elements of the type: Integer, Long, Single, or Double.

Microsoft Access Example:

This example gets data from a table within an access database and assigns it to the desiredData property. Note: GetRows transposes the data as it is read from the table.

```
Dim rstData As Recordset
Set rstData = CurrentDb.OpenRecordset("My Data Table", dbOpenSnapshot)
Dim desiredData As Variant
desiredData = rstData.GetRows(numberOfExemplars)

Dim nn As New NSLearningNetwork
nn.desiredData = desiredData
```

Microsoft Excel Example:

This example gets data from a worksheet, transposes it to get it in the form (outputs, exemplars), and assigns it to the desiredData property.

```
Dim rawData As Variant
rawData = ActiveWorkbook.Sheets("Sheet1").Range("$C$2:$C$5").Value
Dim desiredData() As Variant
ReDim desiredData(LBound(rawData, 2) To UBound(rawData, 2), LBound(rawData, 1) To
   UBound(rawData, 1))
Dim rowNum As Long
For rowNum = LBound(rawData, 1) To UBound(rawData, 1)
   Dim colNum As Long
   For colNum = LBound(rawData, 2) To UBound(rawData, 2)
      desiredData(colNum, rowNum) = dataArray(rowNum, colNum)
   Next colNum
Next rowNum

Dim nn As New NSLearningNetwork
nn.desiredData = desiredData
```

General Example:

```
Dim desiredData(0 to 0, 0 to 3)  As Variant
desiredData(0, 0) = -1!
desiredData(0, 1) = 1!
desiredData(0, 2) = 1!
```

```
desiredData(0, 3) = -1!

Dim nn As New NSLearningNetwork
nn.desiredData = desiredData
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.
-2	The number of outputs expected by the network DLL does not match the number of outputs defined by the size of the training desired data array.
-3	The variant data must be of type Array.
-4	Cannot determine data type.
-5	The variant array must be 2-dimensional.

Description

Returns or sets the cross validation enabled flag within the generated neural network DLL. If this property is set to true, cross validation will be performed during training at the end of each epoch. During cross validation, the error (cost) for a cross validation dataset (separate from the training dataset) is computed by passing this dataset through the network without updating the network weights. The neural network DLL can be configured to automatically save the network weights for the minimum cross validation error (see the saveBestWeightsEnabled, bestWeightsPathName, and saveBestWeightsForTraining properties). Using cross validation, you can build models with better generalization capabilities, since the network weights are saved at the minimum error of a dataset that is not used to update the network weights.

Note: For cross validation to be performed, the crossValidationInputData and crossValidationDesiredData properties must also be set to valid data arrays).

Note: This property is always initialized to False when the DLL is generated, regardless of whether or not the corresponding NeuroSolutions breadboard was configured to use cross validation.

Note: If cross validation is enabled, it will be performed at the end of each epoch regardless of the value of the *Epochs/Cross Val.* setting within the *Control* component on the corresponding NeuroSolutions breadboard.

Syntax

object.**crossValidationEnabled** [= *boolean*]

Part	Description
object	An NSLearningNetwork object.
boolean	A boolean value indicating whether or not cross validation will be used during training.

Example:

This example sets up a neural network DLL for performing cross validation during training and for automatically saving the weights during the minimum cross validation error. Note: The example assumes you have already created input and desired data arrays for training and cross validation.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.crossValidationInputData = myCrossValidationInputDataArray
nn.crossValidationDesiredData = myCrossValidationDesiredDataArray
nn.crossValidationEnabled = True
nn.saveBestWeightsEnabled = True
nn.bestWeightsPathName = "C:\MyDirectory\BestWeights.nsw"
nn.saveBestWeightsForTraining = False
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Sets the neural network cross validation input data that is used by the network DLL for the train method during cross validation. The crossValidationInputData property should be set to a two dimensional variant array of the form (inputs, exemplars). This variant array should contain numeric elements of the type: Integer, Long, Single, or Double.

Note: Cross Validation is only performed if the crossValidationEnabled property is set to True.

Syntax

object.**crossValidationInputData** = *variant*

Part	Description
object	An NSLearningNetwork object.
variant	A 2-D variant array (inputs, exemplars) containing numeric elements of the type: Integer, Long, Single, or Double.

Microsoft Access Example:

This example gets data from a table within an access database and assigns it to the crossValidationInputData property. Note: GetRows transposes the data as it is read from the table.

```
Dim rstData As Recordset
Set rstData = CurrentDb.OpenRecordset("My Data Table", dbOpenSnapshot)
Dim cvInputData As Variant
cvInputData = rstData.GetRows(numberOfExemplars)

Dim nn As New NSLearningNetwork
nn.crossValidationInputData = cvInputData
```

Microsoft Excel Example:

This example gets data from a worksheet, transposes it to get it in the form (inputs, exemplars), and assigns it to the crossValidationInputData property.

```
Dim rawData As Variant
rawData = ActiveWorkbook.Sheets("Sheet1").Range("$A$2:$B$5").Value
Dim cvInputData() As Variant
ReDim cvInputData(LBound(rawData, 2) To UBound(rawData, 2), LBound(rawData, 1) To
   UBound(rawData, 1))
Dim rowNum As Long
For rowNum = LBound(rawData, 1) To UBound(rawData, 1)
   Dim colNum As Long
   For colNum = LBound(rawData, 2) To UBound(rawData, 2)
      cvInputData(colNum, rowNum) = dataArray(rowNum, colNum)
   Next colNum
Next rowNum

Dim nn As New NSLearningNetwork
nn.crossValidationInputData = cvInputData
```

General Example:

```
Dim cvInputData(0 to 1, 0 to 3)  As Variant
cvInputData(0, 0) = -1!
cvInputData(0, 1) = -1!
cvInputData(0, 2) = 1!
cvInputData(0, 3) = 1!
cvInputData(1, 0) = -1!
cvInputData(1, 1) = 1!
cvInputData(1, 2) = -1!
cvInputData(1, 3) = 1!

Dim nn As New NSLearningNetwork
nn.crossValidationInputData = cvInputData
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.
-2	The number of inputs expected by the network DLL does not match the number of inputs defined by the size of the cross validation input data array.
-3	The variant data must be of type Array.
-4	Cannot determine data type.
-5	The variant array must be 2-dimensional.

Description

Sets the neural network cross validation desired data that is used by the network DLL for the train method during cross validation. The crossValidationDesiredData property should be set to a two dimensional variant array of the form (outputs, exemplars). This variant array should contain numeric elements of the type: Integer, Long, Single, or Double.

Note: Cross Validation is only performed if the crossValidationEnabled property is set to True.

Syntax

object.**crossValidationDesiredData** = *variant*

Part	Description
object	An NSLearningNetwork object.
variant	A 2-D variant array (outputs, exemplars) containing numeric elements of the type: Integer, Long, Single, or Double.

Microsoft Access Example:

This example gets data from a table within an access database and assigns it to the crossValidationDesiredData property. Note: GetRows transposes the data as it is read from the table.

```
Dim rstData As Recordset
Set rstData = CurrentDb.OpenRecordset("My Data Table", dbOpenSnapshot)
Dim cvDesiredData As Variant
cvDesiredData = rstData.GetRows(numberOfExemplars)

Dim nn As New NSLearningNetwork
nn.crossValidationDesiredData = cvDesiredData
```

Microsoft Excel Example:

This example gets data from a worksheet, transposes it to get it in the form (outputs, exemplars), and assigns it to the crossValidationDesiredData property.

```
Dim rawData As Variant
rawData = ActiveWorkbook.Sheets("Sheet1").Range("$C$2:$C$5").Value
Dim cvDesiredData() As Variant
ReDim cvDesiredData(LBound(rawData, 2) To UBound(rawData, 2), LBound(rawData, 1) To
   UBound(rawData, 1))
Dim rowNum As Long
For rowNum = LBound(rawData, 1) To UBound(rawData, 1)
   Dim colNum As Long
   For colNum = LBound(rawData, 2) To UBound(rawData, 2)
      cvDesiredData(colNum, rowNum) = dataArray(rowNum, colNum)
   Next colNum
Next rowNum

Dim nn As New NSLearningNetwork
nn.crossValidationDesiredData = cvDesiredData
```

General Example:

```
Dim cvDesiredData(0 to 0, 0 to 3)  As Variant
cvDesiredData(0, 0) = -1!
cvDesiredData(0, 1) = 1!
cvDesiredData(0, 2) = 1!
cvDesiredData(0, 3) = -1!

Dim nn As New NSLearningNetwork
nn.crossValidationDesiredData = cvDesiredData
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.
-2	The number of outputs expected by the network DLL does not match the number of outputs defined by the size of the cross validation desired data array.
-3	The variant data must be of type Array.
-4	Cannot determine data type.
-5	The variant array must be 2-dimensional.

Description

Returns or sets a flag in the network DLL used for enabling/disabling the automatic saving of the best weights during the execution of the train method. Enabling this property will cause the train method to save the network weights during the epoch with the minimum training error (or cross validation error depending on the value of the saveBestWeightsForTraining property).

Note: This property is always initialized to False when the DLL is generated, regardless of whether the corresponding NeuroSolutions breadboard was configured for saving the best weights.

Note: If this property is set to True, the bestWeightsPathName property must be set to a valid pathName before using the train method.

Syntax

object.**saveBestWeightsEnabled** [= *boolean*]

Part	Description
object	An NSLearningNetwork object.
boolean	A boolean indicating whether the save best weights feature is enabled or disabled.

Example:

This example sets up a neural network DLL for automatically saving the weights during the minimum training error. Note: The example assumes you have already created training input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.saveBestWeightsEnabled = True
nn.bestWeightsPathName = "C:\MyDirectory\BestWeights.nsw"
nn.saveBestWeightsForTraining = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets whether the best weights saved during training correspond to the minimum error in the training dataset or the minimum error in the cross validation dataset. The value of this property also affects whether the value returned by the bestCost property corresponds to the training dataset or the cross validation dataset. See the bestCost property for more details.

Note: In order for the best weights to be saved during training, the saveBestWeightsEnabled property must be set to True and the bestWeightPathName property must be set to a valid pathName.

Syntax

object.**saveBestWeightsForTraining** [= *boolean*]

Part	Description
object	An NSLearningNetwork object.
boolean	A boolean indicating whether the best weights are saved during training or cross validation.

Example:

This example sets up a neural network DLL for automatically saving the weights during the minimum training error. Note: The example assumes you have already created training input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.saveBestWeightsEnabled = True
nn.bestWeightsPathName = "C:\MyDirectory\BestWeights.nsw"
nn.saveBestWeightsForTraining = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets the pathName where the best weights will be saved (during training or cross validation) if the saveBestWeightsEnabled property is set to True. The initial value of this property is an empty string (““).

Note: This property must be set to a valid pathName before using the train method if the saveBestWeightsEnabled property has been set to True.

Syntax

object.**bestWeightsPathName** [= *string*]

Part	Description
object	An NSLearningNetwork object.
string	A string that specifies the location for saving the best weights.

Example:

This example sets up a neural network DLL for automatically saving the weights during the minimum training error. Note: The example assumes you have already created training input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = “C:\MyDirectory\MyDLL.dll”
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.saveBestWeightsEnabled = True
nn.bestWeightsPathName = “C:\MyDirectory\BestWeights.nsw”
nn.saveBestWeightsForTraining = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets the best cost for the neural network DLL. The initial value of this property is 1E+009. The value of this property can be changed by setting the property directly, by calling the train method, or by calling the resetNetwork method. Calling the train method updates the bestCost property to the minimum cost (error) achieved during the training process. The bestCost reflects the minimum cost for the training dataset if the crossValidationEnabled property is set to False or if the saveBestWeightsForTraining property is set to True. If the crossValidationEnabled property is set to True and the saveBestWeightsForTraining property is set to False, then the bestCost reflects the minimum cost for the cross validation dataset. The resetNetwork property resets the bestCost to the initial value of 1E+009.

Note: The cost within the generated DLL is computed once every epoch regardless of the value of the *Average cost for:* setting within the *ErrorCriteria* component on the corresponding NeuroSolutions breadboard.

Syntax

object.**bestCost** [= *single*]

Part	Description
object	An NSLearningNetwork object.
single	A single that holds the minimum cost (error).

Example:

This example shows how you would get the best cost for the training dataset after training the neural network DLL. Note: The example assumes that you have already created input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.resetNetwork    'randomizes the network weights
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.train 100
Dim bestCost as single
bestCost = nn.bestCost
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns the epoch number (as a Long) during which the neural network achieved its best cost (minimum error). The returned epoch number corresponds to the best cost for the training dataset if the crossValidationEnabled property is set to False or the saveBestWeightsForTraining property is set to True. If the crossValidationEnabled property is set to True and the saveBestWeightsForTraining property is set to False, then the returned epoch number corresponds to the best cost for the cross validation dataset.

Until the network has been trained, this property will return 0. Furthermore, calling the resetNetwork method will reset the epoch of the best cost to 0.

Note: The corresponding best cost can be obtained by calling the bestCost property.

Syntax

object.**epochOfBestCost**

Part	Description
object	An NSLearningNetwork object.

Example:

This example shows how you would get the epoch of the best cost for the training dataset after training the neural network DLL. Note: The example assumes that you have already created input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.resetNetwork    'randomizes the network weights
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.train 100
Dim epochNumber as Long
epochNumber = nn.epochOfBestCost
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns the training learning curve data for the most recent training run. The cost (error) for the training dataset during each epoch the network was run is returned in a one dimensional variant array. The numberOfEpochsTrained property can be used to determine the size of this array.

Note: The number of epochs actually run may be different than the number passed to the train method. This can occur when the neural network is configured to stop after reaching a certain cost, once the cross validation error begins to rise, etc. This is done through the use of transmitters on the original NeuroSolutions breadboard.

Syntax

object.**costData**

Part	Description
object	An NSLearningNetwork object.

Example:

This example shows how you would train the neural network DLL without using cross validation then retrieve the learning curve for the training dataset. Note: The example assumes that you have already created input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.resetNetwork    'randomizes the network weights
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.train 100
Dim learningCurve as Variant
learningCurve = nn.costData
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.
2	Network must be trained before attempting to retrieve the cost.

Description

Returns the cross validation learning curve data for the most recent training run. The cost (error) for the cross validation dataset during each epoch the network was run is returned in a one dimensional variant array. The numberOfEpochsTrained property can be used to determine the size of this array.

Note: The number of epochs actually run may be different than the number passed to the train method. This can occur when the neural network is configured to stop after reaching a certain cost, once the cross validation error begins to rise, etc. This is done through the use of transmitters on the original NeuroSolutions breadboard.

Note: The network must have been trained with cross validation in order to use this property. Otherwise, an error will occur.

Syntax

object.**crossValidationCostData**

Part	Description
object	An NSLearningNetwork object.

Example:

This example shows how you would train the neural network DLL with cross validation then retrieve the learning curve for the cross validation dataset. Note: The example assumes that you have already created input and desired data arrays for both training and cross validation.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.resetNetwork    'randomizes the network weights
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.crossValidationInputData = myCrossValidationInputDataArray
nn.crossValidationDesiredData = myCrossValidationDesiredDataArray
nn.crossValidationEnabled = True
nn.train 100
Dim cvLearningCurve as Variant
cvLearningCurve = nn.crossValidationCostData
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.
2	Network must be trained (with cross validation) before attempting to retrieve the cross validation cost.

Description

Returns the number of epochs completed during the most recent training run (as a Long). Until the network has been trained, this property will return 0. Furthermore, calling the resetNetwork method will reset the number of epochs trained to 0.

This property is used to get the actual number of epochs completed since this value may be different from the number passed to the train method. This can happen if the neural network was configured to stop after reaching a certain cost, once the cross validation error begins to rise, etc.

Syntax

object.**numberOfEpochsTrained**

Part	Description
object	An NSLearningNetwork object.

Example:

This example shows how you would get the number of epochs actually completed during a training run. Note: The example assumes that you have already created input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.resetNetwork    'randomizes the network weights
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.train 100
Dim epochsCompleted as Long
epochsCompleted = nn.numberOfEpochsTrained
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets a flag indicating whether or not the input normalization coefficients are automatically computed at the beginning of each training run. If this flag is true, the training input data (see the inputData property) will be used to compute the input normalization coefficients at the beginning of a call to the train method. The normalization coefficients are computed according to the inputNormMin , inputNormMax , and normalizeInputByChannel settings.

If this flag is false, the input normalization coefficients are not computed at the beginning of a training run. However, this does not mean that input normalization will not be performed. Once input normalization is activated, it will be performed until it is deactivated, either by calling removeInputNormalization or by loading a weights file that doesn't contain input normalization coefficients.

The initial value of the autoComputeInputNormCoeff property corresponds to the state of the "Normalize" switch on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the input "File" component inspector). This switch will usually be on and, correspondingly, autoComputeInputNormCoeff will usually have a default value of True. If the "Normalize" switch was off, the input normalization was marked as "Read Only"(see the "Data Sets" tab of the input "File" component inspector), or no input "File" component was found when generating the DLL, autoComputeInputNormCoeff will have a default value of False.

Note: Input normalization is activated either by loading a weights file containing input normalization coefficients or by setting the autoComputeInputNormCoeff property to True, followed by a training run.

Syntax

object.**autoComputeInputNormCoeff** [= *boolean*]

Part	Description
object	An NSLearningNetwork object.
boolean	A boolean indicating whether or not the input normalization coefficients are automatically computed at the beginning of each training run.

Example:

This example sets up a neural network DLL for automatically computing the input normalization coefficients by channel using a normalization range of [-1, 1]. The coefficients will not actually be computed until a call to the train method is made.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.inputNormMin = -1
nn.inputNormMax = 1
nn.normalizeInputByChannel = True
nn.autoComputeInputNormCoeff = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets the lower bound used for calculating the input normalization coefficients. The coefficients are calculated (using the training dataset) such that the minimum training input value after normalization is equal to inputNormMin.

The initial value of the inputNormMin property corresponds to the value of the "Lower" setting on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the input "File" component inspector). If no input "File" component was found when generating the DLL, inputNormMin will have a default value of -1.

See the autoComputeInputNormCoeff property for more information on input normalization.

Syntax

object.**inputNormMin** [= *single*]

Part	Description
object	An NSLearningNetwork object.
single	A single that holds the lower bound used for calculating the input normalization coefficients.

Example:

This example sets up a neural network DLL for automatically computing the input normalization coefficients by channel using a normalization range of [-1, 1]. The coefficients will not actually be computed until a call to the train method is made.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.inputNormMin = -1
nn.inputNormMax = 1
nn.normalizeInputByChannel = True
nn.autoComputeInputNormCoeff = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets the upper bound used for calculating the input normalization coefficients. The coefficients are calculated (using the training dataset) such that the maximum training input value after normalization is equal to inputNormMax.

The initial value of the inputNormMax property corresponds to the value of the "Upper" setting on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the input "File" component inspector). If no input "File" component was found when generating the DLL, inputNormMax will have a default value of 1.

See the autoComputeInputNormCoeff property for more information on input normalization.

Syntax

object.**inputNormMax** [= *single*]

Part	Description
object	An NSLearningNetwork object.
single	A single that holds the upper bound used for calculating the input normalization coefficients.

Example:

This example sets up a neural network DLL for automatically computing the input normalization coefficients by channel using a normalization range of [-1, 1]. The coefficients will not actually be computed until a call to the train method is made.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.inputNormMin = -1
nn.inputNormMax = 1
nn.normalizeInputByChannel = True
nn.autoComputeInputNormCoeff = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets a flag indicating whether the input normalization coefficients are calculated on a channel-by-channel basis (each input column considered individually) or across the entire training input dataset.

The initial value of the normalizeInputByChannel property corresponds to the state of the "By Channel" switch on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the input "File" component inspector). If no input "File" component was found when generating the DLL, normalizeInputByChannel will have a default value of True.

See the autoComputeInputNormCoeff property for more information on input normalization.

Syntax

object.**normalizeInputByChannel** [= *boolean*]

Part	Description
object	An NSLearningNetwork object.
boolean	A boolean indicating whether the input normalization coefficients are calculated by channel or across the entire training input dataset.

Example:

This example sets up a neural network DLL for automatically computing the input normalization coefficients by channel using a normalization range of [-1, 1]. The coefficients will not actually be computed until a call to the train method is made.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.inputNormMin = -1
nn.inputNormMax = 1
nn.normalizeInputByChannel = True
nn.autoComputeInputNormCoeff = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets a flag indicating whether or not the output normalization coefficients are automatically computed at the beginning of each training run. If this flag is true, the training desired data (see the desiredData property) will be used to compute the output normalization coefficients at the beginning of a call to the train method. The normalization coefficients are computed according to the outputNormMin, outputNormMax, and normalizeOutputByChannel settings.

If this flag is false, the output normalization coefficients are not computed at the beginning of a training run. However, this does not mean that output normalization will not be performed. Once output normalization is activated, it will be performed until it is deactivated, either by calling removeOutputNormalization or by loading a weights file that doesn't contain output normalization coefficients.

The initial value of the autoComputeOutputNormCoeff property corresponds to the state of the "Normalize" switch on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the desired "File" component inspector). This switch will usually be on and, correspondingly, autoComputeOutputNormCoeff will usually have a default value of True. If the "Normalize" switch was off, the desired normalization was marked as "Read Only"(see the "Data Sets" tab of the desired "File" component inspector), or no desired "File" component was found when generating the DLL, autoComputeOutputNormCoeff will have a default value of False.

Note: Output normalization is activated either by loading a weights file containing output normalization coefficients or by setting the autoComputeOutputNormCoeff property to True, followed by a training run.

Syntax

object.**autoComputeOutputNormCoeff** [= *boolean*]

Part	Description
object	An NSLearningNetwork object.
boolean	A boolean indicating whether or not the output normalization coefficients are automatically computed at the beginning of each training run.

Example:

This example sets up a neural network DLL for automatically computing the output normalization coefficients by channel using a normalization range of [-1, 1]. The coefficients will not actually be computed until a call to the train method is made.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.outputNormMin = -1
nn.outputNormMax = 1
nn.normalizeOutputByChannel = True
nn.autoComputeOutputNormCoeff = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets the lower bound used for calculating the output normalization coefficients. The coefficients are calculated (using the training dataset) such that the minimum training desired value after normalization is equal to outputNormMin.

The initial value of the outputNormMin property corresponds to the value of the "Lower" setting on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the desired "File" component inspector). If no desired "File" component was found when generating the DLL, outputNormMin will have a default value of -1.

See the autoComputeOutputNormCoeff property for more information on output normalization.

Syntax

object.**outputNormMin** [= *single*]

Part	Description
object	An NSLearningNetwork object.
single	A single that holds the lower bound used for calculating the output normalization coefficients.

Example:

This example sets up a neural network DLL for automatically computing the output normalization coefficients by channel using a normalization range of [-1, 1]. The coefficients will not actually be computed until a call to the train method is made.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.outputNormMin = -1
nn.outputNormMax = 1
nn.normalizeOutputByChannel = True
nn.autoComputeOutputNormCoeff = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets the upper bound used for calculating the output normalization coefficients. The coefficients are calculated (using the training dataset) such that the maximum training desired value after normalization is equal to outputNormMax.

The initial value of the outputNormMax property corresponds to the value of the "Upper" setting on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the desired "File" component inspector). If no desired "File" component was found when generating the DLL, outputNormMax will have a default value of 1.

See the autoComputeOutputNormCoeff property for more information on output normalization.

Syntax

object.**outputNormMax** [= *single*]

Part	Description
object	An NSLearningNetwork object.
single	A single that holds the upper bound used for calculating the output normalization coefficients.

Example:

This example sets up a neural network DLL for automatically computing the output normalization coefficients by channel using a normalization range of [-1, 1]. The coefficients will not actually be computed until a call to the train method is made.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.outputNormMin = -1
nn.outputNormMax = 1
nn.normalizeOutputByChannel = True
nn.autoComputeOutputNormCoeff = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets a flag indicating whether the output normalization coefficients are calculated on a channel-by-channel basis (each desired output column considered individually) or across the entire training desired dataset.

The initial value of the normalizeOutputByChannel property corresponds to the state of the "By Channel" switch on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the desired "File" component inspector). If no desired "File" component was found when generating the DLL, normalizeOutputByChannel will have a default value of True.

See the autoComputeOutputNormCoeff property for more information on output normalization.

Syntax

object.**normalizeOutputByChannel** [= *boolean*]

Part	Description
object	An NSLearningNetwork object.
boolean	A boolean indicating whether the output normalization coefficients are calculated by channel or across the entire training desired dataset.

Example:

This example sets up a neural network DLL for automatically computing the output normalization coefficients by channel using a normalization range of [-1, 1]. The coefficients will not actually be computed until a call to the train method is made.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.outputNormMin = -1
nn.outputNormMax = 1
nn.normalizeOutputByChannel = True
nn.autoComputeOutputNormCoeff = True
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Returns or sets whether messages will be displayed by the NeuroSolutions Object Library when an error occurs. Messages are displayed by default.

Syntax

object.**messageErrors** [= *boolean*]

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.
boolean	A boolean indicating whether or not the server will display error messages.

Example:

```
Dim nn As New NSLearningNetwork
nn.messageErrors = False
```

Description

Loads the specified weights file into the neural network DLL. This will load each component's weights and the input and output normalization coefficients. Loading a weights file that contains normalization coefficients will automatically enable the use of normalization when appropriate. If desired, normalization can be disabled using the removeInputNormalization and/or removeOutputNormalization methods.

Only weights files saved by the NeuroSolutions breadboard used to generate the network DLL or by the network DLL itself should be loaded into the network DLL. During DLL generation, the Custom Solution Wizard saves the current breadboard weights. If these weights are not loaded, the network DLL will start with random initial weights.

Syntax

object.**loadWeights** *weightsPathName*

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.
weightsPathName	A string specifying the location of the weights to load.

Example:

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
```

Error Numbers	Description
-1	dllPathName must be set before calling this method.
1	Error loading weights file.

Possible causes

- § The weights file may not be present in the location passed to this method.
- § The weights file may be in use by another application.

2	Invalid weights file.

Possible causes

- § The weights for a component could not be found within the specified weights file.
- § The class name of a component may be incorrectly specified within the weights file.
- § The dimensions (number of weights) of a component specified within the weights file may not match the dimensions (number of weights) expected by the DLL for that component

3	Invalid normalization coefficients.

Possible cause

- § The number of input or output normalization coefficients found in the specified weights file does not match the number of coefficients expected by the DLL.

Description

Immediately saves the neural network DLL's weights/normalization coefficients at the specified location.

Syntax

object.**saveWeights** *weightsPathName*

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.
weightsPathName	A string specifying the location for saving the neural network weights.

Example:

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.saveWeights "C:\MyDirectory\MyWeights.nsw"
```

Error Numbers	Description
-1	dllPathName must be set before calling this method.
1	Error saving weights file.

Possible causes

§ The location passed to this function for saving the weights may be an invalid pathName.
§ The weights file may already exist and be in use by another application.
§ Write access may be restricted for the specified location.

Description

Sets the starting point (seed) for the random number generator. If the seed is set to a fixed value before the execution of any methods or properties that use the random number generator (such as randomizeWeights and resetNetwork), the neural network will produce the same results each time it is run. By default, the random number generator is seeded using the current time when the neural network DLL is initially assigned to the dllPathName property.

Syntax

object.**seedRandom** *seed*

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.
seed	A long specifying the starting point for the random number generator.

Example:

This example shows you how to seed the random number generator so the training results will be identical each time this procedure is run. Note: The example assumes that you have already created input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.seedRandom 1000
nn.resetNetwork    'randomizes the network weights
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.train 100
```

Error Numbers	Description
-1	dllPathName must be set before calling this method.

Description

Sets all of the neural networks weights to small random values. By default, these random values are chosen using a mean of 0 and a variance of .5. The mean and variance used to choose the random values can be set on a component by component basis within the NeuroSolutions breadboard before the DLL is generated.

Syntax

object.**randomizeWeights**

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.

Example:

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.randomizeWeights
```

Error Numbers	Description
-1	dllPathName must be set before calling this method.

Description

Randomizes the network weights and performs any actions set to be performed during a network reset. For example, in the case of a hybrid network, calling the resetNetwork method causes the data flow to be turned off between the unsupervised and supervised portions of the network, the unsupervised learning to be turned on, and the supervised learning to be turned off. In addition, calling resetNetwork causes the bestCost property to be reset to its initial value of 1E+009. In most cases, you will want to use resetNetwork over randomizeWeights when starting a new training session.

Note: Calling the resetNetwork method is equivalent to clicking the reset button in NeuroSolutions.

Syntax

object.**resetNetwork**

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.

Example:

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.resetNetwork
```

Error Numbers	Description
-1	dllPathName must be set before calling this method.

Description

Trains the neural network DLL for the specified number of epochs. Networks can be trained with or without cross validation. See the crossValidationEnabled, crossValidationInputData, and crossValidationDesiredData properties. By default, cross validation is disabled.

The weights file saved during DLL generation can be loaded (using the loadWeights method) before calling the train method in order to start the training of the neural network DLL at the same state the as the original NeuroSolutions breadboard (same weights and normalization coefficients). If the weights file is not loaded, the weights will start at random initial values and the normalization coefficients will be calculated based on the training data (if the original NeuroSolutions breadboard had normalization enabled).

A network can be trained more than once. If the weights are not loaded or randomized after a training session, the execution of the train method for a second time will result in the network starting off where the first training session left off.

Note: Hybrid networks must be trained long enough for the data flow to be turned on between the unsupervised portion and the supervised portion. Otherwise, the getResponse method will return all zeros.

Note: Transmitters and schedulers will affect the training process just as they do in NeuroSolutions. For example, if you had a transmitter (on the NeuroSolutions breadboard used to generate the network DLL) that was set to stop the network when the cost (error) decreased below .001, this transmitter will stop the network when this error is reached, just as it does within NeuroSolutions.

Syntax

*object.***train** *numberOfEpochs*

Part	Description
object	An NSLearningNetwork object.
numberOfEpochs	A long specifying the number of epochs to train the network.

Example:

This example shows how you would train the neural network DLL without using cross validation. Note: The example assumes that you have already created input and desired data arrays.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.resetNetwork    'randomizes the network weights
nn.inputData = myInputDataArray
nn.desiredData = myDesiredDataArray
nn.train 100
```

Error Numbers	Description
-1	dllPathName must be set before calling this method.
-2	inputData must be set before calling this method.
-3	desiredData must be set before calling this method.

-4	The number of training input exemplars does not match the number of training desired exemplars.
-5	The number of cross validation input exemplars does not match the number of cross validation desired exemplars.
2	Error saving weights file.

Possible causes

§ The location specified for saving the best weights (see bestWeightsPathName property) may be invalid.

§ The weights file may already exist and be in use by another application.

§ Write access may be restricted for the directory specified by the bestWeightsPathName property.

3	Invalid number of exemplars in the training dataset.

Possible cause

§ For a dynamic network, the number of exemplars in the training dataset must be evenly divisible be the Samples/Exemplar setting of the original NeuroSolutions breadboard used for generating the DLL. This setting can be found within the DynamicControl inspector.

4	Function bypassed.

Possible causes

§ A DLL overridden component has returned False from the fireIsReady function of the Breadboard Sub-Protocol causing the execution of this function to be bypassed.

§ The number of epochs being passed to the train function may be less than or equal to zero.

Description

Computes the response (output) of the neural network DLL. This method injects the data specified by the inputData property into the network and returns the corresponding network output as a two dimensional variant array of the form (exemplars ,outputs). Before using this method you should either train the neural network DLL or load a set of weights that were saved after a training session. Otherwise, the network output will be random.

Syntax

response = *object*.**getResponse**

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.
response	A 2-D variant array (exemplars ,outputs) containing numeric elements of the type Single.

Example:

This example shows how you would get the response (output) of the neural network DLL. Note: The example assumes that you have a weights file from a previous training session and you have already created an input data array.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.inputData = myInputDataArray
Dim networkOutput as Variant
networkOutput = nn.getResponse
```

Error Numbers	Description
-1	dllPathName must be set before calling this method.
-2	inputData must be set before calling this method.

Description

Computes the raw sensitivity of a neural network model. To compute the sensitivity, each input of the neural network is dithered one by one by the *dither* amount passed into this function. The corresponding effect on each output is computed by calculating the absolute value of the change in the output from its value when the input is not dithered. This computation is performed across all of the exemplars within the input dataset. The total effect that dithering an input has on an output is the sum of the absolute output change across all of the exemplars. This is known as the raw sensitivity. The results are reported in a two dimensional array of the form (inputs, outputs). Each element in this array represents the effect that dithering an input has on an output. Inputs that have little effect on an output (relative to the other inputs) can usually be removed without degrading the model's performance. In many cases, removing worthless inputs will actually enhance the model's performance.

Note: Before using this method, you should either train the neural network DLL or load a set of weights that were saved after a training session. Otherwise, the network sensitivity will be random and meaningless.

Note: A good value for the dither is 0.1 if your input data is being normalized between 0 and 1. This is 10% of the data range. Adjust the dither according to your data range.

Syntax

sensitivity = *object*.**getSensitivity**(*dither*)

Part	Description
object	An NSLearningNetwork or NSRecallNetwork object.
sensitivity	A 2-D variant array (inputs, outputs) containing numeric elements of the type Single.
dither	Amount that gets added to an input during the computation of the sensitivity.

Example:

This example shows how you would get the sensitivity of the neural network DLL. Note: The example assumes that you have a weights file from a previous training session and you have already created an input data array.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.inputData = myInputDataArray
Dim networkSensitivity as Variant
networkSensitivity = nn.getSensitivity(0.1)
```

Error Numbers	Description
-1	dllPathName must be set before calling this method.
-2	inputData must be set before calling this method.

Description

Deactivates input normalization and sets the autoComputeInputNormCoeff property to False (so input normalization won't automatically be reactivated at the beginning of a training run). To reactivate input normalization, either load a weights file containing input normalization coefficients or set the autoComputeInputNormCoeff property to True then train the network.

Syntax

object.**removeInputNormalization**

Part	Description
object	An NSLearningNetwork object.

Example:

This example removes normalization from the input. Note: Be sure not to accidentally reactivate input normalization by loading a weights file containing input normalization coefficients.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.removeInputNormalization
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

Description

Deactivates output normalization and sets the autoComputeOutputNormCoeff property to False (so output normalization won't automatically be reactivated at the beginning of a training run). To reactivate output normalization, either load a weights file containing output normalization coefficients or set the autoComputeOutputNormCoeff property to True then train the network.

Syntax

object.**removeOutputNormalization**

Part	Description
object	An NSLearningNetwork object.

Example:

This example removes normalization from the output. Note: Be sure not to accidentally reactivate output normalization by loading a weights file containing output normalization coefficients.

```
Dim nn As New NSLearningNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"
nn.removeOutputNormalization
```

Error Numbers	Description
-1	dllPathName must be set before calling this property.

The easiest way to build a Visual Basic application for using a neural network DLL is to start with a project shell. A Visual Basic project shell can be generated automatically by choosing the *Visual Basic 5.0/6.0* project type on the Choose Project Type Panel during the creation of the neural network DLL. This will create a sample application (with source code) that will load in your DLL and allow you to train the network and get the network's output.

If you would rather create a Visual Basic application from scratch, this topic will demonstrate how to do this. Simply follow the step-by-step instructions below.

Note: A completed version of this example can be found in the directory: *[NSDirectory]\Wizards\CustomSolutionWizard\Examples\VBExample* where *[NSDirectory]* is the directory where NeuroSolutions was installed (*C:\Program Files\NeuroSolutions 4* by default).

Step 1:
Build a multilayer perceptron (MLP) within NeuroSolutions using the exclusive-or data for the input and desired output.

Inputs		**Desired Output**
X	Y	Z
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Step 2:
Use the Custom Solution Wizard to generate a neural network DLL and weights file for this NeuroSolutions breadboard.

Step 3:
Launch Visual Basic.

Step 4:
Create a new ‘‘Standard EXE’’ project.

Step 5:
Choose *Project*|*References* from the menu bar.

Step 6:
Place a check next to the *NeuroSolutions 4.00 Object Library* item in the *References* dialog box then click *OK.*

Step 7:
Add a button to Form1.

Step 8:
Double-click the button to view the function for its *Click* event.

Step 9:
Insert the code from the Visual Basic Example Code topic into this function.

Step 10:

Change the call to the *dllPathName* property to reflect the location of the neural network DLL you created in **Step 2.**

Step 11:
Change the call to the *loadWeights* method to reflect the location of the weights file you created in **Step 2.**

Step 12:
Run the program.

This is the code to use for **Step 9** of the Visual Basic Example. The code trains the neural network, retrieves its output, and then displays this output in a message box.

```
'Define an input data array using the exclusive-or data
Dim inputData(0 to 1, 0 to 3)  As Variant
inputData(0, 0) = -1!
inputData(0, 1) = -1!
inputData(0, 2) = 1!
inputData(0, 3) = 1!
inputData(1, 0) = -1!
inputData(1, 1) = 1!
inputData(1, 2) = -1!
inputData(1, 3) = 1!

'Define a desired data array using the exclusive-or data
Dim desiredData(0 To 0, 0 To 3) As Variant
desiredData(0, 0) = -1!
desiredData (0, 1) = 1!
desiredData (0, 2) = 1!
desiredData (0, 3) = -1!

'Create and new neural network object of the type NSLearningNetwork
Dim nn As New NSLearningNetwork

'Set the pathName to the generated neural network DLL
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"

'Set the pathName to the weights file saved by the Custom Solution Wizard
nn.loadWeights "C:\MyDirectory\MyWeights.nsw"

'Set the input data defined above as the input data
nn.inputData = inputData

'Set the desired data defined above as the desired data
nn.desiredData = desiredData

'Train the neural network for 100 epochs
nn.train 100

'Get the network response (output)
Dim networkOutput as Variant
networkOutput = nn.getResponse

'Display the network output in a message box
Dim outputString As String
outputString = ""
Dim exemplarNum As Integer, outputNum As Integer
For exemplarNum = LBound(networkOutput, 1) To UBound(networkOutput, 1)
   For outputNum = LBound(networkOutput, 2) To UBound(networkOutput, 2)
```

```
      outputString = outputString & “Output(” & exemplarNum & “, ” & outputNum & “) =
         ” & networkOutput(exemplarNum, outputNum) & Chr(13) & Chr(10)
   Next outputNum
Next exemplarNum

MsgBox outputString

Set nn = Nothing
```

The easiest way to read data into a Visual Basic application from a text file with a standard format is to use the Microsoft DAO 3.6 Object Library. In order to use the DAO object library, you must add a reference to the library:

1. Open or create a new project within Visual Basic 6.0
2. Click the *References* menu item within the *Project* menu of Visual Basic.
3. Place a check next to the *Microsoft DAO 3.6 Object Library* item within the *References* dialog box then click *OK*.

The next step required for reading a text file is to create a schema information file. The schema information file is a file that describes the format of the data in the text file that you want to read. This file should always be named *schema.ini* and be placed in the same location as the file it describes. The schema.ini file is made up of the following five parts:

- The text file name
- The file format
- The field names, widths, and data types
- The character set
- Special data type formats and conversions

Specifying the Text File Name

The first entry in the schema.ini file should always be the name of the text file enclosed in brackets. For example, to specify the file *MyTextFile.txt* as your text file, you would add the following line to the schema.ini file:

```
[MyTextFile.txt]
```

Specifying the File Format

This entry is based on the format of the file itself. The following formats are available:

Format	Description
TabDelimited	Fields in the text file are delimited by tabs.
CSVDelimited	Fields in the text file are delimited by commas.
Delimited(*)	Fields in the text file are delimited by asterisks. You can substitute any character for the asterisk except for the double(“) quotation mark.
FixedLength	Fields in the text file are of a fixed width.

For example, to specify that your file is tab-delimited, you would add the following line to the schema.ini file:

```
Format = TabDelimited
```

Specifying the Field Names, Widths, and Data Types

For delimited text files, you can specify field names in one of two ways:

- Include the field names in the first record of the text file and set the *ColNameHeader* entry to True to indicate that the first record of data specifies the field names.
- Specify each field by number and designate the field name and data type.

To have DAO determine the data types of the fields for you, set the *MaxScanRows* entry to the number of records that should scanned for determining the field data types. If you set the MaxScanRows entry to 0, the whole file will

be scanned. For example, to specify that the first record in the text file contains the field names and that the entire file should be scanned to determine the data type of each field, add the following entries to your Schema.ini file:

```
ColNameHeader=True
MaxScanRows=0
```

For fixed-width files, you must specify each field by number and designate the field name, data type, and width. The Col*n* entry is used for this purpose. The syntax of the Col*n* entry is:

Col*n*=*ColumnName type* [**Width** #]

Part	**Description**
n	Designates the column (field) being specified.
ColumnName	The name of the field. If the field name contains embedded spaces, you must enclose it in double quotation marks.
type	Specifies the data type for the field. It can be any of the following data types: Byte, Long, Currency, Single, Double, DateTime, Text, Memo
#	An integer value that specifies the number of characters in the field (required if **Width** is specified).

For example, to specify two fields, a 10-character text field named *Field1* and a 30-character text field named *Field2*, you would add the following entries to your Schema.ini file:

```
Col1=Field1 Text Width 10
Col2= Field2 Text Width 30
```

Specifying a Character Set

The *CharacterSet* entry specifies which character set your computer uses. You can specify one of two character sets: ANSI or OEM. For example, to specify the ANSI character set, you would add the following entry to your Schema.ini file:

```
CharacterSet=ANSI
```

Specifying Data Type Formats and Conversions

The Schema.ini file contains a number of entries that you can use to specify how data is converted. The following table describes each of these entries.

Entry	**Description**
DateTimeFormat	A format string that specifies the format for dates and times. You should specify this entry if all Date/Time fields have the same format. You can use any Date/Time format except A.M. and P.M. display formats.
DecimalSymbol	Specifies the character used to separate the integer from the fractional portion of a number.
NumberDigits	Specifies the number of decimal digits in the fractional portion of a number.
NumberLeadingZeros	Specifies whether a decimal value less than 1 and greater than -1 should contain leading zeros; you can set this entry to False (no leading zeros) or True.
CurrencySymbol	Specifies the currency symbol used for currency values in the text file; for example, the dollar sign ($) or Dm.
CurrencyPosFormat	Specifies the position of the currency symbol. You can set it to any of the following values: 0 Currency symbol prefix with no separation ($1) 1 Currency symbol suffix with no separation (1$)

	2 Currency symbol prefix with one character separation ($ 1) 3 Currency symbol suffix with one character separation (1 $)
CurrencyDigits	Specifies the number of digits used for the fractional portion of a currency value.
CurrencyNegFormat	Specifies the format of negative currency values. You can set it to any of the following numbers: 0 ($1), 1 -$1, 2 $-1, 3 $1-, 4 (1$), 5 -1$, 6 1-$, 7 1$-, 8 -1 $, 9 -$ 1, 10 1 $-, 11 $ 1-, 12 $ -1, 13 1- $, 14 ($ 1), 15 (1 $) The dollar sign ($) is shown only as an example; you should replace it with the appropriate currency symbol.
CurrencyThousandSymbol	Specifies the character used to separate thousands in currency values.
CurrencyDecimalSymbol	Specifies the character used to separate the whole from the fractional portion of a currency value.

Note: If you omit an entry, the default value specified in the Windows control panel is used.

The final step for reading a text file is to write the Visual Basic code for accessing the data in the file. To do this, you first create a *Database* object using the *OpenDatabase* method. The arguments of this method specify the full path to the text file, whether to open the text file exclusively, whether to open it with read/write or read-only permissions, and the source database type. Next, you create a *Recordset* object using the *OpenRecordset* method specifying the file name as the source argument. Finally, the *GetRows* method is used to read in the specfied number of records from the file. GetRows returns a 2-D Variant array holding the data read in from the file. Note: GetRows transposes the data from its original orientation.

See Reading Data from Text Files Example for a complete example that uses the concepts discussed above to read a tab-delimited text file.

Note: The Microsoft Jet Text ISAM must also be installed on your machine in order to use this method for reading text files.

This example demonstrates the use of the concepts discussed in the Reading Data from Text Files topic by showing how to read data into a Visual Basic application from a tab-delimited ASCII text file. The data read-in is converted into a 2-D variant array ready to function as the input for a neural network object created with the NeuroSolutions Object Library.

Assume that you have the following exclusive-or input data located in a file named *MyInputData.txt* at the location *C:\MyDirectory*:

X	Y
-1	-1
-1	1
1	-1
1	1

Also, assume that the data is tab-delimited meaning that a schema.ini file (placed at the same location as the data file) with the following information could be used to define this data file:

```
[MyInputData.txt]
ColNameHeader = True
Format = TabDelimited
MaxScanRows = 0
CharacterSet = ANSI
```

The following code can be used to read in the exclusive-or input data and convert it into a 2-D variant array.

```
Dim dbs As Database
Set dbs = OpenDatabase("C:\MyDirectory", False, True, "Text;")

Dim rst As Recordset
Set rst = dbs.OpenRecordset("MyInputData.txt", dbOpenSnapshot)
rst.MoveLast
Dim numberOfRecords As Long
numberOfRecords = rst.RecordCount
rst.MoveFirst
Dim myInputDataArray as Variant
myInputDataArray = rst.GetRows(numberOfRecords)

rst.Close
dbs.Close
```

After running this code, the array *myInputDataArray* will contain the exclusive-or data. The following example shows how this array can then be used to define the input data for a neural network object:

```
Dim nn As NSRecallNetwork
nn.dllPathName = "C:\MyDirectory\MyDLL.dll"
nn.loadWeights "C:\MyDirectory\MyWeights.dll"
nn.inputData = myInputDataArray
Dim networkOutput As Variant
networkOutput = nn.getResponse
```

Like Visual Basic applications, C++ applications can use the NeuroSolutions Object Library to communicate with the generated neural network DLL. However, it is more straightforward and efficient to simply communicate with the DLL directly. The protocol for this direct communication is defined below. You will find that it is very similar to the protocol of the NeuroSolutions Object Library.

The first step in communicating with the neural network DLL is to load the DLL and establish pointers to each of the DLLs functions. The code for performing this task is provided in the C++ Code Needed to Load the Generated DLL topic and can simply be copied and pasted into your application.

Once the DLL has been loaded, the next step is to use the createNetwork function to create a recall (NSRecallNetwork) or learning (NSLearningNetwork) instance of the neural network DLL. A pointer to this network instance gets set during this function call. This pointer must be passed as a parameter when calling any of the other functions. A list of the available functions is given below. After you are done using the neural network DLL, you must call the detroyNetwork function to free the memory that was allocated for this instance. You must follow this up with a call to the *FreeLibrary* function in order to release the DLL.

The protocol for recall networks is a subset of the protocol for learning networks. The functions for each of these network type are listed below. Those functions that only apply to learning networks are marked (NSLearningNetwork only).

Note: To create a learning instance of a neural network, the neural network DLL must be capable of learning. Only DLLs created with the *Developers* level of the Custom Solution Wizard are capable of learning. DLLs created with any other level can only be instantiated as recall networks.

Available Functions:

Operations

§ createNetwork
§ destroyNetwork
§ getInputOutputInfo
§ loadWeights
§ saveWeights
§ seedRandom (NSLearningNetwork only)
§ randomizeWeights
§ resetNetwork
§ train (NSLearningNetwork only)
§ getResponse
§ getSensitivity
§ removeInputNormalization
§ removeOutputNormalization

Parameters

§ (set / get)CrossValidationEnabled (NSLearningNetwork only)
§ (set / get)SaveBestWeightsEnabled (NSLearningNetwork only)
§ (set / get)SaveBestWeightsForTraining (NSLearningNetwork only)
§ (set / get)BestWeightsPathName (NSLearningNetwork only)
§ (set / get)BestCost (NSLearningNetwork only)
§ getEpochOfBestCost (NSLearningNetwork only)
§ getCostData (NSLearningNetwork only)
§ getCrossValidationCostData (NSLearningNetwork only)
§ getNumberOfEpochsTrained (NSLearningNetwork only)

§ (set / get)AutoComputeInputNormCoeff (NSLearningNetwork only)
§ (set / get)InputNormMin (NSLearningNetwork only)
§ (set / get)InputNormMax (NSLearningNetwork only)
§ (set / get)NormalizeInputByChannel (NSLearningNetwork only)
§ (set / get)AutoComputeOutputNormCoeff (NSLearningNetwork only)
§ (set / get)OutputNormMin (NSLearningNetwork only)
§ (set / get)OutputNormMax (NSLearningNetwork only)
§ (set / get)NormalizeOutputByChannel (NSLearningNetwork only)

The first step in communicating with a neural network DLL is to load the DLL and establish pointers to each of the DLLs functions. The code for performing this task is provided below and can be copied and pasted into your application. Before you can call any of the DLLs functions, you will need to call the *LoadDLLFunctions* function, passing it the path to the DLL. Be sure to save the return value of this function as it should be passed to the *FreeLibrary* function in order to release the DLL when you are done using it.

Note: If your DLL only supports recall (only the Developers version of the Custom Solution Wizard can create learning DLLs), you will need to remove the blocks of code whose comment contains the phrase "for LEARNING DLLs only".

```
#include <windows.h>
HINSTANCE LoadDLLFunctions(char *dllPathName);

// Define function pointer types for both RECALL and LEARNING DLLs
typedef int (*NSCreateNetwork)(void *&pNeuralNetwork, int networkType);
typedef int (*NSDestroyNetwork)(void *pNeuralNetwork);
typedef int (*NSGetInputOutputInfo)(void *pNeuralNetwork, int &numInputs, int
   &numOutputs);
typedef int (*NSGetResponse)(void *pNeuralNetwork, int exemplars, float *inputData,
   float *outputData);
typedef int (*NSGetSensitivity)(void *pNeuralNetwork, int exemplars, float
   *inputData, float *&sensitivityData, float dither);
typedef int (*NSLoadWeights)(void *pNeuralNetwork, const char *weightsPathName);
typedef int (*NSSaveWeights)(void *pNeuralNetwork, const char *weightsPathName);
typedef int (*NSRandomizeWeights)(void *pNeuralNetwork);
typedef int (*NSResetNetwork)(void *pNeuralNetwork);

// Define function pointer types for LEARNING DLLs only
typedef int (*NSTrain)(void *pNeuralNetwork, int epochs, int exemplars, float
   *inputData, float *desiredData, int cvExemplars, float *cvInputData, float
   *cvDesiredData);
typedef int (*NSGetBestCost)(void *pNeuralNetwork, float &bestCost);
typedef int (*NSSetBestCost)(void *pNeuralNetwork, float bestCost);
typedef int (*NSGetBestWeightsPathName)(void *pNeuralNetwork, char
   *bestWeightsPathName, int bufferLength, int &pathNameLength);
typedef int (*NSSetBestWeightsPathName)(void *pNeuralNetwork, const char
   *bestWeightsPathName);
typedef int (*NSGetCrossValidationEnabled)(void *pNeuralNetwork, bool
   &crossValidationEnabled);
typedef int (*NSSetCrossValidationEnabled)(void *pNeuralNetwork, bool
   crossValidationEnabled);
typedef int (*NSGetSaveBestWeightsEnabled)(void *pNeuralNetwork, bool
   &saveBestWeightsEnabled);
typedef int (*NSSetSaveBestWeightsEnabled)(void *pNeuralNetwork, bool
   saveBestWeightsEnabled);
typedef int (*NSGetSaveBestWeightsForTraining)(void *pNeuralNetwork, bool
   &saveBestWeightsForTraining);
typedef int (*NSSetSaveBestWeightsForTraining)(void *pNeuralNetwork, bool
   saveBestWeightsForTraining);
typedef int (*NSSetAutoComputeInputNormCoeff)(void *pNeuralNetwork, bool
   autoComputeInputNormCoeff);
typedef int (*NSGetAutoComputeInputNormCoeff)(void *pNeuralNetwork, bool
   &autoComputeInputNormCoeff);
```

```
typedef int (*NSSetAutoComputeOutputNormCoeff)(void *pNeuralNetwork, bool
   autoComputeOutputNormCoeff);
typedef int (*NSGetAutoComputeOutputNormCoeff)(void *pNeuralNetwork, bool
   &autoComputeOutputNormCoeff);
typedef int (*NSRemoveInputNormalization)(void *pNeuralNetwork);
typedef int (*NSRemoveOutputNormalization)(void *pNeuralNetwork);
typedef int (*NSSetInputNormMin)(void *pNeuralNetwork, float inputNormMin);
typedef int (*NSGetInputNormMin)(void *pNeuralNetwork, float &inputNormMin);
typedef int (*NSSetInputNormMax)(void *pNeuralNetwork, float inputNormMax);
typedef int (*NSGetInputNormMax)(void *pNeuralNetwork, float &inputNormMax);
typedef int (*NSSetOutputNormMin)(void *pNeuralNetwork, float outputNormMin);
typedef int (*NSGetOutputNormMin)(void *pNeuralNetwork, float &outputNormMin);
typedef int (*NSSetOutputNormMax)(void *pNeuralNetwork, float outputNormMax);
typedef int (*NSGetOutputNormMax)(void *pNeuralNetwork, float &outputNormMax);
typedef int (*NSSetNormalizeInputByChannel)(void *pNeuralNetwork, bool
   normalizeInputByChannel);
typedef int (*NSGetNormalizeInputByChannel)(void *pNeuralNetwork, bool
   &normalizeInputByChannel);
typedef int (*NSSetNormalizeOutputByChannel)(void *pNeuralNetwork, bool
   normalizeOutputByChannel);
typedef int (*NSGetNormalizeOutputByChannel)(void *pNeuralNetwork, bool
   &normalizeOutputByChannel);
typedef int (*NSGetCrossValidationCostData)(void *pNeuralNetwork, float
   *cvCostData);
typedef int (*NSGetCostData)(void *pNeuralNetwork, float *costData);
typedef int (*NSGetNumberOfEpochsTrained)(void *pNeuralNetwork, int
   &numberOfEpochsTrained);
typedef int (*NSGetEpochOfBestCost)(void *pNeuralNetwork, int &epochOfBestCost);
typedef int (*NSSeedRandom)(void *pNeuralNetwork, unsigned int seed);

// Declare function pointers for both RECALL and LEARNING DLLs
NSCreateNetwork createNetwork;
NSDestroyNetwork destroyNetwork;
NSLoadWeights loadWeights;
NSSaveWeights saveWeights;
NSRandomizeWeights randomizeWeights;
NSResetNetwork resetNetwork;
NSGetInputOutputInfo getInputOutputInfo;
NSGetResponse getResponse;
NSGetSensitivity getSensitivity;

// Declare function pointers for LEARNING DLLs only
NSGetBestCost getBestCost;
NSSetBestCost setBestCost;
NSTrain train;
NSGetBestWeightsPathName getBestWeightsPathName;
NSSetBestWeightsPathName setBestWeightsPathName;
NSGetCrossValidationEnabled getCrossValidationEnabled;
NSSetCrossValidationEnabled setCrossValidationEnabled;
NSGetSaveBestWeightsEnabled getSaveBestWeightsEnabled;
NSSetSaveBestWeightsEnabled setSaveBestWeightsEnabled;
NSGetSaveBestWeightsForTraining getSaveBestWeightsForTraining;
NSSetSaveBestWeightsForTraining setSaveBestWeightsForTraining;
```

```
NSSetAutoComputeInputNormCoeff setAutoComputeInputNormCoeff;
NSGetAutoComputeInputNormCoeff getAutoComputeInputNormCoeff;
NSSetAutoComputeOutputNormCoeff setAutoComputeOutputNormCoeff;
NSGetAutoComputeOutputNormCoeff getAutoComputeOutputNormCoeff;
NSRemoveInputNormalization removeInputNormalization;
NSRemoveOutputNormalization removeOutputNormalization;
NSSetInputNormMin setInputNormMin;
NSGetInputNormMin getInputNormMin;
NSSetInputNormMax setInputNormMax;
NSGetInputNormMax getInputNormMax;
NSSetOutputNormMin setOutputNormMin;
NSGetOutputNormMin getOutputNormMin;
NSSetOutputNormMax setOutputNormMax;
NSGetOutputNormMax getOutputNormMax;
NSSetNormalizeInputByChannel setNormalizeInputByChannel;
NSGetNormalizeInputByChannel getNormalizeInputByChannel;
NSSetNormalizeOutputByChannel setNormalizeOutputByChannel;
NSGetNormalizeOutputByChannel getNormalizeOutputByChannel;
NSGetCrossValidationCostData getCrossValidationCostData;
NSGetCostData getCostData;
NSGetNumberOfEpochsTrained getNumberOfEpochsTrained;
NSGetEpochOfBestCost getEpochOfBestCost;
NSSeedRandom seedRandom;

HINSTANCE LoadDLLFunctions(char *dllPathName)
{
   HINSTANCE hDLL = LoadLibrary(dllPathName);
   if (hDLL)
   {
      // Assign function pointers for both RECALL and LEARNING DLLs
      createNetwork = (NSCreateNetwork)GetProcAddress(hDLL, "createNetwork");
      destroyNetwork = (NSDestroyNetwork)GetProcAddress(hDLL, "destroyNetwork");
      getInputOutputInfo = (NSGetInputOutputInfo)GetProcAddress(hDLL,
         "getInputOutputInfo");
      getResponse = (NSGetResponse)GetProcAddress(hDLL, "getResponse");
      getSensitivity = (NSGetSensitivity)GetProcAddress(hDLL, "getSensitivity");
      loadWeights = (NSLoadWeights)GetProcAddress(hDLL, "loadWeights");
      saveWeights = (NSSaveWeights)GetProcAddress(hDLL, "saveWeights");
      randomizeWeights = (NSRandomizeWeights)GetProcAddress(hDLL,
         "randomizeWeights");
      resetNetwork = (NSResetNetwork)GetProcAddress(hDLL, "resetNetwork");

      // Assign function pointers for LEARNING DLLs only
      train = (NSTrain)GetProcAddress(hDLL, "train");
      getBestCost = (NSGetBestCost)GetProcAddress(hDLL, "getBestCost");
      setBestCost = (NSSetBestCost)GetProcAddress(hDLL, "setBestCost");
      getBestWeightsPathName = (NSGetBestWeightsPathName)GetProcAddress(hDLL,
         "getBestWeightsPathName");
      setBestWeightsPathName = (NSSetBestWeightsPathName)GetProcAddress(hDLL,
         "setBestWeightsPathName");
      getSaveBestWeightsEnabled = (NSGetSaveBestWeightsEnabled)GetProcAddress(hDLL,
         "getSaveBestWeightsEnabled");
```

```
setSaveBestWeightsEnabled = (NSSetSaveBestWeightsEnabled)GetProcAddress(hDLL,
   “setSaveBestWeightsEnabled”);
getSaveBestWeightsForTraining =
   (NSGetSaveBestWeightsForTraining)GetProcAddress(hDLL,
   “getSaveBestWeightsForTraining”);
setSaveBestWeightsForTraining =
   (NSSetSaveBestWeightsForTraining)GetProcAddress(hDLL,
   “setSaveBestWeightsForTraining”);
getCrossValidationEnabled = (NSGetCrossValidationEnabled)GetProcAddress(hDLL,
   “getCrossValidationEnabled”);
setCrossValidationEnabled = (NSSetCrossValidationEnabled)GetProcAddress(hDLL,
   “setCrossValidationEnabled”);
setAutoComputeInputNormCoeff =
   (NSSetAutoComputeInputNormCoeff)GetProcAddress(hDLL,
   “setAutoComputeInputNormCoeff”);
getAutoComputeInputNormCoeff =
   (NSGetAutoComputeInputNormCoeff)GetProcAddress(hDLL,
   “getAutoComputeInputNormCoeff”);
setAutoComputeOutputNormCoeff =
   (NSSetAutoComputeOutputNormCoeff)GetProcAddress(hDLL,
   “setAutoComputeOutputNormCoeff”);
getAutoComputeOutputNormCoeff =
   (NSGetAutoComputeOutputNormCoeff)GetProcAddress(hDLL,
   “getAutoComputeOutputNormCoeff”);
removeInputNormalization = (NSRemoveInputNormalization)GetProcAddress(hDLL,
   “removeInputNormalization”);
removeOutputNormalization = (NSRemoveOutputNormalization)GetProcAddress(hDLL,
   “removeOutputNormalization”);
setInputNormMin = (NSSetInputNormMin)GetProcAddress(hDLL, “setInputNormMin”);
getInputNormMin = (NSGetInputNormMin)GetProcAddress(hDLL, “getInputNormMin”);
setInputNormMax = (NSSetInputNormMax)GetProcAddress(hDLL, “setInputNormMax”);
getInputNormMax = (NSGetInputNormMax)GetProcAddress(hDLL, “getInputNormMax”);
setOutputNormMin = (NSSetOutputNormMin)GetProcAddress(hDLL,
   “setOutputNormMin”);
getOutputNormMin = (NSGetOutputNormMin)GetProcAddress(hDLL,
   “getOutputNormMin”);
setOutputNormMax = (NSSetOutputNormMax)GetProcAddress(hDLL,
   “setOutputNormMax”);
getOutputNormMax = (NSGetOutputNormMax)GetProcAddress(hDLL,
   “getOutputNormMax”);
setNormalizeInputByChannel =
   (NSSetNormalizeInputByChannel)GetProcAddress(hDLL,
   “setNormalizeInputByChannel”);
getNormalizeInputByChannel =
   (NSGetNormalizeInputByChannel)GetProcAddress(hDLL,
   “getNormalizeInputByChannel”);
setNormalizeOutputByChannel =
   (NSSetNormalizeOutputByChannel)GetProcAddress(hDLL,
   “setNormalizeOutputByChannel”);
getNormalizeOutputByChannel =
   (NSGetNormalizeOutputByChannel)GetProcAddress(hDLL,
   “getNormalizeOutputByChannel”);
getCrossValidationCostData =
   (NSGetCrossValidationCostData)GetProcAddress(hDLL,
   “getCrossValidationCostData”);
getCostData = (NSGetCostData)GetProcAddress(hDLL, “getCostData”);
```

```
getNumberOfEpochsTrained = (NSGetNumberOfEpochsTrained)GetProcAddress(hDLL,
   “getNumberOfEpochsTrained”);
getEpochOfBestCost = (NSGetEpochOfBestCost)GetProcAddress(hDLL,
   “getEpochOfBestCost”);
seedRandom = (NSSeedRandom)GetProcAddress(hDLL, “seedRandom”);

if (!(
   // Check function pointers for both RECALL and LEARNING DLLs
   createNetwork &&
   destroyNetwork &&
   getInputOutputInfo &&
   getResponse &&
   getSensitivity &&
   loadWeights &&
   saveWeights &&
   randomizeWeights &&
   resetNetwork &&

   // Check function pointers for LEARNING DLLs only
   train &&
   getBestCost &&
   setBestCost &&
   getBestWeightsPathName &&
   setBestWeightsPathName &&
   getSaveBestWeightsEnabled &&
   setSaveBestWeightsEnabled &&
   getSaveBestWeightsForTraining &&
   setSaveBestWeightsForTraining &&
   getCrossValidationEnabled &&
   setCrossValidationEnabled &&
   setAutoComputeInputNormCoeff &&
   getAutoComputeInputNormCoeff &&
   setAutoComputeOutputNormCoeff &&
   getAutoComputeOutputNormCoeff &&
   removeInputNormalization &&
   removeOutputNormalization &&
   setInputNormMin &&
   getInputNormMin &&
   setInputNormMax &&
   getInputNormMax &&
   setOutputNormMin &&
   getOutputNormMin &&
   setOutputNormMax &&
   getOutputNormMax &&
   setNormalizeInputByChannel &&
   getNormalizeInputByChannel &&
   setNormalizeOutputByChannel &&
   getNormalizeOutputByChannel &&
   getCrossValidationCostData &&
   getCostData &&
   getNumberOfEpochsTrained &&
   getEpochOfBestCost &&
```

```
            seedRandom
            ))
        {
            FreeLibrary(hDLL);
            hDLL = 0;
        }
    }
    return hDLL;
}
```

Description

Creates an instance of the neural network contained within the DLL. You must call createNetwork to get a pointer to a neural network instance before calling any other functions in the DLL protocol. All other functions require a pointer to a network instance to be passed in as a parameter. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int createNetwork(void *&aNN, int networkType)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
networkType	0 = NSRecallNetwork, 1 = NSLearningNetwork.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue = createNetwork(nn, NSLearningNetwork);
```

Return Values	Description
0	Success.
1	Invalid network type.

Possible cause

- You passed an invalid network type to the createNetwork function. Available network types include:
  - networkType = 0 = NSRecallNetwork
  - networkType = 1 = NSLearningNetwork

2	This DLL does not support learning.

Possible causes

- The DLL was created with a version of the Custom Solution Wizard that is not licensed for generating learning DLLs. Only the Developers level of the Custom Solution Wizard can generate learning DLLs.
- The NeuroSolutions breadboard that was used to generate the DLL was a recall network or it did not have one or more of the components necessary for learning.

Applies to: NSLearningNetwork, NSRecallNetwork

Description

Frees the memory associated with the neural network instance. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int destroyNetwork(void *aNN)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1)
   destroyNetwork(nn);
```

Return Values	Description
0	Success.

Applies to: NSLearningNetwork, NSRecallNetwork

Description

Retrieves the number of network inputs and network outputs and stores these values in the variables passed by reference. The number of inputs is determined by the number of processing elements contained within the input axon of the breadboard used to create the DLL. The number of outputs is determined by the number of processing elements contained within the output axon (and error criterion) of the breadboard. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int getInputOutputInfo(void *aNN, int &inputs, int &outputs)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
inputs	The number of parameters feeding into the input of the network. This value is used to determine the size of the input data array (size = exemplars * inputs) passed to the train function.
outputs	The number of parameters the network will feed out. This value is used to determine the size of the desired output data array (size = exemplars * outputs) passed to the train function.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   int inputs;
   int outputs;
   int returnValue2 = getInputOutputInfo(nn, inputs, outputs);
   if (!returnValue2)
      printf("Inputs=%d; Outputs=%d\n", inputs, outputs);
}
```

Return Values	Description
0	Success.

Applies to: **NSLearningNetwork**, **NSRecallNetwork**

Description

Loads the specified weights file (*.nsw) into the neural network instance. This will load each component's weights and the input and output normalization coefficients. Loading a weights file that contains normalization coefficients will automatically enable the use of normalization when appropriate. If desired, normalization can be disabled using the removeInputNormalization and/or removeOutputNormalization functions.

Only weights files saved by the NeuroSolutions breadboard used to generate the network DLL or by the network DLL itself should be loaded into the network DLL. During DLL generation, the Custom Solution Wizard saves the current breadboard weights. If these weights are not loaded, the network DLL will start with random initial weights. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int loadWeights(void *aNN, char *weightsPathName)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
weightsPathName	The full path name of the weights file (*.nsw) to load.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1)
   int returnValue2 = loadWeights(nn, "c:\\MyDirectory\\MyWeights.nsw");
```

Return Values	Description
0	Success.
1	Error loading weights file.

Possible causes

§ The weights file may not be present in the location passed to this function.

§ The weights file may be in use by another application.

2	Invalid weights file.

Possible causes

§ The weights for a component could not be found within the specified weights file.

§ The class name of a component may be incorrectly specified within the weights file.

§ The dimensions (number of weights) of a component specified within the weights file may not match the dimensions (number of weights) expected by the DLL for that component.

3	Invalid normalization coefficients.

Possible cause

§ The number of input or output normalization coefficients found in the specified weights file does not match the number of coefficients expected by the DLL.

Applies to: NSLearningNetwork, NSRecallNetwork

Description

Immediately saves the current state of the neural network instance (weights/normalization coefficients) into the specified weights file (*.nsw). This operation is normally performed after the network has been trained. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int saveWeights(void *aNN, char *weightsPathName)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
weightsPathName	The full path name of the NeuroSolutions weights file (*.nsw) to write to.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
   int returnValue2 = saveWeights(nn, "c:\\MyDirectory\\MyWeights.nsw");
}
```

Return Values	Description
0	Success.
1	Error saving weights file.

Possible causes

§ The location passed to this function for saving the weights may be an invalid pathName.

§ The weights file may already exist and be in use by another application.

§ Write access may be restricted for the specified location.

Applies to: NSLearningNetwork

Description

Sets the starting point (seed) for the random number generator. If the seed is set to a fixed value before the execution of any operations that use the random number generator (such as randomizeWeights , resetNetwork), the neural network will produce the same results each time it is run. By default, the random number generator is seeded using the current time when the network is initially created (createNetwork). This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int seedRandom(void *aNN, unsigned int seed)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
seed	The starting point for the random number generator.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   seedRandom(nn, 1000);
   resetNetwork(nn);
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   int returnValue2 = train(nn,1000,4,inputData,desiredData,4,NULL,NULL);
}
```

Return Values	Description
0	Success.

Applies to: NSLearningNetwork, NSRecallNetwork

Description

Randomizes the weights of the neural network instance without resetting the network. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int randomizeWeights(void *aNN)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   float CVInputData[2*4] = {0.1,0.1,0.1,0.9,0.9,0.1,0.9,0.9};
   float CVDesiredData[1*4] = {0.1,0.9,0.9,0.1};
   setSaveBestWeightsEnabled(nn, true);
   setBestWeightsPathName(nn, "c:\\MyDirectory\\BestWeights.nsw");
   setSaveBestWeightsForTraining(nn, false);
   for (int i=0; i<10; i++) { //Saves one best weights file for all 10 runs
      train(nn,1000,4,inputData,desiredData,4, CVInputData, CVDesiredData);
      randomizeWeights(nn);
   }
}
```

Return Values	Description
0	Success.

Applies to: NSLearningNetwork, NSRecallNetwork

Description

Randomizes the weights of the neural network instance and resets the network. Resetting the network will reset the best cost (see getBestCost) to 1E+009. For hybrid networks, this function will switch the network back to the unsupervised training mode. In most cases, you will want to use resetNetwork over randomizeWeights when starting a new training session. This function returns an integer value indicating whether the function succeeded or failed.

Note: Calling the resetNetwork function is equivalent to clicking the reset button in NeuroSolutions.

Prototype

int resetNetwork(void *aNN)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1)
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   float CVInputData[2*4] = {0.1,0.1,0.1,0.9,0.9,0.1,0.9,0.9};
   float CVDesiredData[1*4] = {0.1,0.9,0.9,0.1};
   char bestWeightsPath[512];
   setSaveBestWeightsEnabled(nn, true);
   setSaveBestWeightsForTraining(nn, false);
   for (int i=0; i<10; i++) { //Saves one best weights file for each of the 10 runs
      sprintf(bestWeightsPath, "c:\\MyDirectory\\BestWeights%d.nsw", i)
      setBestWeightsPathName(nn, bestWeightsPath);
      train(nn,1000,4,inputData,desiredData,4, CVInputData, CVDesiredData);
      resetNetwork(nn);
   }
}
```

Return Values	Description
0	Success.

Applies to: NSLearningNetwork

Description

Trains the neural network instance for the specified number of epochs. Networks can be trained with or without cross validation (see the setCrossValidationEnabled function). By default, cross validation is disabled. The training and cross validation data are passed directly to the train function as floating point arrays.

The weights file saved during DLL generation can be loaded (using the loadWeights method) before calling the train function in order to start the training of the neural network DLL at the same state the as the original NeuroSolutions breadboard (same weights and normalization coefficients). If the weights file is not loaded, the weights will start at random initial values and the normalization coefficients will be calculated based on the training data (if the original NeuroSolutions breadboard had normalization enabled).

A network can be trained more than once. If the weights are not loaded or randomized after a training session, the execution of the train method for a second time will result in the network starting off where the first training session left off. This function returns an integer value indicating whether the function succeeded or failed.

Note: Hybrid networks must be trained long enough for the data flow to be turned on between the unsupervised portion and the supervised portion. Otherwise, the getResponse function will return all zeros.

Note: Transmitters and schedulers will affect the training process just as they do in NeuroSolutions. For example, if you had a transmitter (on the NeuroSolutions breadboard used to generate the network DLL) that was set to stop the network when the cost (error) decreased below .001, this transmitter will stop the network when this error is reached, just as it does within NeuroSolutions.

Prototype

```
int train(void *aNN, int epochs, int exemplars, float *inputData, float *desiredData, int CVExemplars, float
        *CVInputData, float *CVDesiredData)
```

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
epochs	The number of times the data is presented to the network.
exemplars	The number of patterns (rows) of training data being injected into the network.
inputData	A floating point array containing the training input data of the network. The data is ordered as follows: inputData[0] = Exemplar(0), Input(0) inputData[1] = Exemplar(0), Input(1) inputData[N-1] = Exemplar(0), Input(N-1) inputData[N] = Exemplar(1), Input(0) inputData[M*N -1] = Exemplar(M-1), Input(N-1) where N = number of inputs of the network M = number of exemplars in the data array
desiredData	A floating point array containing the training desired output data of the network. The data is ordered as follows: desiredData[0] = Exemplar(0), Output(0) desiredData[1] = Exemplar(0), Output(1) desiredData[N-1] = Exemplar(0), Output (N-1) desiredData[N] = Exemplar(1), Output (0) desiredData[M*N -1] = Exemplar(M-1), Output (N-1) where

N = number of outputs of the network

M = number of exemplars in the data array

CVExemplars — The number of patterns (rows) of cross validation data injected into the network. Set to 0 if there is no cross validation data.

CVInputData — A floating point array containing the cross validation input data of the network. Set to NULL if there is no cross validation data. The data is ordered as follows:

CVInputData[0] = CVExemplar(0), Input(0)
CVInputData[1] = CVExemplar(0), Input(1)
CVInputData[N-1] = CVExemplar(0), Input(N-1)
CVInputData[N] = CVExemplar(1), Input(0)
CVInputData[M*N -1] = CVExemplar(M-1), Input(N-1)

where

N = number of inputs of the network

M = number of cross validation exemplars in the data array

CVDesiredData — A floating point array containing the cross validation desired output data of the network. Set to NULL if there is no cross validation data. The data is ordered as follows:

CVDesiredData[0] = CVExemplar(0), Output(0)
CVDesiredData[1] = CVExemplar(0), Output(1)
CVDesiredData[N-1] = CVExemplar(0), Output (N-1)
CVDesiredData[N] = CVExemplar(1), Output (0)
CVDesiredData[M*N -1] = CVExemplar(M-1), Output (N-1)

where

N = number of outputs of the network

M = number of cross validation exemplars in the data array

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   float CVInputData[2*4] = {0.1,0.1,0.1,0.9,0.9,0.1,0.9,0.9};
   float CVDesiredData[1*4] = {0.1,0.9,0.9,0.1};
   setCrossValidationEnabled(nn, true);
   int returnValue2 = train(nn,1000,4,inputData,desiredData,4,CVInputData,
      CVDesiredData);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called train for a recall network. This function is only available for learning networks.

2	Error saving weights file.

Possible causes

§ The location specified for saving the best weights (getBestWeightsPathName) may be invalid.

§ The weights file may already exist and be in use by another application.

§ Write access may be restricted for the directory specified in getBestWeightsPathName.

3 Invalid number of exemplars in the training dataset.

Possible cause

§ For a dynamic network, the number of exemplars in the training dataset must be evenly divisible be the Samples/Exemplar setting of the original NeuroSolutions breadboard used for generating the DLL. This setting can be found within the DynamicControl inspector.

4 Function bypassed.

Possible causes

§ A DLL overridden component has returned false from the fireIsReady function of the Breadboard Sub-Protocol causing the execution of this function to be bypassed.

§ The number of epochs being passed to the train function may be less than or equal to zero.

Applies to: NSLearningNetwork, NSRecallNetwork

Description

Injects the specified input data array into the neural network and stores the resulting network output into the specified output data array. This operation does not modify the state of the network (i.e., the weights remain fixed). Before using this method you should either train the neural network or load a set of weights that were saved after a training session. Otherwise, the network output will be random. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int getResponse(void *aNN, int exemplars, float *inputData, float *outputData)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
exemplars	The number of patterns (rows) of data being injected into the network.
inputData	A floating point array containing the input data of the network. The data is ordered as follows: inputData[0] = Exemplar(0), Input(0) inputData[1] = Exemplar(0), Input(1) inputData[N-1] = Exemplar(0), Input(N-1) inputData[N] = Exemplar(1), Input(0) inputData[M*N-1] = Exemplar(M-1), Input(N-1) where N = number of inputs of the network M = number of exemplars in the data array
outputData	A floating point array that the function will use to write the network output into. The data is ordered as follows: outputData[0] = Exemplar(0), Output(0) outputData[1] = Exemplar(0), Output (1) outputData[N-1] = Exemplar(0), Output (N-1) outputData[N] = Exemplar(1), Output (0) outputData[M*N-1] = Exemplar(M-1), Output (N-1) where N = number of outputs of the network M = number of exemplars in the data array

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   loadWeights(nn, "c:\\MyDirectory\\MyWeights.nsw");
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float outputData[1*4];
   int returnValue2 = nn.getResponse(nn,4,inputData,outputData);
   if (!returnValue2)
```

```
        printf("Network output: %f %f %f %f\n", outputData[0], outputData[1],
           outputData[2], outputData[3]);
}
```

Return Values	Description
0	Success.
1	Function bypassed.

Possible cause

§ A DLL overridden component has returned False from the fireIsReady function of the Breadboard Sub-Protocol causing the execution of this function to be bypassed.

Applies to: NSLearningNetwork, NSRecallNetwork

Description

Performs the sensitivity analysis function on the network with the specified input data and returns the results in an array. Sensitivity analysis is a method for extracting the cause and effect relationship between the inputs and outputs of the network. The network learning is disabled during this operation such that the network weights are not affected. The basic idea is that the inputs to the network are shifted slightly (by the amount specified in the dither parameter) and the corresponding change in the output is reported as a raw difference for each input-output combination. Therefore, the size of the array that the function writes to is inputs*outputs. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int getSensitivity(void *aNN, int exemplars, float *inputData, float *&sensitivityData, float dither)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
exemplars	The number of patterns (rows) of data being injected into the network.
inputData	A floating point array containing the input data of the network. The data is ordered as follows: inputData[0] = Exemplar(0), Input(0) inputData[1] = Exemplar(0), Input(1) inputData[N-1] = Exemplar(0), Input(N-1) inputData[N] = Exemplar(1), Input(0) inputData[M*N -1] = Exemplar(M-1), Input(N-1) where N = number of inputs of the network M = number of exemplars in the data array
sensitivityData	A floating point array that the function will use to write the sensitivity values into. The data is ordered as follows: sensitivityData[0] = Output(0), Input(0) sensitivityData[1] = Output(0), Input(1) sensitivityData[N-1] = Output(0), Input(N-1) sensitivityData[N] = Output(1), Input(0) sensitivityData[M*N-1] = Output(M-1), Input(N-1) where N = number of inputs of the network M = number of outputs of the network
dither	The amount that the network inputs are changed from their original value during the sensitivity analysis operation.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   loadWeights(nn, "c:\\MyDirectory\\MyWeights.nsw");
```

```
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float sensitivityData[2*1];
   int returnValue2 = nn.getSensitivity(nn,4,inputData, sensitivityData,0.1f);
   if (!returnValue2)
      printf("Sensitivity Values: %f %f\n", sensitivityData[0], sensitivityData[1]);
}
```

Return Values	Description
0	Success.
1	Function bypassed.

Possible cause

§ A DLL overridden component has returned False from the fireIsReady function of the Breadboard Sub-Protocol causing the execution of this function to be bypassed.

Applies to: NSLearningNetwork

Description

Deactivates input normalization and disables the automatic computation of input normalization coefficients during training. To reactivate input normalization, either load a weights file containing input normalization coefficients or call the setAutoComputeInputNormCoeff function (passing it true) then train the network. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int removeInputNormalization(void *aNN)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   loadWeights(nn, "C:\MyDirectory\MyWeights.nsw");
   removeInputNormalization(nn);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Deactivates output normalization and disables the automatic computation of output normalization coefficients during training. To reactivate output normalization, either load a weights file containing output normalization coefficients or call the setAutoComputeOutputNormCoeff function (passing it true) then train the network. This function returns an integer value indicating whether the function succeeded or failed.

Prototype

int removeOutputNormalization(void *aNN)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   loadWeights(nn, "C:\MyDirectory\MyWeights.nsw");
   removeOutputNormalization(nn);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets a flag indicating whether or not a cross validation dataset will be fed through the network and evaluated after each epoch of training. Cross validation is used to improve the generalization of the model produced. Saving the best weights based on the lowest cost of the cross validation dataset (see the setSaveBestWeightsEnabled, setBestWeightsPathName, and setSaveBestWeightsForTraining functions) reduces overfitting of the training data and usually results in a model that generalizes better than one that does not use cross validation. Both of these functions return integer values indicating whether they succeeded or failed.

Note: This flag is always initialized to false when the DLL is generated, regardless of whether or not the corresponding NeuroSolutions breadboard was configured to use cross validation.

Note: If cross validation is enabled, it will be performed at the end of each epoch regardless of the value of the *Epochs/Cross Val.* setting within the *Control* component on the corresponding NeuroSolutions breadboard.

Prototypes

int setCrossValidationEnabled(void *aNN, bool crossValidationEnabled)
int getCrossValidationEnabled(void *aNN, bool &crossValidationEnabled)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
crossValidationEnabled	Flag that indicates whether or not cross validation will be used during training.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   float CVInputData[2*4] = {0.1,0.1,0.1,0.9,0.9,0.1,0.9,0.9};
   float CVDesiredData[1*4] = {0.1,0.9,0.9,0.1};
   setSaveBestWeightsEnabled(nn, true);
   setBestWeightsPathName(nn, "c:\\MyDirectory\\BestWeights.nsw");
   setSaveBestWeightsForTraining(nn, false);
   bool crossValEnabled;
   getCrossValidationEnabled(nn, crossValEnabled);
   if (!crossValEnabled)
      setCrossValidationEnabled(nn, true);
   returnValue2 = train(nn,1000,4,inputData,desiredData, 4, CVInputData,
      CVDesiredData);
}
```

Return Values	Description
0	Success.

1 Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets a flag indicating whether or not the train function will automatically save the network weights during the epoch in which the lowest error is achieved. The setSaveBestWeightsForTraining function controls which type of error is used: training or cross validation. The weights are saved to the file specified by the setBestWeightsPathName function. Both of these functions return integer values indicating whether they succeeded or failed.

Note: This flag is always initialized to false when the DLL is generated, regardless of whether the corresponding NeuroSolutions breadboard was configured for saving the best weights.

Note: If this flag is set to true, a valid bestWeightsPathName must be defined before calling the train function.

Prototypes

int setSaveBestWeightsEnabled(void *aNN, bool saveBestWeightsEnabled)
int getSaveBestWeightsEnabled(void *aNN, bool &saveBestWeightsEnabled)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
saveBestWeightsEnabled	Flag that indicates whether or not the best weights will be saved during training.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setBestWeightsPathName(nn, "c:\\MyDirectory\\BestWeights.nsw");
   bool saveBest;
   getSaveBestWeightsEnabled(nn, saveBest);
   if (!saveBest)
      setSaveBestWeightsEnabled(nn, true);
   setSaveBestWeightsForTraining(nn, true);
   int returnValue2 = train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets a flag indicating whether the best weights saved during training correspond to the minimum error in the training dataset (flag equals true) or to the minimum error in the cross validation dataset (flag equals false). The value of this flag also affects whether the best cost returned by the getbestCost function corresponds to the training dataset or to the cross validation dataset. Both of these functions return integer values indicating whether they succeeded or failed.

When the saving of the best weights is enabled (see setSaveBestWeightsEnabled)

Note: In order for the best weights to be saved during training, the saving of the best weights must be enabled (see setSaveBestWeightsEnabled) and a valid bestWeightsPathName must be defined (see setBestWeightsPathName).

Prototypes

int setSaveBestWeightsForTraining(void *aNN, bool saveBestWeightsForTraining)

int getSaveBestWeightsForTraining(void *aNN, bool &saveBestWeightsForTraining)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
saveBestWeightsForTraining	Flag that indicates whether the best weights will be saved for the best cost of the training dataset or for the best cost of the cross validation dataset.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setBestWeightsPathName(nn, "c:\\MyDirectory\\BestWeights.nsw");
   setSaveBestWeightsEnabled(nn, true);
   bool saveForTrain;
   getSaveBestWeightsForTraining(nn, saveForTrain);
   if (!saveForTrain)
      setSaveBestWeightsForTraining(nn, true);
   int returnValue2 = train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets the full path name of the weights file (*.nsw) that the train procedure will use when saving the best weights. This file stores the network weights of the training epoch that achieved the lowest cost. Note that for the best weights to be saved, the setSaveBestWeightsEnabled function must be set to true before training.

This getBestWeightsPathName function has two modes: 1) retrieving the length of the path name and 2) retrieving the path name itself. If the bestWeightsPathName parameter is NULL, then only the pathNameLength is returned. If this parameter is not NULL, then it writes the path name string into this buffer. It is important to note that the bufferLength must be at least one more than the pathNameLength (to account for the end-of-string character).

Both of these functions return an integer value indicating whether the function succeeded or failed.

Prototypes

int setBestWeightsPathName(void *aNN, char *bestWeightsPathName)

int getBestWeightsPathName(void *aNN, char *bestWeightsPathName, int bufferLength, int &pathNameLength)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
bestWeightsPathName	A null-terminated character string containing the path and name of a file to use for saving the best weights (*.nsw).
bufferLength	The number of characters allocated to the buffer pointed to by bestWeightsPathName. This must be at least one greater than the pathNameLength (to account for the end-of-string character).
pathNameLength	The string length of the weights file path that will be written into the bestWeightsPathName buffer.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   setBestWeightsPathName(nn, "c:\\MyDirectory\\BestWeights.nsw");
   int pathNameLength;
   char *weightsPathName;
   getBestWeightsPathName(nn, NULL, 0, pathNameLength);
   weightsPathName = (char *)malloc((pathNameLength + 1) * sizeof(char));
   getBestWeightsPathName(nn, weightsPathName, pathNameLength+1, pathNameLength);
   printf("Best Weights PathName = %s", weightsPathName);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

2 Buffer size too small.

Possible cause

§ The buffer length for holding the bestWeightsPathName was too small. It must be at least the length of the bestWeightsPathName string plus 1 character for the null terminator.

Applies to: NSLearningNetwork

Description

Sets/Gets the best cost for the neural network. The initial value of the best cost is 1E+009. Once the network has been trained, the best cost will represent the minimum error obtained for the training dataset if the network was trained without cross validation or if the best weights were set to be saved for the training dataset (see setCrossValidationEnabled and setSaveBestWeightsForTraining). If the network was trained with cross validation and the best weights were set to be saved for the cross validation dataset, the best cost will represent the minimum error obtained for the cross validation dataset.

The best cost can also be set manually for use as a threshold that must be bettered before the best weights will be saved during training (see setSaveBestWeightsEnabled). The best weights will only be saved and the best cost will only be updated if the best cost during training falls below the current best cost value.

Both of these functions return integer values indicating whether they succeeded or failed.

Note: The best cost is reset to 1E+009 when the resetNetwork function is called.

Note: The cost within the generated DLL is computed once every epoch regardless of the value of the *Average cost for:* setting within the *ErrorCriteria* component on the corresponding NeuroSolutions breadboard.

Prototypes

int setBestCost(void *aNN, float bestCost)
int getBestCost(void *aNN, float &bestCost)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
bestCost	The best cost.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setBestWeightsPathName(nn, "c:\\MyDirectory\\BestWeights.nsw");
   setSaveBestWeightsEnabled(nn, true);
   setSaveBestWeightsForTraining(nn, true);
   setBestCost(nn, 1.0e9f);
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
   float bestCost;
   getBestCost(nn, bestCost);
   printf("Best Cost = %f\n", bestCost);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Gets the epoch number during which the neural network achieved its best cost (minimum error). Once the network has been trained, the epoch number returned will correspond to the best cost obtained for the training dataset if the network was trained without cross validation or if the best weights were set to be saved for the training dataset (see setCrossValidationEnabled and setSaveBestWeightsForTraining). If the network was trained with cross validation and the best weights were set to be saved for the cross validation dataset, the epoch number returned will correspond to the best cost obtained for the cross validation dataset.

Until the network has been trained, the epoch of the best cost will be 0. Furthermore, calling the resetNetwork function will reset the epoch of the best cost to 0.

This function returns an integer value indicating whether it succeeded or failed.

Note: The corresponding best cost can be obtained by calling the getbestCost function.

Prototype

int getEpochOfBestCost(void *aNN, int &epochOfBestCost)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
epochOfBestCost	The epoch of the best cost.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   loadWeights(nn, "C:\\MyDirectory\\MyWeights.nsw");
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
   int epochNumber;
   getEpochOfBestCost(nn, epochNumber);
   printf("Epoch of Best Cost = %d\n", epochNumber);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Gets the training learning curve data for the most recent training run. The cost (error) for the training dataset during each epoch the network was run is written to the memory location pointed to by the *costData* parameter. This memory must be allocated before calling this function with a size equal to the number of epoch trained (see the getNumberOfEpochsTrained function).

Note: The number of epochs actually run may be different than the number passed to the train function. This can occur when the neural network is configured to stop after reaching a certain cost, once the cross validation error begins to rise, etc. This is done through the use of transmitters on the original NeuroSolutions breadboard.

Prototype

int getCostData(void *aNN, float *costData)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
costData	Pointer to the memory location to use for writing the training learning curve data.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   loadWeights(nn, "C:\\MyDirectory\\MyWeights.nsw");
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
   int numberOfEpochsTrained;
   getNumberOfEpochsTrained(nn, numberOfEpochsTrained);
   float *costData = new float[numberOfEpochsTrained];
   getCostData(nn, costData);
   for (int i = 0; i < numberOfEpochsTrained; i++)
      printf("%f\n", costData[i]);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

2	Network must be trained before attempting to retrieve the cost.

Possible cause

§ The network has not been trained or the network was reset after it was trained.

Applies to: NSLearningNetwork

Description

Gets the cross validation learning curve data for the most recent training run. The cost (error) for the cross validation dataset during each epoch the network was run is written to the memory location pointed to by the *CVCostData* parameter. This memory must be allocated before calling this function with a size equal to the number of epoch trained (see the getNumberOfEpochsTrained function).

Note: The number of epochs actually run may be different than the number passed to the train function. This can occur when the neural network is configured to stop after reaching a certain cost, once the cross validation error begins to rise, etc. This is done through the use of transmitters on the original NeuroSolutions breadboard.

Note: The network must have been trained with cross validation in order to retrieve the cross validation cost data.

Prototype

int getCrossValidationCostData(void *aNN, float *CVCostData)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
CVCostData	Pointer to the memory location to use for writing the cross validation learning curve data.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   float CVInputData[2*4] = {0.1,0.1,0.1,0.9,0.9,0.1,0.9,0.9};
   float CVDesiredData[1*4] = {0.1,0.9,0.9,0.1};
   setCrossValidationEnabled(nn, true);
   loadWeights(nn, "C:\\MyDirectory\\MyWeights.nsw");
   train(nn,1000,4,inputData,desiredData, 4, CVInputData, CVDesiredData);
   int numberOfEpochsTrained;
   getNumberOfEpochsTrained(nn, numberOfEpochsTrained);
   float *CVCostData = new float[numberOfEpochsTrained];
   getCrossValidationCostData(nn, CVCostData);
   for (int i = 0; i < numberOfEpochsTrained; i++)
      printf("%f\n", CVCostData[i]);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

2 Network must be trained (with cross validation) before attempting to retrieve the cross validation cost.

Possible cause

§ Either the network has not been trained, it was reset after it was trained, or it was not trained with cross validation.

Applies to: NSLearningNetwork

Description

Gets the number of epochs completed during the most recent training run. The result will be 0 until the network has been trained. Also, calling the resetNetwork function will reset the number of epochs trained to 0.

This function is used to get the actual number of epochs completed since this value may be different from the number passed to the train function. This can happen if the neural network was configured to stop after reaching a certain cost, once the cross validation error begins to rise, etc.

Prototype

int getNumberOfEpochsTrained(void *aNN, int &numberOfEpochsTrained)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
numberOfEpochsTrained	The number of epochs trained during the latest training run.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   loadWeights(nn, "C:\\MyDirectory\\MyWeights.nsw");
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
   int numberOfEpochsTrained;
   getNumberOfEpochsTrained(nn, numberOfEpochsTrained);
   printf("Number of Epochs Trained = %d\n", numberOfEpochsTrained);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets a flag indicating whether or not the input normalization coefficients are automatically computed at the beginning of each training run. If this flag is true, the training input data will be used to compute the input normalization coefficients at the beginning of a call to the train function. The normalization coefficients are computed according to the input norm min, input norm max, and normalize input by channel settings (see the setInputNormMin, setInputNormMax, and setNormalizeInputByChannel functions)

If this flag is false, the input normalization coefficients are not computed at the beginning of a training run. However, this does not mean that input normalization will not be performed. Once input normalization is activated, it will be performed until it is deactivated, either by calling removeInputNormalization or by loading a weights file that doesn't contain input normalization coefficients.

The initial value of this flag corresponds to the state of the "Normalize" switch on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the input "File" component inspector). This switch will usually be on and, correspondingly, the flag will usually have a default value of true. If the "Normalize" switch was off, the input normalization was marked as "Read Only"(see the "Data Sets" tab of the input "File" component inspector), or no input "File" component was found when generating the DLL, the flag will have a default value of false.

Note: Input normalization is activated either by loading a weights file containing input normalization coefficients or by setting this flag to true, followed by a training run.

Prototype

int setAutoComputeInputNormCoeff(void *aNN, bool autoComputeInputNormCoeff)

int getAutoComputeInputNormCoeff(void *aNN, bool &autoComputeInputNormCoeff)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
autoComputeInputNormCoeff	Flag that indicates whether or not the input normalization coefficients are automatically computed at the beginning of each training run.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setInputNormMin(nn, -1.0f);
   setInputNormMax(nn, 1.0f);
   setNormalizeInputByChannel(nn, true);
   bool autoComputeInputNormCoeff;
   getAutoComputeInputNormCoeff(nn, autoComputeInputNormCoeff);
   if (!autoComputeInputNormCoeff)
      setAutoComputeInputNormCoeff(nn, true);
```

```
    train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks. **Possible cause** § You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets the lower bound used for calculating the input normalization coefficients. The coefficients are calculated (using the training dataset) such that the minimum training input value after normalization is equal to this lower bound.

The initial value of the input norm min corresponds to the value of the "Lower" setting on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the input "File" component inspector). If no input "File" component was found when generating the DLL, the input norm min will have a default value of -1.

See the setAutoComputeInputNormCoeff function for more information on input normalization.

Prototype

int setInputNormMin(void *aNN, float inputNormMin)

int getInputNormMin(void *aNN, float &inputNormMin)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
inputNormMin	The lower bound used for calculating the input normalization coefficients.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   float inputNormMin;
   getInputNormMin(nn, inputNormMin)
   if (inputNormMin != -1.0f)
      setInputNormMin(nn, -1.0f);
   setInputNormMax(nn, 1.0f);
   setNormalizeInputByChannel(nn, true);
   setAutoComputeInputNormCoeff(nn, true);
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets the upper bound used for calculating the input normalization coefficients. The coefficients are calculated (using the training dataset) such that the maximum training input value after normalization is equal to this upper bound.

The initial value of the input norm max corresponds to the value of the "Upper" setting on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the input "File" component inspector). If no input "File" component was found when generating the DLL, the input norm max will have a default value of 1.

See the setAutoComputeInputNormCoeff function for more information on input normalization.

Prototype

int setInputNormMax(void *aNN, float inputNormMax)

int getInputNormMax(void *aNN, float &inputNormMax)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
inputNormMax	The upper bound used for calculating the input normalization coefficients.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setInputNormMin(nn, -1.0f);
   float inputNormMax;
   getInputNormMax(nn, inputNormMax)
   if (inputNormMax != 1.0f)
      setInputNormMax(nn, 1.0f);
   setNormalizeInputByChannel(nn, true);
   setAutoComputeInputNormCoeff(nn, true);
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets a flag indicating whether the input normalization coefficients are calculated on a channel-by-channel basis (each input column considered individually) or across the entire training input dataset.

The initial value of this flag corresponds to the state of the "By Channel" switch on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the input "File" component inspector). If no input "File" component was found when generating the DLL, the normalize input by channel setting will have a default value of true.

See the setAutoComputeInputNormCoeff function for more information on input normalization.

Prototype

int setNormalizeInputByChannel(void *aNN, bool normalizeInputByChannel)

int getNormalizeInputByChannel(void *aNN, bool &normalizeInputByChannel)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
normalizeInputByChannel	Flag that indicates whether the input normalization coefficients are calculated on a channel-by-channel basis or across the entire training input dataset.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setInputNormMin(nn, -1.0f);
   setInputNormMax(nn, 1.0f);
   bool normalizeInputByChannel;
   getNormalizeInputByChannel(nn, normalizeInputByChannel);
   if (!normalizeInputByChannel)
      setNormalizeInputByChannel(nn, true);
   setAutoComputeInputNormCoeff(nn, true);
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets a flag indicating whether or not the output normalization coefficients are automatically computed at the beginning of each training run. If this flag is true, the training desired data will be used to compute the output normalization coefficients at the beginning of a call to the train function. The normalization coefficients are computed according to the output norm min, output norm max, and normalize output by channel settings (see the setOutputNormMin, setOutputNormMax, and setNormalizeOutputByChannel functions)

If this flag is false, the output normalization coefficients are not computed at the beginning of a training run. However, this does not mean that output normalization will not be performed. Once output normalization is activated, it will be performed until it is deactivated, either by calling removeOutputNormalization or by loading a weights file that doesn't contain output normalization coefficients.

The initial value of this flag corresponds to the state of the "Normalize" switch on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the desired "File" component inspector). This switch will usually be on and, correspondingly, the flag will usually have a default value of true. If the "Normalize" switch was off, the desired normalization was marked as "Read Only"(see the "Data Sets" tab of the desired "File" component inspector), or no desired "File" component was found when generating the DLL, the flag will have a default value of false.

Note: Output normalization is activated either by loading a weights file containing output normalization coefficients or by setting this flag to true, followed by a training run.

Prototype

int setAutoComputeOutputNormCoeff(void *aNN, bool autoComputeOutputNormCoeff)

int getAutoComputeOutputNormCoeff(void *aNN, bool &autoComputeOutputNormCoeff)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
autoComputeOutputNormCoeff	Flag that indicates whether or not the output normalization coefficients are automatically computed at the beginning of each training run.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setOutputNormMin(nn, -1.0f);
   setOutputNormMax(nn, 1.0f);
   setNormalizeOutputByChannel(nn, true);
   bool autoComputeOutputNormCoeff;
   getAutoComputeOutputNormCoeff(nn, autoComputeOutputNormCoeff);
   if (!autoComputeOutputNormCoeff)
      setAutoComputeOutputNormCoeff(nn, true);
```

```
    train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks. **Possible cause** § You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets the lower bound used for calculating the output normalization coefficients. The coefficients are calculated (using the training dataset) such that the minimum training desired value after normalization is equal to this lower bound.

The initial value of the output norm min corresponds to the value of the "Lower" setting on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the desired "File" component inspector). If no desired "File" component was found when generating the DLL, the output norm min will have a default value of -1.

See the setAutoComputeOutputNormCoeff function for more information on output normalization.

Prototype

int setOutputNormMin(void *aNN, float outputNormMin)

int getOutputNormMin(void *aNN, float &outputNormMin)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
outputNormMin	The lower bound used for calculating the output normalization coefficients.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   float outputNormMin;
   getOutputNormMin(nn, outputNormMin)
   if (outputNormMin != -1.0f)
      setOutputNormMin(nn, -1.0f);
   setOutputNormMax(nn, 1.0f);
   setNormalizeOutputByChannel(nn, true);
   setAutoComputeOutputNormCoeff(nn, true);
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets the upper bound used for calculating the output normalization coefficients. The coefficients are calculated (using the training dataset) such that the maximum training desired value after normalization is equal to this upper bound.

The initial value of the output norm max corresponds to the value of the "Upper" setting on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the desired "File" component inspector). If no desired "File" component was found when generating the DLL, the output norm max will have a default value of 1.

See the setAutoComputeOutputNormCoeff function for more information on output normalization.

Prototype

int setOutputNormMax(void *aNN, float outputNormMax)

int getOutputNormMax(void *aNN, float &outputNormMax)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
outputNormMax	The upper bound used for calculating the output normalization coefficients.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setOutputNormMin(nn, -1.0f);
   float outputNormMax;
   getOutputNormMax(nn, outputNormMax)
   if (outputNormMax != 1.0f)
      setOutputNormMax(nn, 1.0f);
   setNormalizeOutputByChannel(nn, true);
   setAutoComputeOutputNormCoeff(nn, true);
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

Applies to: NSLearningNetwork

Description

Sets/Gets a flag indicating whether the output normalization coefficients are calculated on a channel-by-channel basis (each output column considered individually) or across the entire training desired dataset.

The initial value of this flag corresponds to the state of the "By Channel" switch on the NeuroSolutions breadboard used to generate the neural network DLL (see the "Stream" tab of the desired "File" component inspector). If no desired "File" component was found when generating the DLL, the normalize output by channel setting will have a default value of true.

See the setAutoComputeOutputNormCoeff function for more information on output normalization.

Prototype

int setNormalizeOutputByChannel(void *aNN, bool normalizeOutputByChannel)

int getNormalizeOutputByChannel(void *aNN, bool &normalizeOutputByChannel)

Parameter	Description
aNN	Pointer to the neural network instance set by the createNetwork function.
normalizeOutputByChannel	Flag that indicates whether the output normalization coefficients are calculated on a channel-by-channel basis or across the entire training desired dataset.

Example:

This example assumes that the neural network DLL has already been loaded. See the C++ Code Needed to Load the Generated DLL topic for instructions on how to do this.

```
enum {NSRecallNetwork, NSLearningNetwork};
void *nn;
int returnValue1 = createNetwork(nn, NSLearningNetwork);
if (!returnValue1) {
   float inputData[2*4] = {0,0,0,1,1,0,1,1};
   float desiredData[1*4] = {0,1,1,0};
   setOutputNormMin(nn, -1.0f);
   setOutputNormMax(nn, 1.0f);
   bool normalizeOutputByChannel;
   getNormalizeOutputByChannel(nn, normalizeOutputByChannel);
   if (!normalizeOutputByChannel)
      setNormalizeOutputByChannel(nn, true);
   setAutoComputeOutputNormCoeff(nn, true);
   train(nn,1000,4,inputData,desiredData,0,NULL,NULL);
}
```

Return Values	Description
0	Success.
1	Function unavailable for recall networks.

Possible cause

§ You called this function for a recall network. This function is only available for learning networks.

The easiest way to build a C++ application for using a neural network DLL is to start with a project shell. A Visual C++ project shell can be generated automatically by choosing the *Visual C++ 5.0/6.0* project type on the Choose Project Type Panel during the creation of the neural network DLL. This will create a sample application (with source code) that will load in your DLL and allow you to train the network and get the network's output.

If you would rather create a Visual C++ application from scratch, this topic will demonstrate how to do this. Simply follow the step-by-step instructions below.

Note: A completed version of this example can be found in the directory: *[NSDirectory]\Wizards\CustomSolutionWizard\Examples\VCPPExample* where *[NSDirectory]* is the directory where NeuroSolutions was installed (*C:\Program Files\NeuroSolutions 4* by default).

Step 1:

Build a multilayer perceptron (MLP) within NeuroSolutions using the exclusive-or data for the input and desired output.

Inputs		**Desired Output**
X	Y	Z
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Step 2:

Use the Custom Solution Wizard to generate a neural network DLL for this NeuroSolutions breadboard.

Step 3:

Launch Visual C++.

Step 4:

Create an empty Console Application.

Step 5:

Create a new source file named *VCPPExample.cpp* and a new header file named *VCPPExample.h*.

Step 6:

Insert the code from the Visual C++ Example Code topic into the source file and the code from the C++ Code Needed to Load the Generated DLL topic into the header file.

Step 7:

Change the string passed to the *LoadDLLFunctions* function to reflect the location of the neural network DLL you created in **Step 2.**

Step 8:

Change the call to the *loadWeights* function to reflect the location of the weights file you created in **Step 2.**

Step 9:

Build and run the program.

This is the source code to use for **Step 6** of the Visual C++ Example. The code first loads the neural network DLL and its corresponding functions. Next, the input and desired data are assigned to arrays and the network is trained for a user-specified number of epochs. Finally, the output of the network is retrieved and displayed.

```
//VCPPExample.cpp
#include <iostream>
#include "VCPPExample.h"

enum {NSRecallNetwork, NSLearningNetwork};

int main()
{
   HINSTANCE hDLL = LoadDLLFunctions("C:\\MyDirectory\\MyDLL.dll");
   if (hDLL)
   {
      float *inputData = new float[8];
      inputData[0] = -1;
      inputData[1] = -1;
      inputData[2] = -1;
      inputData[3] = 1;
      inputData[4] = 1;
      inputData[5] = -1;
      inputData[6] = 1;
      inputData[7] = 1;

      float *desiredData = new float[4];
      desiredData[0] = -1;
      desiredData[1] = 1;
      desiredData[2] = 1;
      desiredData[3] = -1;

      void *nn;
      createNetwork(nn, NSLearningNetwork);

      loadWeights(nn, "C:\\MyDirectory\\MyWeights.nsw");

      std::cout << "Epochs: ";
      int epochs;
      std::cin >> epochs;

      train(nn, epochs, 4, inputData, desiredData, 0, NULL, NULL);

      float *outputData = new float[4];
      getResponse(nn, 4, inputData, outputData);

      std::cout << "\nNetwork Output\n";
      for (int i = 0; i < 4; i++)
         std::cout << outputData[i] << "\n";
```

```
      delete inputData;
      delete desiredData;
      destroyNetwork(nn);
      FreeLibrary(hDLL);
   }
   else
      std::cout << “ERROR LOADING DLL”;

   std::cout << “\n\nType any character then click Enter to exit!\n”;
   char aChar;
   std::cin >> aChar;

   return 0;
}
```

If the Custom Solution Wizard encounters a problem creating the neural network DLL, you will see a *DLL creation failed* error. This error can happen for a number of reasons:

- Your breadboard may not be supported by the Custom Solution Wizard (see Limitations of the Custom Solution Wizard).
- The Custom Solution Wizard may not be able to locate Visual C++ 5.0 or 6.0 on your computer.
- Your version of Visual C++ 5.0 or 6.0 may not be set up correctly.
- The Custom Solution Wizard may not be able to locate the required NeuroSolutions library files (msvc50.lib and msvc50.h if compiling with Visual C++ 5.0 or msvc60.lib and msvc60.h if compiling with Visual C++ 6.0). These files should be located in the *CodeGen* directory -- one level under the directory in which NeuroSolutions was installed.
- One or more of the files that the Custom Solution Wizard needs to create may already exist on your file system and be in use by another application.
- The Custom Solution Wizard may not have permission to write to the directory you chose for the *Project Location*.

When you distribute an application based on the NeuroSolutions Object Library, you must also include the *NeuroSolutionsOL.dll* file. This file is located in your *Windows\System* or *Winnt\System32* directory (depending upon your operating system). You must also register this file on the end user's machine. The best way to do this is to have your setup program call *regsvr32.exe*. This executable is located in the *Windows\System* or *Winnt\System32* directory and is put there during the installation of Windows or Windows NT. To register the NeuroSolutions Object Library call *regsvr32* passing it the path to the NeuroSolutionsOL.dll file.

{ewl RoboEx32.dll, WinHelp2000, }