AMORE++

pre-alpha (active development aiming to release a beta version this summer (2011))

Generated by Doxygen 1.7.4

Wed May 25 2011 23:34:40

Contents

Chapter 1

The AMORE++ package

1.1 Introduction

Here you will find the documentation of the C++ component of the AMORE++ R package. The AMORE++ package is a new version of the publicly available AMORE package for neural network training and simulation under R

1.2 Motivation

Since the release of the previous version of the AMORE many things have changed in the R programming world. The advent of the Reference Classes and of packages like Rcpp, inline and RUnit compel us to write a better version of the package in order to provide a more useful framework for neural network training and simulation.

1.3 Road Map

This project is currently very active and the development team intends to provide a beta version as soon as this summer (2011)

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

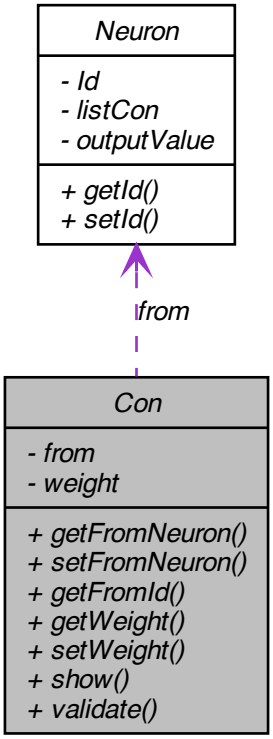
Class Documentation

4.1 Con Class Reference

A class to handle the information needed to describe an input connection.

`#include <Con.h>`

Collaboration diagram for Con:



Public Member Functions

- Neuron * getFromNeuron ()

  from field accessor.
- void setFromNeuron (Neuron *f)

  from field accessor.
- int getFromId ()

  A getter of the Id of the Neuron pointed by the from field.
- double getWeight ()

  weight field accessor.
- void setWeight (double w)

  weight field accessor.
- bool show ()

Pretty print of the Con information.

- bool validate ()

  Object validator.

Private Attributes

- Neuron * from

  A pointer to the Neuron used as input during simulation or training.

- double weight

  A double variable that contains the weight of the connection.

4.1.1 Detailed Description

A class to handle the information needed to describe an input connection.

The Con class provides a simple class for a connection described by a pair of values: a pointer to the Neuron used as the from field and the weight used to propagate the value of that Neuron object.

Definition at line 16 of file Con.h.

4.1.2 Member Function Documentation

4.1.2.1 int Con::getFromId ( )

A getter of the Id of the Neuron pointed by the from field.

This method gets the Id of the Neuron referred to by the from field

Returns

The value of the Id (an integer).

```
//================
//Usage example:
//================
      Con myCon;
      Neuron MyNeuron;
      MyNeuron.setId(16);
      myCon.setFromNeuron(&MyNeuron);
      int result= myCon.getFromId();
```

After execution of the code shown above, MyNeuron::Id is set to the integer value 16 and, thus, result is equal to 16.

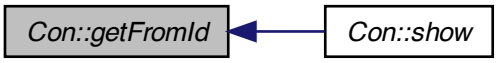
See also

getFromNeuron, setFromNeuron and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

Definition at line 72 of file Con.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.2.2 Neuron * Con::getFromNeuron ( )

from field accessor.

This method allows access to the address stored in the private from field (a pointer to a Neuron object).*

Returns

A pointer to the Neuron object referred to by the from field.

```
//================
//Usage example:
//================
      Con myCon;
      Neuron MyNeuron;
      Neuron * ptNeuron;
      MyNeuron.setId(1);
      myCon.setFromNeuron(&MyNeuron);
      ptNeuron = myCon.getFromNeuron();
      int result= ptNeuron->getId();
```

After execution of the code shown above, ptNeuron is pointing at MyNeuron and, thus, result is equal to 1.

See also

getFromId and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

Definition at line 36 of file Con.cpp.

4.1.2.3 double Con::getWeight ( )

weight field accessor.

This method allows access to the value stored in the private field weight

Returns

The value of weight (double)

```
//================
//Usage example:
//================
      Con myCon;
      Neuron MyNeuron;
      MyNeuron.setId(16);
      myCon.setFromNeuron(&MyNeuron);
      myCon.setWeight(12.4);
      double result1= myCon.getWeight();
      myCon.setWeight(2.2);
      double result2= myCon.getWeight();
```

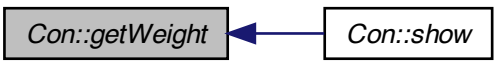
After execution of the code shown above, result1 is set to the double value 12.4 and result2 is set to the double value 2.2.

See also

setWeight and the unit test files, e.g., runit.Cpp.Con.R, for further examples.

Definition at line 102 of file Con.cpp.

Here is the caller graph for this function:



4.1.2.4 void Con::setFromNeuron (Neuron * f)

from field accessor.

This method sets the value of the from field with the address used as parameter.

Parameters

f	is a pointer to the neuron that is to be inserted in the from field.

See also

getFromNeuron and getFromId contain usage examples. For further examples see the unit test files, e.g., runit.Cpp.Con.R

Definition at line 47 of file Con.cpp.

4.1.2.5 void Con::setWeight (double *w*)

weight field accessor.

This method sets the value of the weight field.

Parameters

w	is the new value (double) to be set in the weight field.

```
//================
//Usage example:
//================
      Con myCon;
      Neuron n;
      n.setId(16);
      myCon.setFromNeuron(&n);
      myCon.setWeight(12.4);
      myCon.show();
```

After execution of the code shown above, the output at the R terminal would show:

```
FROM=16 WEIGHT=12.4
```

See also

getWeight and the unit test files (e.g. runit.Cpp.Con.R)

Definition at line 133 of file Con.cpp.

4.1.2.6 bool Con::show ( )

Pretty print of the Con information.

This method outputs in the R terminal the contents of the Con fields.

Returns

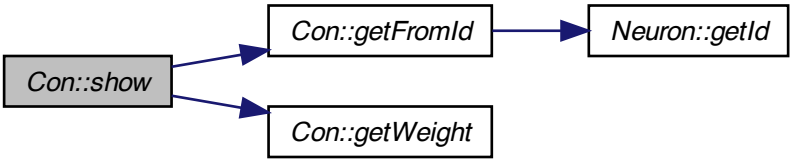
true in case everything works without throwing an exception

See also

setWeight and the unit test files, e.g., runit.Cpp.Con.R, for usage examples.

Definition at line 145 of file Con.cpp.

Here is the call graph for this function:



4.1.2.7 bool Con::validate ( )

Object validator.

This method checks the object for internal coherence. A try / catch mechanism exits normal execution and returns control to the R terminal in case the contents of the Con object are identified as corrupted.

Returns

true in case the checks are Ok.

Exceptions

An	std::range error if weight or from are not finite.

Definition at line 159 of file Con.cpp.

Here is the call graph for this function:

4.1.3 Member Data Documentation

4.1.3.1 Neuron* Con::from [private]

A pointer to the Neuron used as input during simulation or training.

The from field contains the address of the Neuron whose output will be used as input by the Neuron containing the Con object.

Definition at line 21 of file Con.h.

4.1.3.2 double Con::weight [private]

A double variable that contains the weight of the connection.

The weight field contains the factor by which the output value of the Neuron addressed by the from field is multiplied during simulation or training.

Definition at line 26 of file Con.h.

The documentation for this class was generated from the following files:

- pkg/AMORE/src/Con.h
- pkg/AMORE/src/Con.cpp

4.2 Neuron Class Reference

A class to handle the information contained in a general Neuron.

`#include <Neuron.h>`

Public Member Functions

- int getId ()
- void setId (int id)

Private Attributes

- int Id

  An integer variable with the Neuron Id.
- std::vector< Con > listCon

  A vector of input connections.
- double outputValue

4.2.1 Detailed Description

A class to handle the information contained in a general Neuron.

A general class for neurons. The MLPneuron and RBFneuron classes will specialize this general class

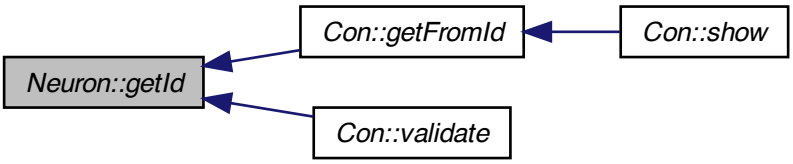
Definition at line 16 of file Neuron.h.

4.2.2 Member Function Documentation

4.2.2.1 int Neuron::getId ( )

Definition at line 15 of file Neuron.cpp.

Here is the caller graph for this function:



4.2.2.2 void Neuron::setId (int *id*)

Definition at line 19 of file Neuron.cpp.

4.2.3 Member Data Documentation

4.2.3.1 int Neuron::Id [private]

An integer variable with the Neuron Id.

The Neuron Id provides a name to the neuron. This value is not expected to be used neither during simulation nor training but it provides an easy reference for human readers.

Definition at line 21 of file Neuron.h.

4.2.3.2 std::vector<Con> Neuron::listCon [private]

A vector of input connections.

Definition at line 28 of file Neuron.h.

4.2.3.3 double Neuron::outputValue [private]

Definition at line 29 of file Neuron.h.

The documentation for this class was generated from the following files:

- pkg/AMORE/src/Neuron.h
- pkg/AMORE/src/Neuron.cpp

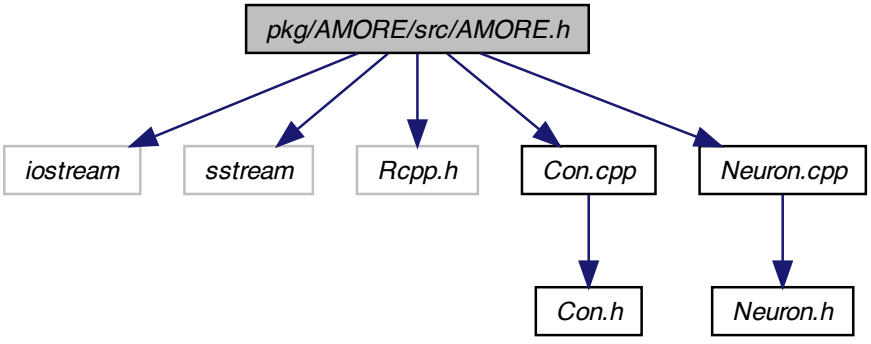
Chapter 5

File Documentation

5.1 pkg/AMORE/src/AMORE.h File Reference

```
#include <iostream>
#include <sstream>
#include <Rcpp.h>
#include "Con.cpp"
#include "Neuron.cpp"
```

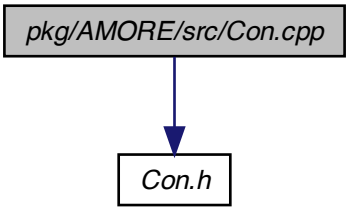
Include dependency graph for AMORE.h:



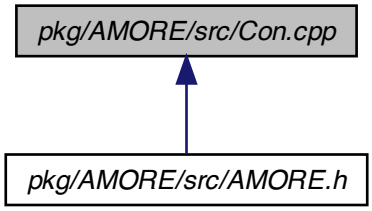
5.2 pkg/AMORE/src/Con.cpp File Reference

```
#include "Con.h"
```

Include dependency graph for Con.cpp:

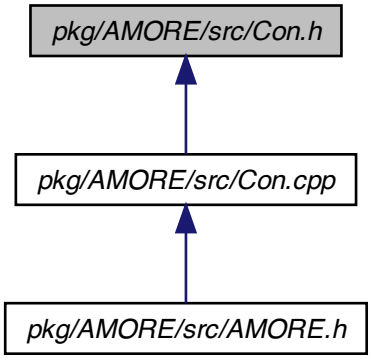


This graph shows which files directly or indirectly include this file:



5.3 pkg/AMORE/src/Con.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

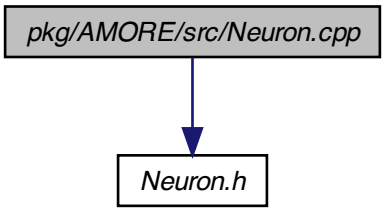
- class Con

  A class to handle the information needed to describe an input connection.

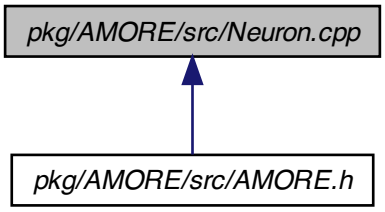
5.4 pkg/AMORE/src/Neuron.cpp File Reference

```
#include "Neuron.h"
```

Include dependency graph for Neuron.cpp:

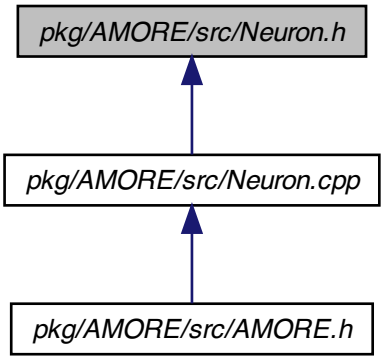


This graph shows which files directly or indirectly include this file:



5.5 pkg/AMORE/src/Neuron.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Neuron

  A class to handle the information contained in a general Neuron.

Index