



UNIVERSIDAD CARLOS III DE MADRID

INFORMÁTICA INDUSTRIAL I

Neural Network for handwritten digits recognition

Author:

David Estévez Fernández

Teacher:

Alberto Valero Gómez

December 18, 2012

Contents

1	Introduction	2
1.1	Objective	2
1.2	Neural network implementation	2
1.3	Training	2
1.4	Conclusion	2
2	User manual	3
2.1	General usage	3
2.2	Main menu	3
2.3	Network configuration	3
2.4	Guess number	5
2.5	Train Neural Network	5
2.6	Save	6
3	Activities diagram	7
4	Class Index	8
4.1	Class List	8
5	Class Documentation	9
5.1	Dendrite Struct Reference	9
5.1.1	Detailed Description	9
5.1.2	Member Data Documentation	9
5.2	Layer Class Reference	9
5.2.1	Detailed Description	10
5.2.2	Constructor & Destructor Documentation	10
5.2.3	Member Function Documentation	10
5.2.4	Member Data Documentation	12
5.3	Matrix Class Reference	12
5.3.1	Detailed Description	14
5.3.2	Constructor & Destructor Documentation	14
5.3.3	Member Function Documentation	14
5.3.4	Friends And Related Function Documentation	19
5.3.5	Member Data Documentation	19
5.4	NeuralNetwork Class Reference	20
5.4.1	Detailed Description	21
5.4.2	Constructor & Destructor Documentation	21
5.4.3	Member Function Documentation	21
5.4.4	Member Data Documentation	23
5.5	NeuralNetworkIO Class Reference	23
5.5.1	Detailed Description	25
5.5.2	Constructor & Destructor Documentation	25

5.5.3	Member Function Documentation	25
5.5.4	Member Data Documentation	25
5.6	Neuron Class Reference	25
5.6.1	Detailed Description	26
5.6.2	Constructor & Destructor Documentation	26
5.6.3	Member Function Documentation	27
5.6.4	Member Data Documentation	28
5.7	NNFileInput Class Reference	29
5.7.1	Detailed Description	30
5.7.2	Constructor & Destructor Documentation	30
5.7.3	Member Function Documentation	31
5.7.4	Member Data Documentation	32
5.8	NNFileOutput Class Reference	33
5.8.1	Detailed Description	34
5.8.2	Constructor & Destructor Documentation	34
5.8.3	Member Function Documentation	35
5.8.4	Member Data Documentation	36
5.9	NNInput Class Reference	36
5.9.1	Detailed Description	38
5.9.2	Constructor & Destructor Documentation	38
5.9.3	Member Function Documentation	39
5.9.4	Member Data Documentation	39
5.10	NNOutput Class Reference	39
5.10.1	Detailed Description	41
5.10.2	Constructor & Destructor Documentation	41
5.10.3	Member Function Documentation	41
5.11	NNStdOutput Class Reference	41
5.11.1	Detailed Description	43
5.11.2	Constructor & Destructor Documentation	43
5.11.3	Member Function Documentation	43
5.11.4	Member Data Documentation	43
5.12	NNTrainer Class Reference	44
5.12.1	Detailed Description	46
5.12.2	Constructor & Destructor Documentation	47
5.12.3	Member Function Documentation	47
5.12.4	Member Data Documentation	49
5.13	TerminalInterface Class Reference	50
5.13.1	Detailed Description	51
5.13.2	Constructor & Destructor Documentation	51
5.13.3	Member Function Documentation	51
5.13.4	Member Data Documentation	52
5.14	TerminalMenu Class Reference	52

5.14.1	Detailed Description	54
5.14.2	Constructor & Destructor Documentation	54
5.14.3	Member Function Documentation	54
5.14.4	Member Data Documentation	54
5.15	TerminalTextEdit Class Reference	55
5.15.1	Detailed Description	56
5.15.2	Constructor & Destructor Documentation	56
5.15.3	Member Function Documentation	56
5.15.4	Member Data Documentation	57
5.16	TrainingExample Struct Reference	57
5.16.1	Detailed Description	57
5.16.2	Member Data Documentation	57

1 Introduction

1.1 Objective

The main objective of this project was to learn C++ programming through the implementation of a neural network able to recognize handwritten digits. These digits were digitalized into matrices of 20x20 pixels, with a value of grey for each pixel.

1.2 Neural network implementation

The implementation was done by means of a down-top approach, starting by modelling a neuron, as the simplest unit of the network, and then modelling groups of neurons (layers) and groups of layers (networks).

This implementation is very convenient from a conceptual point of view, as it is very intuitive to model the neurons and their relationships by using objects and classes. But, from a performance point of view, this implementation has a lower performance than a implementation of the network modelled as vectors and matrices, in which the hypothesis or guess of the network is obtained by using linear algebra operations, as linear algebra libraries for C++ are faster calculating the output of the network as well as fitting the weights to the network to the training set.

1.3 Training

For the training of the network, 5000 of the 60000 training examples from the United States National Institute of Science and Technology (NIST) handwritten digit database were used. That dataset is provided with this code, for the user to be able to train the network.

The actual training was carried out by implementing the backpropagation algorithm to find the gradient of the cost function of the network for each value of weights vector. Once the gradient is obtained, a gradient descend algorithm is used to find the optimal values for the weights so that they fit perfectly the training examples.

As no numerical computing library is being used to make those calculations, the training of the network is very slow and demanding for the computer. For that reason, a set of trained weights for the network is also provided.

1.4 Conclusion

If computation time is a concern I would recommend either reimplement the neural network and trainer to work with a numerical computing library or perform the training of the network on a numerical computing software such as Octave, as computation of the optimal set is much faster than with the current implementation.

For calculating the network guess, as it is a less computational demanding process this implementation could still work with a reasonable performance level.

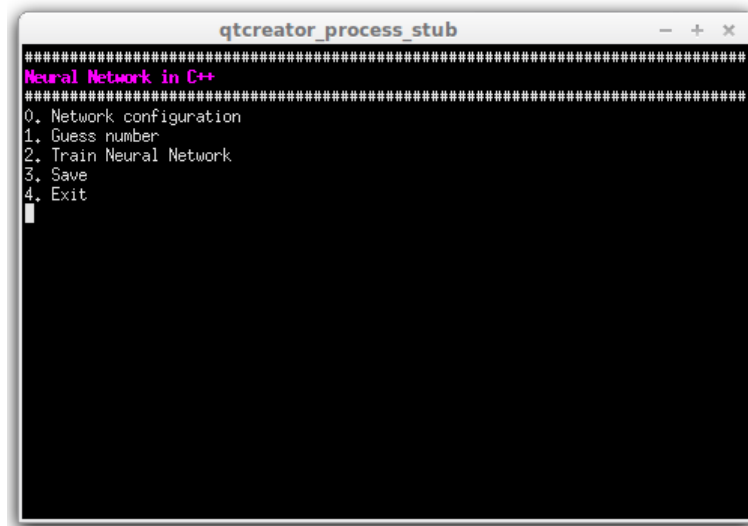
2 User manual

2.1 General usage

The interface of the program consist of several menus displayed on a ANSI escape sequence capable terminal. The way to select an option from the menu is to type the index of the option to be selected, then press the enter key.

2.2 Main menu

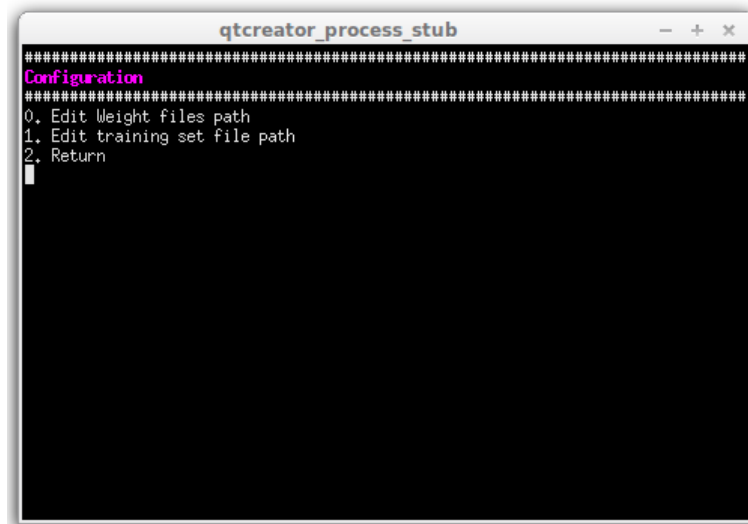
The main menu allows the user to select among several option to work with the neural network, such as configuring the input file paths or training it.



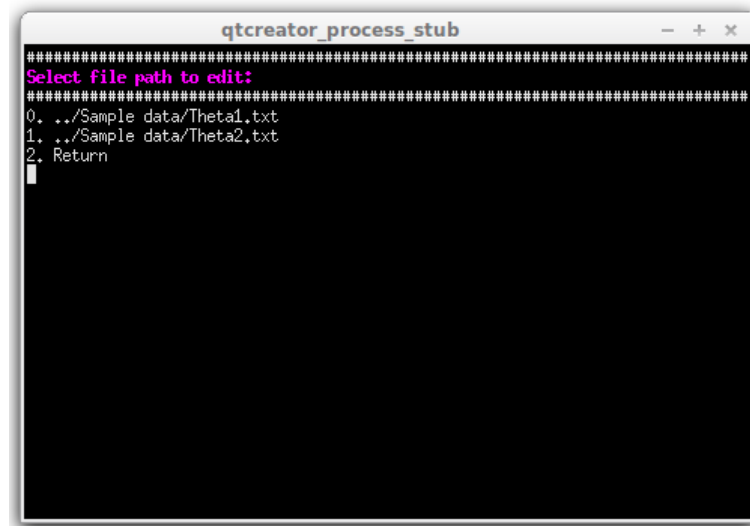
```
qtcreator_process_stub
#####
Neural Network in C++
#####
0. Network configuration
1. Guess number
2. Train Neural Network
3. Save
4. Exit
█
```

2.3 Network configuration

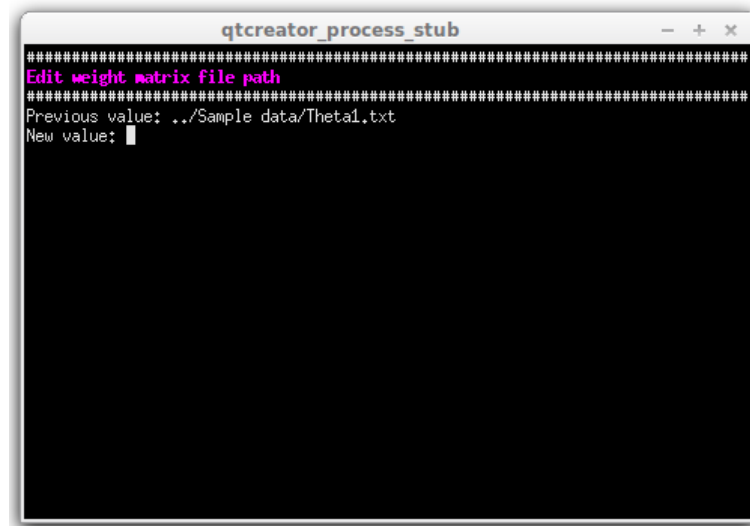
In this menu the user will be able to specify the system path for the input files where the weights of the network and the training set can be found.



```
qtcreator_process_stub
#####
Configuration
#####
0. Edit Weight files path
1. Edit training set file path
2. Return
█
```



Using the menu system a file path can be selected and edited:

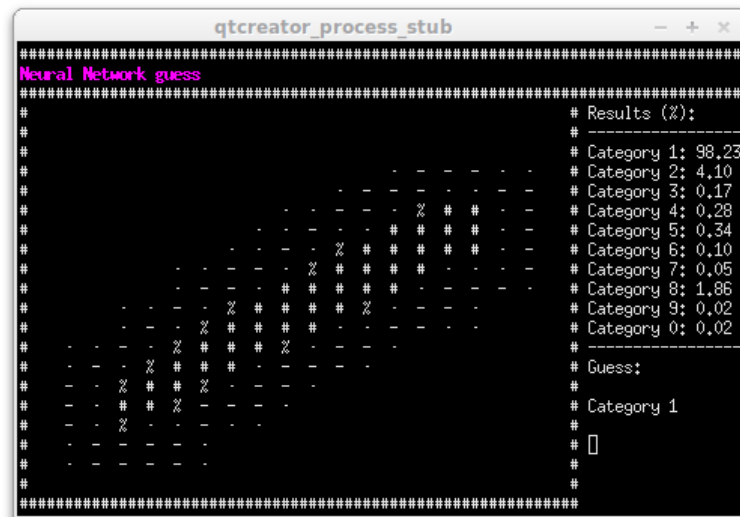


2.4 Guess number

With this option a number, expressed as a 20x20 pixel matrix in a text document can be loaded to the program, and the neural network will guess which number it is.

The program will ask the user for a path where to find an input file. Some sample inputs are provided to the user in the "Sample data" folder, under the name of "input-XX.txt", where XX is a number.

Once we have specified the input file, the network will return its guess:



```

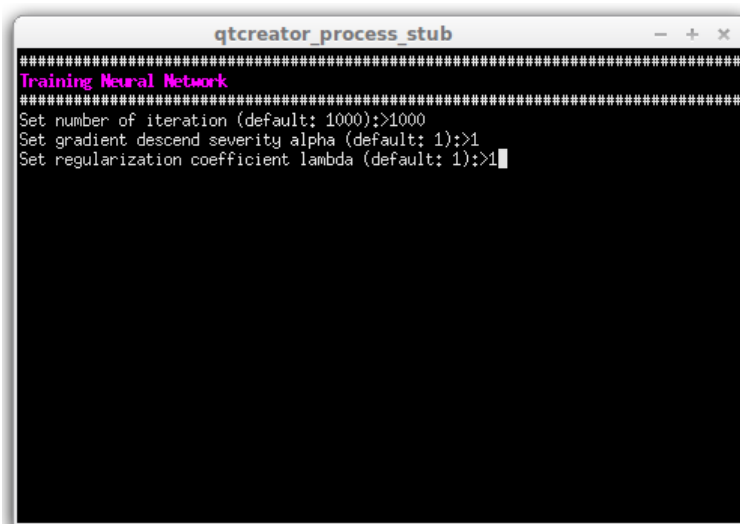
qtcreator_process_stub
=====
Neural Network guess
=====
# Results (%):
# Category 1: 98.23
# Category 2: 4.10
# Category 3: 0.17
# Category 4: 0.28
# Category 5: 0.34
# Category 6: 0.10
# Category 7: 0.05
# Category 8: 1.86
# Category 9: 0.02
# Category 0: 0.02
#
# Guess:
# Category 1
#
#
=====
  
```

Once the user is done, it can return to the main menu pressing the enter key.

2.5 Train Neural Network

With this option the user will call the trainer to fit the weights of the network to a given training set. A sample training set is provided to the user in the "Sample data" folder, and will be loaded by default by the neural network.

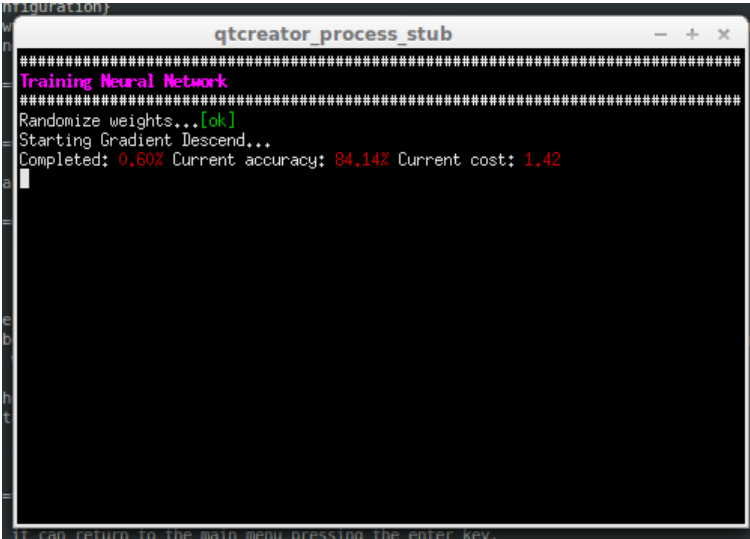
Before starting the training process, the program will ask the user for some training parameters to configure the gradient descent algorithm, such as the number of iterations that it will be running, the constant to be multiplied to the gradient, alpha, and the regularization term to avoid overfitting, lambda.



```

qtcreator_process_stub
=====
Training Neural Network
=====
Set number of iteration (default: 1000):>1000
Set gradient descend severity alpha (default: 1):>1
Set regularization coefficient lambda (default: 1):>1
  
```


After inserting the parameters, it will start the training. This process is very demanding for the computer, and it will take A LONG TIME. With the default parameters it may need hours to finish the training. Once it is done, the user can return to the main menu by pressing the enter key.



```

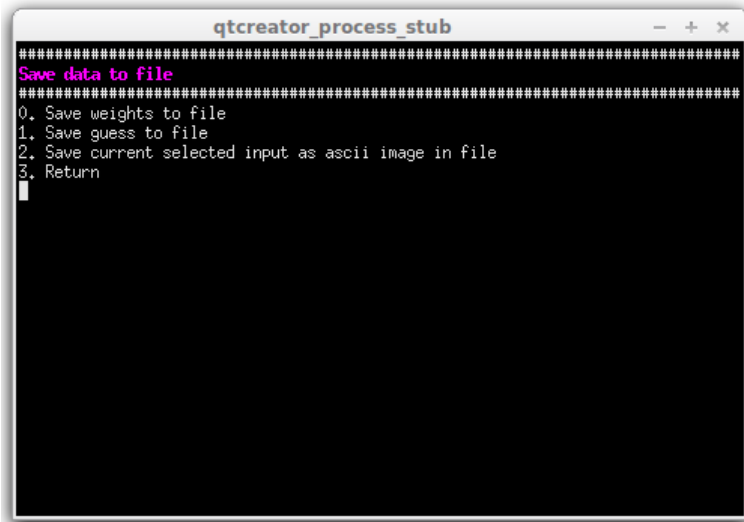
qtcreator_process_stub
=====
Training Neural Network
=====
Randomize weights...[ok]
Starting Gradient Descend...
Completed: 0.60% Current accuracy: 84.14% Current cost: 1.42

```

2.6 Save

This menu gives the user the option of saving some data to a file. User can save as a file:

- Current weights. They are saved in several files, but a single name is asked (The names for the files are generated from that single name).
- Result of the last guess of the network.
- Input data as an ascii image, representing the 20x20 data matrix into something more human-readable.

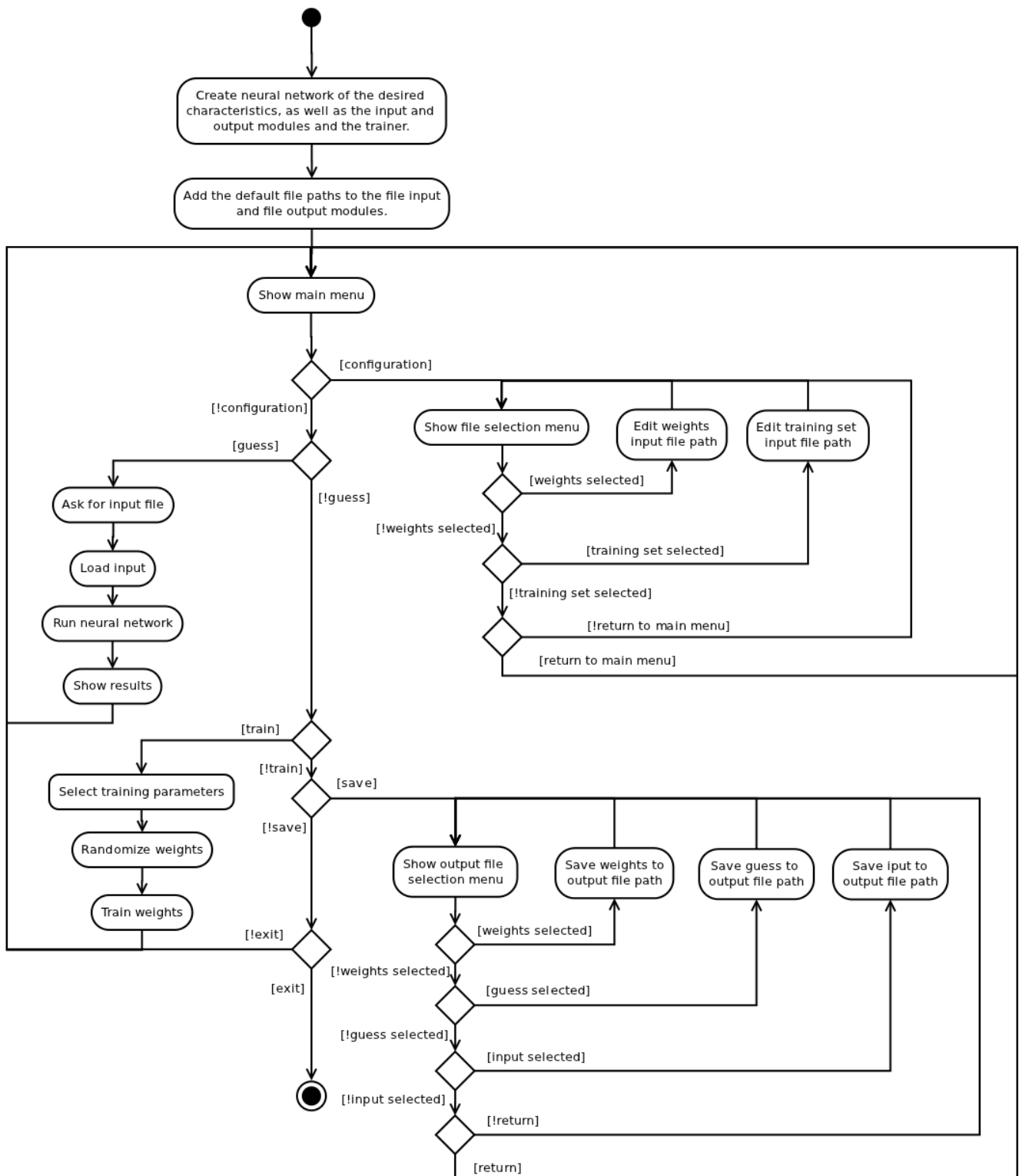


```

qtcreator_process_stub
=====
Save data to file
=====
0. Save weights to file
1. Save guess to file
2. Save current selected input as ascii image in file
3. Return

```

3 Activities diagram



4 Class Index

4.1 Class List

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

Dendrite (Weighted connection between two neurons)	9
Layer (Group of neurons sharing the same level)	9
Matrix (Bidimensional matrix of doubles)	12
NeuralNetwork (Generic implementation of a neural network for data classification)	20
NeuralNetworkIO (Generic input/output interface of a neural network to communicate and exchange data with the outside (std I/O, files, etc))	23
Neuron (Basic unit of a neural network)	25
NNFileInput (File support for data input to NeuralNetwork (p. 20))	29
NNFileOutput (File support for data output from NeuralNetwork (p. 20))	33
NNInput (Generic input interface of a NeuralNetwork (p. 20) to communicate and exchange data with the outside (STD input, input files, etc))	36
NNOutput (Generic output interface of a NeuralNetwork (p. 20) to communicate and send data to the outside (STD output, save files, etc))	39
NNStdOutput (Show NeuralNetwork (p. 20) data using std output)	41
NNTrainer (Trains the weights of a neural network given a training set)	44
TerminalInterface (Base class for creating simple terminal interfaces)	50
TerminalMenu (Basic terminal menu interface)	52
TerminalTextEdit (Basic text edition on terminal interface)	55
TrainingExample (A given input for the network and the expected output for that input)	57

5 Class Documentation

5.1 Dendrite Struct Reference

Weighted connection between two neurons.

```
#include <neuron.h>
```

Public Attributes

- **Neuron * connection**
*Pointer to the neuron connected to the neuron containing this **Dendrite** (p. 9).*
- **double * weight**
*Pointer to a variable containing the value of the weight for this **Dendrite** (p. 9).*

5.1.1 Detailed Description

Weighted connection between two neurons.

"Dendrite" is the scientific name for the connections between real neurons.

5.1.2 Member Data Documentation

5.1.2.1 Neuron * Dendrite::connection

Pointer to the neuron connected to the neuron containing this **Dendrite** (p. 9).

5.1.2.2 double * Dendrite::weight

Pointer to a variable containing the value of the weight for this **Dendrite** (p. 9).

The documentation for this struct was generated from the following file:

- neuron.h

5.2 Layer Class Reference

Group of neurons sharing the same level.

```
#include <layer.h>
```

Public Member Functions

- **Layer** (int n)
*Creates a **Layer** (p. 9) composed by n neurons plus a bias unit (n+1 neurons)*
- void **setOutput** (std::vector< double > **output**)
Sets the output vector of the layer.
- void **setWeights** (**Matrix** *theta)
*Sets the weights of all neurons in the layer to be the ones stored in **Matrix** (p. 12) theta.*
- void **refresh** ()
Recalculates the output vector.
- std::vector< double > **getOutput** ()
Returns the output vector for this layer.
- int **getN** ()
*Return the number of neurons in this **Layer** (p. 9) (including bias unit).*
- **Matrix** **getWeights** ()

Returns the weights of all neurons in the layer.

- void **connectLayer** (**Layer** &prevLayer)
Connect this layer's neurons to the neurons of the previous layer.
- void **operator<<** (**Layer** &prevLayer)
Connect this layer's neurons to the neurons of the previous layer.
- void **connectLayer** (**Layer** *prevLayer)
Connect this layer's neurons to the neurons of the previous layer.
- void **operator<<** (**Layer** *prevLayer)
Connect this layer's neurons to the neurons of the previous layer.

Private Member Functions

- **Layer** ()
Default constructor.

Private Attributes

- std::vector< **Neuron** > **neurons**
*Stores the neurons in this **Layer** (p. 9).*
- std::vector< double > **output**
*Stores output vector of the **Layer** (p. 9).*
- int **n**
*Number of neurons in this **Layer** (p. 9) (including bias unit).*

5.2.1 Detailed Description

Group of neurons sharing the same level.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 **Layer::Layer** (int *n*)

Creates a **Layer** (p. 9) composed by *n* neurons plus a bias unit (*n*+1 neurons)

Parameters

<i>n</i>	Number of neurons in this Layer (p. 9) (Excluding bias unit).
----------	--

5.2.2.2 **Layer::Layer** () [private]

Default constructor.

5.2.3 Member Function Documentation

5.2.3.1 void **Layer::connectLayer** (**Layer** & *prevLayer*)

Connect this layer's neurons to the neurons of the previous layer.

Parameters

<i>prevLayer</i>	Layer (p. 9) to which this Layer (p. 9) will be connected.
------------------	--

5.2.3.2 void **Layer::connectLayer** (**Layer** * *prevLayer*)

Connect this layer's neurons to the neurons of the previous layer.

Parameters

<i>prevLayer</i>	Pointer to Layer (p. 9) to which this Layer (p. 9) will be connected.
------------------	---

5.2.3.3 `int Layer::getN ()`

Return the number of neurons in this **Layer** (p. 9) (including bias unit).

Returns

Number of neurons in this **Layer** (p. 9) (including bias unit).

5.2.3.4 `std::vector< double > Layer::getOutput ()`

Returns the output vector for this layer.

As calculations within the network are carried out by neurons and not by layers, this function as only debug purposes.

Returns

Vector containing the output of the **Layer** (p. 9).

5.2.3.5 `Matrix Layer::getWeights ()`

Returns the weights of all neurons in the layer.

Returns a matrix containing the weights of all connections of all neurons in the layer. Each row corresponds to the weights of a single neuron. Bias unit has no connections, so it has no weights.

Returns

Matrix (p. 12) containing weights of all neurons in layer.

5.2.3.6 `void Layer::operator<< (Layer & prevLayer)`

Connect this layer's neurons to the neurons of the previous layer.

Parameters

<i>prevLayer</i>	Layer (p. 9) to which this Layer (p. 9) will be connected.
------------------	--

5.2.3.7 `void Layer::operator<< (Layer * prevLayer)`

Connect this layer's neurons to the neurons of the previous layer.

Parameters

<i>prevLayer</i>	Pointer to Layer (p. 9) to which this Layer (p. 9) will be connected.
------------------	---

5.2.3.8 `void Layer::refresh ()`

Recalculates the output vector.

Refreshes each one of the neurons in the layer, except the bias unit.

5.2.3.9 `void Layer::setOutput (std::vector< double > output)`

Sets the output vector of the layer.

This is used in the input layer, to place the input of the network in a **Layer** (p. 9).

Parameters

<i>output</i>	Vector to set as output of this Layer (p. 9).
---------------	--

5.2.3.10 void Layer::setWeights (**Matrix** * *theta*)

Sets the weights of all neurons in the layer to be the ones stored in **Matrix** (p. 12) *theta*.

Note

Each neuron's vector is stored in a row of the matrix. The bias unit does not have any weights, as no previous neuron connects with it.

Parameters

<i>theta</i>	Pointer to a Matrix (p. 12) storing the weights of the Layer (p. 9).
--------------	--

5.2.4 Member Data Documentation

5.2.4.1 int Layer::n [private]

Number of neurons in this **Layer** (p. 9) (including bias unit).

5.2.4.2 std::vector< **Neuron** > Layer::neurons [private]

Stores the neurons in this **Layer** (p. 9).

5.2.4.3 std::vector< double > Layer::output [private]

Stores output vector of the **Layer** (p. 9).

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

- layer.h
- layer.cpp

5.3 Matrix Class Reference

Bidimensional matrix of doubles.

```
#include <matrix.h>
```

Public Member Functions

- **Matrix** (const int **rows**, const int **cols**)
Creates a rows x cols matrix of doubles.
- **Matrix** (const std::vector< double > dataVector, const int **rows**, const int **cols**)
Creates a rows x cols matrix from a vector with the data.
- **Matrix** (const **Matrix** &otherMatrix)
Creates a matrix from other matrix.
- ~**Matrix** ()
Default destructor.
- int **getNumRows** () const
Returns the number of rows of the matrix.
- int **getNumCols** () const
Returns the number of columns of the matrix.
- double **get** (const int row, const int col) const
Returns a concrete value from the matrix.
- std::vector< double * > **getRow** (const int row) const
Returns a vector of pointers to each element of a row of the matrix.
- std::vector< double > **getRowValues** (const int row) const
Returns the values from a row of the matrix.
- std::vector< double * > **getCol** (const int col) const

Returns a vector of pointers to each element of a column of the matrix.

- `std::vector< double > getColValues (const int col) const`

Returns the values from a column of the matrix.

- `std::vector< double > unroll ()`

Returns a one-dimensional vector containing all the values from the matrix.

- `void set (const int row, const int col, const double value)`

Set manually one value of the matrix.

- `void setRow (const std::vector< double > row, const int index)`

Change all the values of a row with the values of a vector.

- `void setCol (const std::vector< double > col, const int index)`

Change all the values of a column with the values of a vector.

- `Matrix operator+ (const Matrix &m)`

Sum two matrices.

- `Matrix operator- (const Matrix &m)`

Subtract two matrices.

- `Matrix operator* (const Matrix &m)`

Calculates the product of two matrices.

- `Matrix operator* (const double n)`

Multiply each element by a constant.

- `Matrix operator/ (const double n)`

Divide each element by a constant.

- `Matrix & operator+= (const Matrix &m)`

Add another matrix to this.

- `Matrix & operator-= (const Matrix &m)`

Subtract another matrix to this.

- `Matrix & operator*= (const double n)`

Multiply a constant to this elementwise.

- `Matrix & operator/= (const double n)`

Divide a constant to this elementwise.

- `bool operator== (const Matrix &otherMat)`

Checks if two matrices are equal, element-wise.

- `bool operator!= (const Matrix &otherMat)`

Checks if two matrices are different, element-wise.

- `void operator= (const Matrix &otherMatrix)`

Assign to this matrix the value of other matrix.

- `Matrix transpose ()`

Calculate the transpose matrix of this.

Private Member Functions

- `Matrix ()`

Default constructor.

Private Attributes

- `int cols`

Stores the number of columns in matrix.

- `int rows`

Stores the number of rows in matrix.

- `double * matrix`

points to an array of doubles storing the values of the matrix elements.

Friends

- `std::ostream & operator<< (std::ostream &out, Matrix &matrix)`
Shows a matrix in an output stream.

5.3.1 Detailed Description

Bidimensional matrix of doubles.

It has functions for single value, row or column modification and return, as well as support for basic arithmetics operations.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 `Matrix::Matrix (const int rows, const int cols)`

Creates a rows x cols matrix of doubles.

Parameters

<i>rows</i>	Number of rows.
<i>cols</i>	Number of columns.

5.3.2.2 `Matrix::Matrix (const std::vector< double > dataVector, const int rows, const int cols)`

Creates a rows x cols matrix from a vector with the data.

Parameters

<i>dataVector</i>	Vector containing the numbers of the matrix.
<i>rows</i>	Number of rows.
<i>cols</i>	Number of columns.

5.3.2.3 `Matrix::Matrix (const Matrix & otherMatrix)`

Creates a matrix from other matrix.

Parameters

<i>otherMatrix</i>	The matrix we are copying.
--------------------	----------------------------

5.3.2.4 `Matrix::~~Matrix ()`

Default destructor.

5.3.2.5 `Matrix::Matrix () [private]`

Default constructor.

5.3.3 Member Function Documentation

5.3.3.1 `double Matrix::get (const int row, const int col) const`

Returns a concrete value from the matrix.

Parameters

<i>row</i>	Row selection.
<i>col</i>	Column selection.

Returns

Value of the element at position (row , col)

5.3.3.2 `std::vector< double * > Matrix::getCol (const int col) const`

Returns a vector of pointers to each element of a column of the matrix.

Parameters

<i>col</i>	Column selection.
------------	-------------------

Returns

Vector containing pointers to each element of the selected column.

5.3.3.3 `std::vector< double > Matrix::getColValues (const int col) const`

Returns the values from a column of the matrix.

Parameters

<i>col</i>	Column selection.
------------	-------------------

Returns

Vector containing the values from a column of the matrix.

5.3.3.4 `int Matrix::getNumCols () const`

Returns the number of columns of the matrix.

Returns

Number of columns of the matrix

5.3.3.5 `int Matrix::getNumRows () const`

Returns the number of rows of the matrix.

Returns

Number of rows of the matrix

5.3.3.6 `std::vector< double * > Matrix::getRow (const int row) const`

Returns a vector of pointers to each element of a row of the matrix.

Parameters

<i>row</i>	Row selection.
------------	----------------

Returns

Vector containing pointers to each element of the selected row.

5.3.3.7 `std::vector< double > Matrix::getRowValues (const int row) const`

Returns the values from a row of the matrix.

Parameters

<i>row</i>	Row selection.
------------	----------------

Returns

Vector containing the values from a row of the matrix.

5.3.3.8 `bool Matrix::operator!=(const Matrix & otherMat)`

Checks if two matrices are different, element-wise.

Two matrices are different if any of their corresponding elements are different.

Parameters

<i>otherMat</i>	Matrix (p. 12) we are comparing with this matrix.
-----------------	--

Returns

True if both are different, false otherwise.

5.3.3.9 `Matrix Matrix::operator* (const Matrix & m)`

Calculates the product of two matrices.

Warning

Dimensions must be compatible.

Parameters

<i>m</i>	Matrix (p. 12) to multiply.
----------	------------------------------------

Returns

Result of the product of the matrices.

5.3.3.10 `Matrix Matrix::operator* (const double n)`

Multiply each element by a constant.

Parameters

<i>n</i>	Constant to multiply.
----------	-----------------------

Returns

Result of the operation.

5.3.3.11 `Matrix & Matrix::operator*= (const double n)`

Multiply a constant to this elementwise.

Parameters

<i>n</i>	Constant to multiply.
----------	-----------------------

Returns

Reference to this.

5.3.3.12 `Matrix Matrix::operator+ (const Matrix & m)`

Sum two matrices.

Warning

Dimensions must be compatible.

Parameters

<i>m</i>	Matrix (p. 12) to sum.
----------	-------------------------------

Returns

Result of the sum of the matrices.

5.3.3.13 Matrix & Matrix::operator+= (const Matrix & *m*)

Add another matrix to this.

Warning

Dimensions must be compatible.

Parameters

<i>m</i>	Matrix (p. 12) to sum.
----------	-------------------------------

Returns

Reference to this.

5.3.3.14 Matrix Matrix::operator- (const Matrix & *m*)

Subtract two matrices.

Warning

Dimensions must be compatible.

Parameters

<i>m</i>	Matrix (p. 12) to subtract.
----------	------------------------------------

Returns

Result of the subtraction of the matrices.

5.3.3.15 Matrix & Matrix::operator-= (const Matrix & *m*)

Subtract another matrix to this.

Warning

Dimensions must be compatible.

Parameters

<i>m</i>	Matrix (p. 12) to subtract.
----------	------------------------------------

Returns

Reference to this.

5.3.3.16 Matrix Matrix::operator/ (const double *n*)

Divide each element by a constant.

Parameters

<i>n</i>	Constant to divide by.
----------	------------------------

Returns

Result of the operation.

5.3.3.17 Matrix & Matrix::operator/= (const double *n*)

Divide a constant to this elementwise.

Parameters

<i>n</i>	Constant to divide by.
----------	------------------------

Returns

Reference to this.

5.3.3.18 void Matrix::operator= (const Matrix & *otherMatrix*)

Assign to this matrix the value of other matrix.

Parameters

<i>otherMatrix</i>	Matrix (p. 12) from which values are assigned to this matrix.
--------------------	--

5.3.3.19 bool Matrix::operator== (const Matrix & *otherMat*)

Checks if two matrices are equal, element-wise.

Two matrices are equal if all their corresponding elements are equal.

Parameters

<i>otherMat</i>	Matrix (p. 12) we are comparing with this matrix.
-----------------	--

Returns

True if both are equal, false otherwise.

5.3.3.20 void Matrix::set (const int *row*, const int *col*, const double *value*)

Set manually one value of the matrix.

Parameters

<i>row</i>	Row selector.
<i>col</i>	Column selector.
<i>value</i>	Value to store.

5.3.3.21 void Matrix::setCol (const std::vector< double > *col*, const int *index*)

Change all the values of a column with the values of a vector.

Warning

Dimensions must be compatible.

Parameters

<i>col</i>	Values for the new column.
<i>index</i>	Index of the column to change values.

5.3.3.22 `void Matrix::setRow (const std::vector< double > row, const int index)`

Change all the values of a row with the values of a vector.

Warning

Dimensions must be compatible.

Parameters

<i>row</i>	Values for the new row.
<i>index</i>	Index of the row to change values.

5.3.3.23 `Matrix Matrix::transpose ()`

Calculate the transpose matrix of this.

Returns

Transpose matrix of this.

5.3.3.24 `std::vector< double > Matrix::unroll ()`

Returns a one-dimensional vector containing all the values from the matrix.

Concatenates columns one after the other to form a one-dimensional vector.

Returns

Vector containing all the values of matrix.

5.3.4 Friends And Related Function Documentation

5.3.4.1 `std::ostream& operator<< (std::ostream & out, Matrix & matrix)` [friend]

Shows a matrix in an output stream.

Parameters

<i>out</i>	Output stream to which we are passing the data that we want to represent.
<i>matrix</i>	Matrix (p. 12) which contains the data passed to the output stream.

5.3.5 Member Data Documentation

5.3.5.1 `int Matrix::cols` [private]

Stores the number of columns in matrix.

5.3.5.2 `double * Matrix::matrix` [private]

points to an array of doubles storing the values of the matrix elements.

5.3.5.3 `int Matrix::rows` [private]

Stores the number of rows in matrix.

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

- matrix.h
- matrix.cpp

5.4 NeuralNetwork Class Reference

Generic implementation of a neural network for data classification.

```
#include <neuralnetwork.h>
```

Public Member Functions

- **NeuralNetwork** (std::vector< int > neuronsInLayer)
Constructor that creates the neural network architecture.
- **~NeuralNetwork** ()
Default destructor.
- void **setInput** (std::vector< double > input)
Sets a vector to be the input of the network, and stores it in the input layer.
- void **setInput** (double *input, int size)
Sets an array to be the input of the network, and stores it in the input layer.
- void **setWeights** (std::vector< **Matrix** * > theta)
Sets the weights of the network.
- std::vector< double > **getOutput** ()
Returns the output of the network.
- std::vector< **Matrix** * > **getWeights** ()
Get a vector containing several the direction of the weight matrices stored in memory.
- std::vector< double > **getInput** ()
Returns the input of the network.
- std::vector< int > **getDimensions** ()
Returns a vector containing the network dimensions.
- std::vector< double > **getActivation** (int n)
Returns the activation vector of layer n.
- int **getL** ()
Return number of layers.

Private Member Functions

- void **refresh** ()
Updates the output of the network.
- **NeuralNetwork** ()
Default constructor.

Private Attributes

- std::vector< double > **output**
Vector containing the output data.
- std::vector< **Matrix** * > **weights**
Vector containing pointers to the weight matrices.
- **Layer** * **inputLayer**
First layer of the network, whose function is to store the input data.
- std::vector< **Layer** > **hiddenLayer**
Vector containing all the hidden layers.
- **Layer** * **outputLayer**
Last layer of the network, calculates and stores the output data.
- int **l**
Number of layer in the network.
- std::vector< int > **dimensions**
*Vector containing the dimensions of the **NeuralNetwork** (p. 20).*

5.4.1 Detailed Description

Generic implementation of a neural network for data classification.

Several architectures of neural network can be created with this class.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 NeuralNetwork::NeuralNetwork (`std::vector< int > neuronsInLayer`)

Constructor that creates the neural network architecture.

Neural network is created from a vector containing l elements, where l is the number of layers composing the network and each element is the number of neurons of each layer (not including the bias unit).

For instance, if input vector is {8,4,1}, a network of 3 layers, with an input layer with 8 + 1 neurons, a hidden layer of 4 + 1 neurons and an output layer of 1 neuron (output layer does not need a bias unit) is created.

Parameters

<i>neuronsInLayer</i>	Vector containing l elements, where l is the number of layers composing the network and each element is the number of neurons of each layer (not including the bias unit).
-----------------------	--

5.4.2.2 NeuralNetwork::~~NeuralNetwork ()

Default destructor.

5.4.2.3 NeuralNetwork::NeuralNetwork () [private]

Default constructor.

5.4.3 Member Function Documentation

5.4.3.1 `std::vector< double > NeuralNetwork::getActivation (int n)`

Returns the activation vector of layer n .

Returns

Activation vector of layer n .

5.4.3.2 `std::vector< int > NeuralNetwork::getDimensions ()`

Returns a vector containing the network dimensions.

Returns

Vector containing the network dimensions.

5.4.3.3 `std::vector< double > NeuralNetwork::getInput ()`

Returns the input of the network.

Returns

Current input vector of the network.

5.4.3.4 `int NeuralNetwork::getL () [inline]`

Return number of layers.

Returns

Number of layers, l .

5.4.3.5 `std::vector< double > NeuralNetwork::getOutput ()`

Returns the output of the network.

The output of the network is a vector containing the probability, between 0 and 1, that the input data belongs to the category represented by each output neuron.

Returns

Vector containing the probability of input data belonging to each category, with values between 0 and 1.

5.4.3.6 `std::vector< Matrix * > NeuralNetwork::getWeights ()`

Get a vector containing several the direction of the weight matrices stored in memory.

The weights are stored in matrices, and there is a matrix for each layer that connects with a previous one (i.e. the input layer has no weights associated).

Each neuron weights are stored in rows, and the dimensions of the matrix should be $n \times m$, where n is the number of neurons in the current layer (not including bias unit) and m is the number of neurons in the previous layer (including bias unit).

Returns

Vector containing the weight matrices.

5.4.3.7 `void NeuralNetwork::refresh () [private]`

Updates the output of the network.

5.4.3.8 `void NeuralNetwork::setInput (double * input, int size)`

Sets an array to be the input of the network, and stores it in the input layer.

Parameters

<i>input</i>	Pointer to the array containing the input data.
<i>size</i>	Size of the input data array.

5.4.3.9 `void NeuralNetwork::setInput (std::vector< double > input)`

Sets a vector to be the input of the network, and stores it in the input layer.

Parameters

<i>input</i>	Vector to set as the input of the network.
--------------	--

5.4.3.10 `void NeuralNetwork::setWeights (std::vector< Matrix * > theta)`

Sets the weights of the network.

The weights are stored in matrices, and there is a matrix for each layer that connects with a previous one (i.e. the input layer has no weights associated).

Each neuron weights are stored in rows, and the dimensions of the matrix should be $n \times m$, where n is the number of neurons in the current layer (not including bias unit) and m is the number of neurons in the previous layer (including bias unit).

The neural network does not store the values of the weights, only the direction of the memory where those values are stored, so that big matrices are not copied several times in the computer's memory.

Parameters

<i>theta</i>	Vector containing the directions of the weight matrices.
--------------	--

5.4.4 Member Data Documentation

5.4.4.1 `std::vector< int > NeuralNetwork::dimensions` [private]

Vector containing the dimensions of the **NeuralNetwork** (p. 20).

5.4.4.2 `std::vector< Layer > NeuralNetwork::hiddenLayer` [private]

Vector containing all the hidden layers.

5.4.4.3 `Layer * NeuralNetwork::inputLayer` [private]

First layer of the network, whose function is to store the input data.

5.4.4.4 `int NeuralNetwork::l` [private]

Number of layer in the network.

5.4.4.5 `std::vector< double > NeuralNetwork::output` [private]

Vector containing the output data.

The output of the network is a vector containing the probability, between 0 and 1, that the input data belongs to the category represented by each output neuron.

5.4.4.6 `Layer * NeuralNetwork::outputLayer` [private]

Last layer of the network, calculates and stores the output data.

5.4.4.7 `std::vector< Matrix * > NeuralNetwork::weights` [private]

Vector containing pointers to the weight matrices.

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

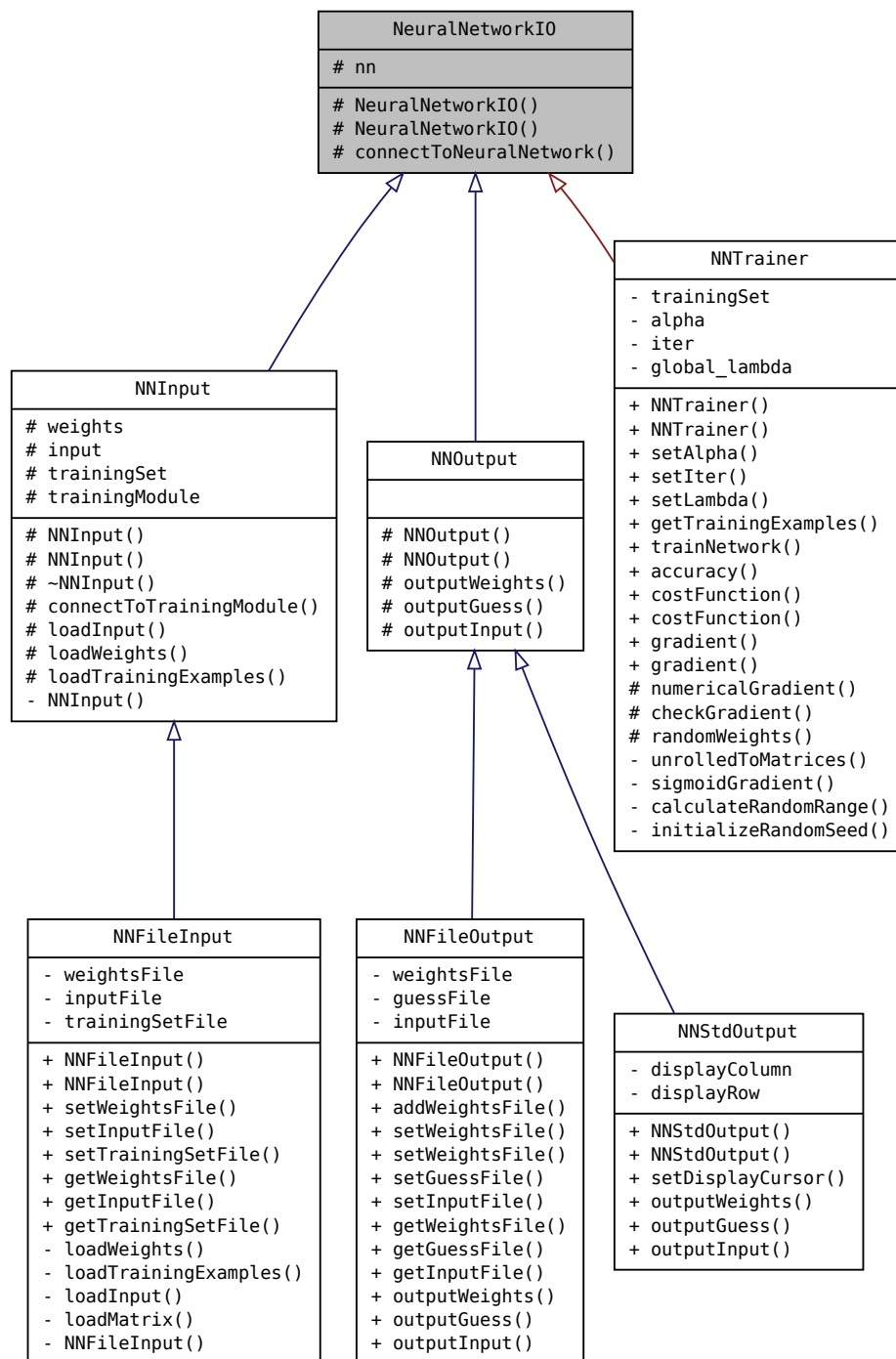
- neuralnetwork.h
- neuralnetwork.cpp

5.5 NeuralNetworkIO Class Reference

Generic input/output interface of a neural network to communicate and exchange data with the outside (std I/O, files, etc).

```
#include <neuralnetworkio.h>
```

Inheritance diagram for NeuralNetworkIO:



Protected Member Functions

- **NeuralNetworkIO ()**
Default constructor.
- **NeuralNetworkIO (NeuralNetwork &nn)**
Creates a I/O interface and connects it to a **NeuralNetwork** (p. 20).
- **void connectToNeuralNetwork (NeuralNetwork &nn)**
Connects a I/O interface to a **NeuralNetwork** (p. 20).

Protected Attributes

- **NeuralNetwork * nn**

*Pointer to a **NeuralNetwork** (p. 20) to which the I/O interface is connected.*

5.5.1 Detailed Description

Generic input/output interface of a neural network to communicate and exchange data with the outside (std I/O, files, etc).

All classes that implement I/O interface with the neural network should inherit from this one.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 NeuralNetworkIO::NeuralNetworkIO () [inline, protected]

Default constructor.

Just creates a I/O interface, connection with a **NeuralNetwork** (p. 20) has to be done later.

5.5.2.2 NeuralNetworkIO::NeuralNetworkIO (NeuralNetwork & nn) [inline, protected]

Creates a I/O interface and connects it to a **NeuralNetwork** (p. 20).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
-----------	---

5.5.3 Member Function Documentation

5.5.3.1 void NeuralNetworkIO::connectToNeuralNetwork (NeuralNetwork & nn) [inline, protected]

Connects a I/O interface to a **NeuralNetwork** (p. 20).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
-----------	---

5.5.4 Member Data Documentation

5.5.4.1 NeuralNetwork * NeuralNetworkIO::nn [protected]

Pointer to a **NeuralNetwork** (p. 20) to which the I/O interface is connected.

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

- neuralnetworkio.h

5.6 Neuron Class Reference

Basic unit of a neural network.

```
#include <neuron.h>
```

Public Member Functions

- **Neuron ()**

Default constructor.

- void **refresh ()**

Calculates the new output of the neuron given the input.

- virtual double **getOutput** ()
Returns the current value at the output.
- std::vector< double > **getWeight** ()
Returns a vector containing the weights of the neuron.
- double **getWeight** (int i)
Returns the value of the weight of the ith dendrite.
- int **getNumDendrites** ()
Returns the number of connections of the neuron.
- void **addConnection** (Neuron &neuronToBeAdded)
Adds a connection to another neuron in the network.
- void **operator<<** (Neuron &neuronToBeAdded)
Adds a connection to another neuron in the network.
- void **setOutput** (double)
Manually set the output of the neuron.
- void **setWeight** (std::vector< double * > weights)
Changes all the variables storing the weights of the dendrites.
- void **setWeight** (int i, double newWeight)
Changes the value of the weight of the ith dendrite.
- void **setWeight** (int i, double *newWeight)
Changes the variable storing the weight of the ith dendrite.

Private Member Functions

- double **activation** (double n)
Calculates the activation function of the neuron.

Private Attributes

- double **axon**
Stores the value of the output of the neuron.
- std::vector< Dendrite > **dendrite**
Stores the data of the neuron connections and their weights.

5.6.1 Detailed Description

Basic unit of a neural network.

The neuron is the basic unit of a neural network. It is composed of some inputs called "dendrites", that connect to other neurons in the network and an output called "axon".

The value of the output of the neuron is the sum of the values of the previous neurons multiplied by each weight, and then this sum is passed to an activation function that calculates the output.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Neuron::Neuron ()

Default constructor.

5.6.3 Member Function Documentation

5.6.3.1 `double Neuron::activation (double n) [private]`

Calculates the activation function of the neuron.

Currently it uses the 'sigmoid function' to calculate the output value of the network, but there are some other function that could be also used.

Parameters

<i>n</i>	Sum of the product of outputs of previous neurons times their weights.
----------	--

5.6.3.2 `void Neuron::addConnection (Neuron & neuronToBeAdded)`

Adds a connection to another neuron in the network.

Parameters

<i>neuronToBeAdded</i>	Neuron (p. 25) to be connected with this neuron.
------------------------	---

5.6.3.3 `int Neuron::getNumDendrites ()`

Returns the number of connections of the neuron.

Returns

Number of connections of the neuron.

5.6.3.4 `double Neuron::getOutput () [virtual]`

Returns the current value at the output.

Returns

Current value at the output.

5.6.3.5 `double Neuron::getWeight (int i)`

Returns the value of the weight of the *i*th dendrite.

Parameters

<i>i</i>	Index of the dendrite selected.
----------	---------------------------------

Returns

Value of the weight of the *i*th dendrite.

5.6.3.6 `std::vector< double > Neuron::getWeight ()`

Returns a vector containing the weights of the neuron.

Returns

Vector containing the weights of the neuron.

5.6.3.7 `void Neuron::operator<< (Neuron & neuronToBeAdded)`

Adds a connection to another neuron in the network.

Parameters

<i>neuronToBeAdded</i>	Neuron (p. 25) to be connected with this neuron.
------------------------	---

5.6.3.8 void Neuron::refresh ()

Calculates the new output of the neuron given the input.

5.6.3.9 void Neuron::setOutput (double *value*)

Manually set the output of the neuron.

This is used to build the input layer, as well as to implement the "Bias unit" of the layer, and for debugging purposes.

Parameters

<i>value</i>	Value that the neuron will output.
--------------	------------------------------------

5.6.3.10 void Neuron::setWeight (int *i*, double * *newWeight*)

Changes the variable storing the weight of the *i*th dendrite.

Parameters

<i>i</i>	Index of the dendrite to modify.
<i>newWeight</i>	Pointer to the new weight of the dendrite.

5.6.3.11 void Neuron::setWeight (std::vector< double * > *weights*)

Changes all the variables storing the weights of the dendrites.

Parameters

<i>weights</i>	Vector of pointers to the new variables to store the weights of the dendrites.
----------------	--

5.6.3.12 void Neuron::setWeight (int *i*, double *newWeight*)

Changes the value of the weight of the *i*th dendrite.

Parameters

<i>i</i>	Index of the dendrite to modify.
<i>newWeight</i>	New value for the weight of the dendrite.

5.6.4 Member Data Documentation

5.6.4.1 double Neuron::axon [private]

Stores the value of the output of the neuron.

"Axon" is name of the part that sends the output in a real neuron.

5.6.4.2 std::vector< Dendrites > Neuron::dendrite [private]

Stores the data of the neuron connections and their weights.

"Dendrite" is the name of the input from other neurons to one neuron in real neurons.

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

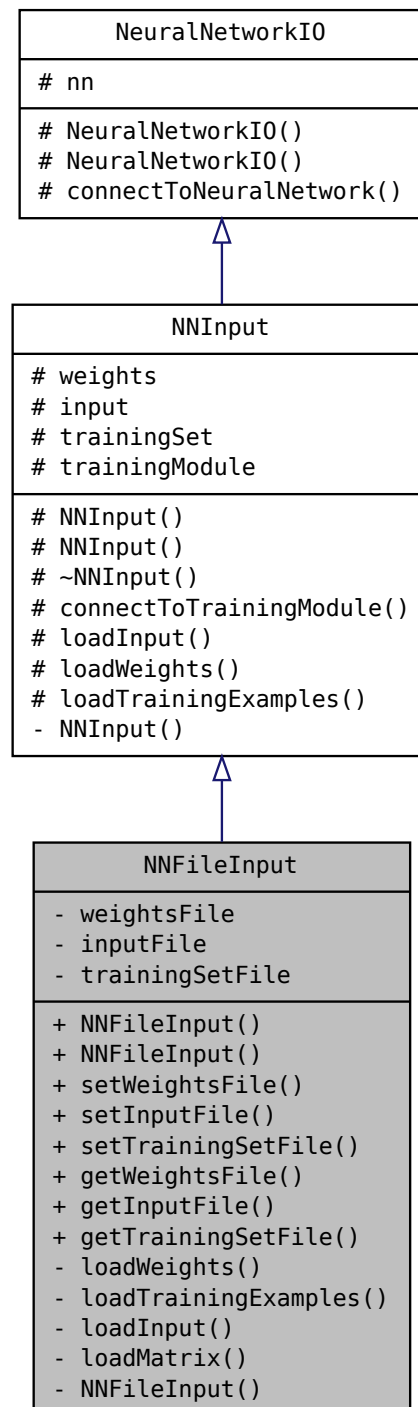
- neuron.h
- neuron.cpp

5.7 NNFileInput Class Reference

File support for data input to **NeuralNetwork** (p. 20).

```
#include <nnfileinput.h>
```

Inheritance diagram for NNFileInput:



Public Member Functions

- **NNFileInput (NeuralNetwork &nn)**
*Creates a file input interface and connects it to a **NeuralNetwork** (p. 20).*
- **NNFileInput (NeuralNetwork &nn, NNTrainer &trainingModule)**
*Creates a file input interface and connects it to a **NeuralNetwork** (p. 20) and **NNTrainer** (p. 44).*
- void **setWeightsFile** (const std::string filePath, const int n)
Edits the path containing the nth weight matrix data.
- void **setInputFile** (const std::string filePath)
Selects the path containing the input data.
- void **setTrainingSetFile** (const std::string filePath, const int n)
Edits the path containing the nth training set data.
- std::vector< std::string > **getWeightsFile** ()
Return all file paths for weights files.
- std::string **getInputFile** ()
Return file path for input file.
- std::vector< std::string > **getTrainingSetFile** ()
Return all file paths for the training set.

Private Member Functions

- virtual void **loadWeights** ()
*Loads the input data of the **NeuralNetwork** (p. 20) from a file, and sends it to the network.*
- virtual void **loadTrainingExamples** ()
*Loads the weights of the **NeuralNetwork** (p. 20) from a file, and sends them to the network.*
- virtual void **loadInput** ()
*Loads the training examples data for the **NeuralNetwork** (p. 20) from a file, and sends it to the network.*
- **Matrix * loadMatrix** (const std::string filePath)
*Loads a new **Matrix** (p. 12) from a file.*
- **NNFileInput** ()
Default constructor is private.

Private Attributes

- std::vector< std::string > **weightsFile**
*Vector containing the paths to the files containing the **NeuralNetwork** (p. 20) weights.*
- std::string **inputFile**
String containing the path to the input file.
- std::vector< std::string > **trainingSetFile**
Vector containing the paths to the training set data.

5.7.1 Detailed Description

File support for data input to **NeuralNetwork** (p. 20).

Loads the data for the input, weights and training examples from different files.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 NNFileInput::NNFileInput (NeuralNetwork & nn)

Creates a file input interface and connects it to a **NeuralNetwork** (p. 20).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
-----------	---

5.7.2.2 NNFileInput::NNFileInput (**NeuralNetwork** & *nn*, **NNTrainer** & *trainingModule*)

Creates a file input interface and connects it to a **NeuralNetwork** (p. 20) and **NNTrainer** (p. 44).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
<i>traininModule</i>	NNTrainer (p. 44) to connect to.

5.7.2.3 NNFileInput::NNFileInput () [private]

Default constructor is private.

5.7.3 Member Function Documentation

5.7.3.1 std::string NNFileInput::getInputFile ()

Return file path for input file.

Returns

String containing file path for input file.

5.7.3.2 std::vector< std::string > NNFileInput::getTrainingSetFile ()

Return all file paths for the training set.

Returns

Vector of strings containing all file paths for the training set.

5.7.3.3 std::vector< std::string > NNFileInput::getWeightsFile ()

Return all file paths for weights files.

Returns

Vector of strings containing all file paths for weights files.

5.7.3.4 void NNFileInput::loadInput () [private, virtual]

Loads the training examples data for the **NeuralNetwork** (p. 20) from a file, and sends it to the network.

Implements **NNInput** (p. 39).

5.7.3.5 **Matrix** * NNFileInput::loadMatrix (const std::string *filePath*) [private]

Loads a new **Matrix** (p. 12) from a file.

Warning

Matrices created with this function have to be deallocated later.

Parameters

<i>filePath</i>	String containing the path to the file.
-----------------	---

5.7.3.6 void NNFileInput::loadTrainingExamples () [private, virtual]

Loads the weights of the **NeuralNetwork** (p. 20) from a file, and sends them to the network.

Todo

If dimensions are nonconsistent, it stills tries to load the data.

Implements **NNInput** (p. 39).

5.7.3.7 void NNFileInput::loadWeights () [private, virtual]

Loads the input data of the **NeuralNetwork** (p. 20) from a file, and sends it to the network.

Implements **NNInput** (p. 39).

5.7.3.8 void NNFileInput::setInputFile (const std::string filePath)

Selects the path containing the input data.

The input data can be loaded from a file containing a row vector (all data in a row, separated by spaces), a column vector (each number in a row) or as a **Matrix** (p. 12), that will be converted to a vector.

Parameters

<i>filePath</i>	String containing the path to the file.
-----------------	---

5.7.3.9 void NNFileInput::setTrainingSetFile (const std::string filePath, const int n)

Edits the path containing the nth training set data.

The position with index 0 is for the inputs of the network, and the position with index 1 is for the expected outputs of the network.

Parameters

<i>filePath</i>	String containing the path to the file.
<i>n</i>	Index of path to change.

5.7.3.10 void NNFileInput::setWeightsFile (const std::string filePath, const int n)

Edits the path containing the nth weight matrix data.

Parameters

<i>filePath</i>	String containing the path to the file.
<i>n</i>	Index of path to change.

5.7.4 Member Data Documentation

5.7.4.1 std::string NNFileInput::inputFile [private]

String containing the path to the input file.

5.7.4.2 std::vector< std::string > NNFileInput::trainingSetFile [private]

Vector containing the paths to the training set data.

5.7.4.3 std::vector< std::string > NNFileInput::weightsFile [private]

Vector containing the paths to the files containing the **NeuralNetwork** (p. 20) weights.

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

- nnfileinput.h

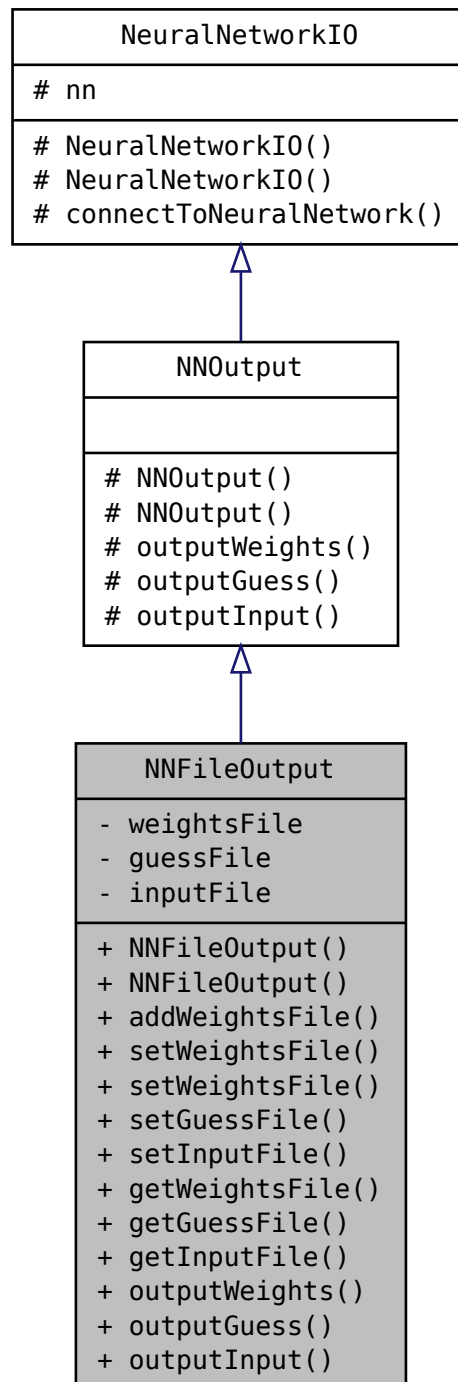
- nnfileinput.cpp

5.8 NNFileOutput Class Reference

File support for data output from **NeuralNetwork** (p. 20).

```
#include <nnfileoutput.h>
```

Inheritance diagram for NNFileOutput:



Public Member Functions

- **NNFileOutput** ()
Default constructor.
- **NNFileOutput (NeuralNetwork &nn)**
*Creates a file output interface and connects it to a **NeuralNetwork** (p. 20).*
- void **addWeightsFile** (const std::string filePath)
Adds one path to store weight matrix data.
- void **setWeightsFile** (const std::string filePath, const int n)
Selects the path to store the nth weight matrix data.
- void **setWeightsFile** (const std::string filePath)
Generates filenames for the matrix data from given file path.
- void **setGuessFile** (const std::string filePath)
Selects the path to store the guess vector of the network.
- void **setInputFile** (const std::string filePath)
Selects the path to store the input data.
- std::vector< std::string > **getWeightsFile** ()
Return all file paths for weights files.
- std::string **getGuessFile** ()
Return file path for guess file.
- std::string **getInputFile** ()
Return file path for input as ascii image file.
- virtual void **outputWeights** ()
*Takes the weights from the **NeuralNetwork** (p. 20), and saves them to a file.*
- virtual void **outputGuess** ()
*Takes output vector data from the **NeuralNetwork** (p. 20), and saves it to a file.*
- virtual void **outputInput** ()
*Takes input vector data from the **NeuralNetwork** (p. 20) and saves it to a file.*

Private Attributes

- std::vector< std::string > **weightsFile**
*Vector containing the paths to the files to store the **NeuralNetwork** (p. 20) weights.*
- std::string **guessFile**
String containing the path to the file that will store the network guess.
- std::string **inputFile**
String containing the path to the input file.

5.8.1 Detailed Description

File support for data output from **NeuralNetwork** (p. 20).

Saves weights, output vector and input data displayed graphically to a file.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 NNFileOutput::NNFileOutput ()

Default constructor.

5.8.2.2 NNFileOutput::NNFileOutput (NeuralNetwork & nn) [inline]

Creates a file output interface and connects it to a **NeuralNetwork** (p. 20).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
-----------	---

5.8.3 Member Function Documentation

5.8.3.1 void NNFileOutput::addWeightsFile (const std::string *filePath*)

Adds one path to store weight matrix data.

Parameters

<i>filePath</i>	String containing the path to the file.
-----------------	---

5.8.3.2 std::string NNFileOutput::getGuessFile ()

Return file path for guess file.

Returns

String containing file path for guess file.

5.8.3.3 std::string NNFileOutput::getInputFile ()

Return file path for input as ascii image file.

Returns

String containing file path for input as ascii image file.

5.8.3.4 std::vector< std::string > NNFileOutput::getWeightsFile ()

Return all file paths for weights files.

Returns

Vector of strings containing all file paths for weights files.

5.8.3.5 void NNFileOutput::outputGuess () [virtual]

Takes output vector data from the **NeuralNetwork** (p. 20), and saves it to a file.

Implements **NNOutput** (p. 41).

5.8.3.6 void NNFileOutput::outputInput () [virtual]

Takes input vector data from the **NeuralNetwork** (p. 20) and saves it to a file.

The input data will be stored as a picture composed of ascii characters. Only useful for data visualization.

Implements **NNOutput** (p. 41).

5.8.3.7 void NNFileOutput::outputWeights () [virtual]

Takes the weights from the **NeuralNetwork** (p. 20), and saves them to a file.

Implements **NNOutput** (p. 41).

5.8.3.8 void NNFileOutput::setGuessFile (const std::string *filePath*)

Selects the path to store the guess vector of the network.

Parameters

<i>filePath</i>	String containing the path to the file.
-----------------	---

5.8.3.9 void NNFileOutput::setInputFile (const std::string *filePath*)

Selects the path to store the input data.

Parameters

<i>filePath</i>	String containing the path to the file.
-----------------	---

5.8.3.10 void NNFileOutput::setWeightsFile (const std::string *filePath*)

Generates filenames for the matrix data from given file path.

Parameters

<i>filePath</i>	String containing the path to the file.
-----------------	---

5.8.3.11 void NNFileOutput::setWeightsFile (const std::string *filePath*, const int *n*)

Selects the path to store the *n*th weight matrix data.

Parameters

<i>filePath</i>	String containing the path to the file.
<i>n</i>	Index of path to change.

5.8.4 Member Data Documentation

5.8.4.1 std::string NNFileOutput::guessFile [private]

String containing the path to the file that will store the network guess.

5.8.4.2 std::string NNFileOutput::inputFile [private]

String containing the path to the input file.

5.8.4.3 std::string NNFileOutput::weightsFile [private]

Vector containing the paths to the files to store the **NeuralNetwork** (p. 20) weights.

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

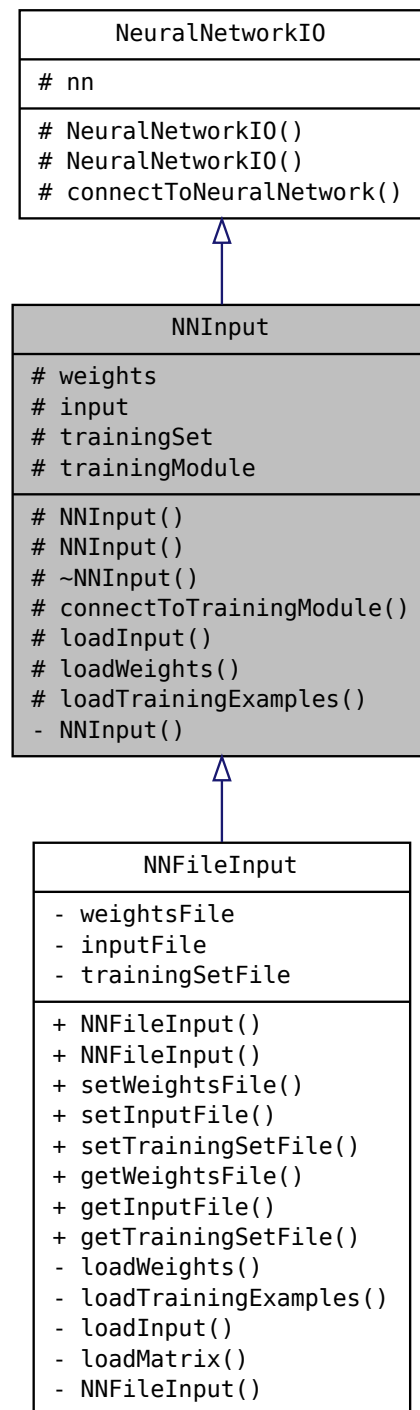
- nnfileoutput.h
- nnfileoutput.cpp

5.9 NNInput Class Reference

Generic input interface of a **NeuralNetwork** (p. 20) to communicate and exchange data with the outside (STD input, input files, etc).

```
#include <nninput.h>
```

Inheritance diagram for NNInput:



Protected Member Functions

- **NNInput (NeuralNetwork &nn)**

*Creates an input interface and connects it to a **NeuralNetwork** (p. 20).*

- **NNInput (NeuralNetwork &nn, NNTrainer &trainingModule)**

*Creates a input interface and connects it to a **NeuralNetwork** (p. 20) and **NNTrainer** (p. 44).*

- virtual `~NNInput ()`
Default destructor.
- void **connectToTrainingModule** (**NNTrainer** &**trainingModule**)
- virtual void **loadInput** ()=0
*Loads the input data of the **NeuralNetwork** (p. 20), and sends it to the network.*
- virtual void **loadWeights** ()=0
*Loads the weights of the **NeuralNetwork** (p. 20), and sends them to the network.*
- virtual void **loadTrainingExamples** ()=0
*Loads the training examples data for the **NeuralNetwork** (p. 20), and sends them to the network.*

Protected Attributes

- `std::vector< Matrix * > weights`
*Vector containing the weights that will be passed to the **NeuralNetwork** (p. 20).*
- `Matrix * input`
*Input vector that will be passed to the **NeuralNetwork** (p. 20), stored as a **Matrix** (p. 12).*
- `std::vector< TrainingExample > trainingSet`
*Vector containing the training examples that will be passed to the **NeuralNetwork** (p. 20).*
- `NNTrainer * trainingModule`
Pointer to the training module to use the training examples:

Private Member Functions

- **NNInput** ()
Default constructor is private.

5.9.1 Detailed Description

Generic input interface of a **NeuralNetwork** (p. 20) to communicate and exchange data with the outside (STD input, input files, etc).

All classes that implement an input interface with the **NeuralNetwork** (p. 20) should inherit from this one.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 NNInput::NNInput (**NeuralNetwork** & *nn*) [protected]

Creates an input interface and connects it to a **NeuralNetwork** (p. 20).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
-----------	---

5.9.2.2 NNInput::NNInput (**NeuralNetwork** & *nn*, **NNTrainer** & *trainingModule*) [protected]

Creates a input interface and connects it to a **NeuralNetwork** (p. 20) and **NNTrainer** (p. 44).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
<i>traininModule</i>	NNTrainer (p. 44) to connect to.

5.9.2.3 NNInput::~~NNInput () [protected, virtual]

Default destructor.

5.9.2.4 NNInput::NNInput () [private]

Default constructor is private.

5.9.3 Member Function Documentation

5.9.3.1 void NNInput::connectToTrainingModule (NNTrainer & trainingModule) [protected]

5.9.3.2 virtual void NNInput::loadInput () [protected, pure virtual]

Loads the input data of the **NeuralNetwork** (p. 20), and sends it to the network.

This function should load the input **Matrix** (p. 12) from some source, store it in the variable input and set this input in the **NeuralNetwork** (p. 20).

Implemented in **NNFileInput** (p. 31).

5.9.3.3 virtual void NNInput::loadTrainingExamples () [protected, pure virtual]

Loads the training examples data for the **NeuralNetwork** (p. 20), and sends them to the network.

This function should load the training examples from some source, store it in the variable trainingExample and call the training routine of the **NeuralNetwork** (p. 20).

Implemented in **NNFileInput** (p. 32).

5.9.3.4 virtual void NNInput::loadWeights () [protected, pure virtual]

Loads the weights of the **NeuralNetwork** (p. 20), and sends them to the network.

This function should load the weight matrices from some source, store it in the variable weights and send them to the **NeuralNetwork** (p. 20).

Implemented in **NNFileInput** (p. 32).

5.9.4 Member Data Documentation

5.9.4.1 Matrix * NNInput::input [protected]

Input vector that will be passed to the **NeuralNetwork** (p. 20), stored as a **Matrix** (p. 12).

5.9.4.2 NNTrainer * NNInput::trainingModule [protected]

Pointer to the training module to use the training examples:

5.9.4.3 std::vector< TrainingExample > NNInput::trainingSet [protected]

Vector containing the training examples that will be passed to the **NeuralNetwork** (p. 20).

5.9.4.4 std::vector< Matrix * > NNInput::weights [protected]

Vector containing the weights that will be passed to the **NeuralNetwork** (p. 20).

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

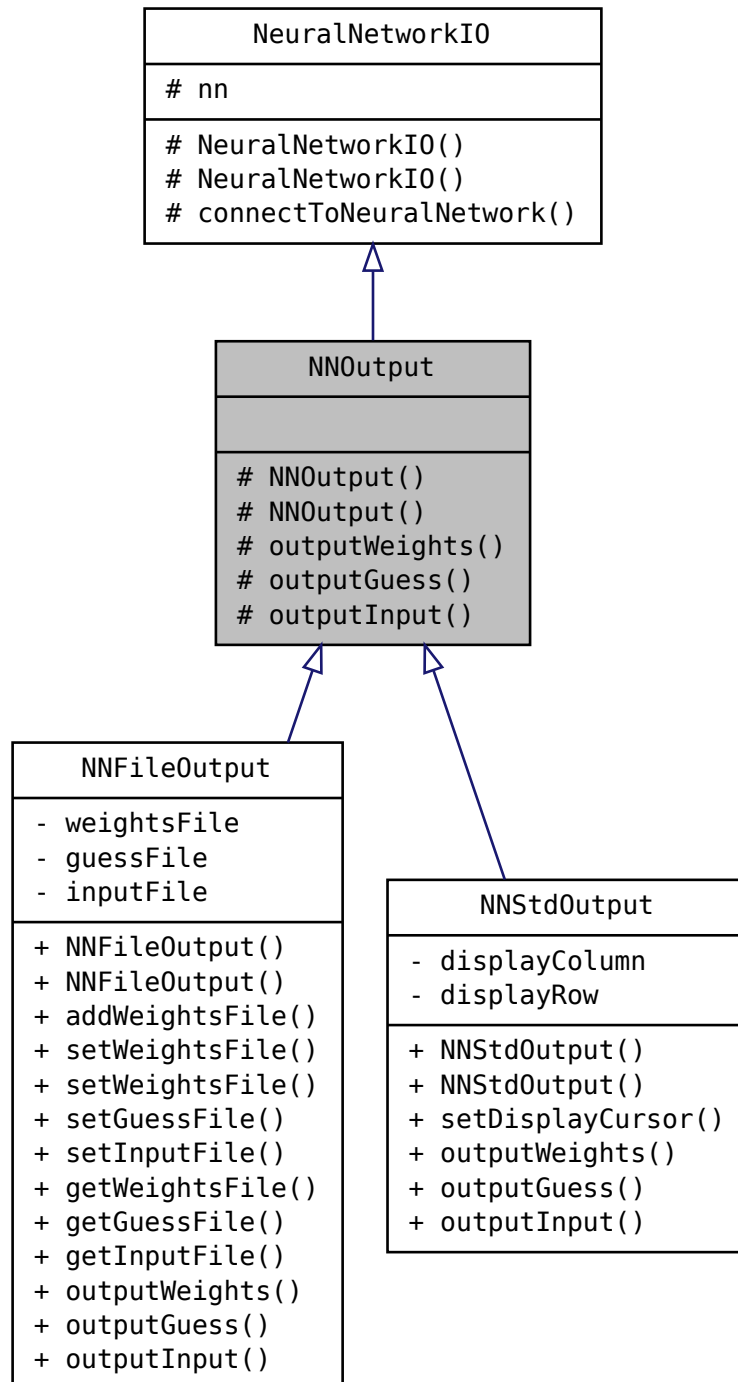
- nninput.h
- nninput.cpp

5.10 NNOutput Class Reference

Generic output interface of a **NeuralNetwork** (p. 20) to communicate and send data to the outside (STD output, save files, etc).

```
#include <nnoutput.h>
```

Inheritance diagram for NNOutput:



Protected Member Functions

- **NNOutput ()**

Default constructor.

- **NNOutput (NeuralNetwork &nn)**

*Creates an output interface and connects it to a **NeuralNetwork** (p. 20).*

- virtual void **outputWeights** ()=0
*Takes the weights from the **NeuralNetwork** (p. 20), sends them to some output.*
- virtual void **outputGuess** ()=0
*Takes output vector data from the **NeuralNetwork** (p. 20), and sends it to some output.*
- virtual void **outputInput** ()=0
*Takes input vector data from the **NeuralNetwork** (p. 20) and sends it to some output.*

5.10.1 Detailed Description

Generic output interface of a **NeuralNetwork** (p. 20) to communicate and send data to the outside (STD output, save files, etc). All classes that implement an output interface for the **NeuralNetwork** (p. 20) should inherit from this one.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 NNOutput::NNOutput () [inline, protected]

Default constructor.

5.10.2.2 NNOutput::NNOutput (NeuralNetwork & nn) [inline, protected]

Creates an output interface and connects it to a **NeuralNetwork** (p. 20).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
-----------	---

5.10.3 Member Function Documentation

5.10.3.1 virtual void NNOutput::outputGuess () [protected, pure virtual]

Takes output vector data from the **NeuralNetwork** (p. 20), and sends it to some output.

This function should load the output vector from the **NeuralNetwork** (p. 20), and output this value somehow (STD output, file, etc).

Implemented in **NNFileOutput** (p. 35), and **NNStdOutput** (p. 43).

5.10.3.2 virtual void NNOutput::outputInput () [protected, pure virtual]

Takes input vector data from the **NeuralNetwork** (p. 20) and sends it to some output.

This function should load the input vector from the **NeuralNetwork** (p. 20), and output this value somehow (STD output, file, etc).

Implemented in **NNFileOutput** (p. 35), and **NNStdOutput** (p. 43).

5.10.3.3 virtual void NNOutput::outputWeights () [protected, pure virtual]

Takes the weights from the **NeuralNetwork** (p. 20), sends them to some output.

This function should take the weight matrices from the **NeuralNetwork** (p. 20), and output their value somehow (STD output, file, etc).

Implemented in **NNFileOutput** (p. 35), and **NNStdOutput** (p. 43).

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

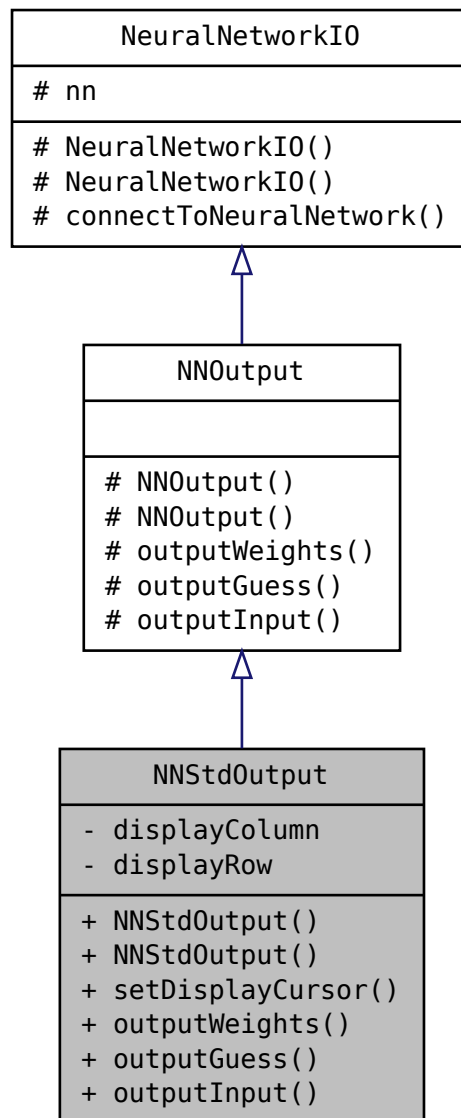
- nnoutput.h

5.11 NNStdOutput Class Reference

Show **NeuralNetwork** (p. 20) data using std output.

```
#include <nnstdoutput.h>
```

Inheritance diagram for NNStdOutput:



Public Member Functions

- **NNStdOutput ()**
Default constructor.
- **NNStdOutput (NeuralNetwork &nn)**
*Creates an std output interface and connects it to a **NeuralNetwork** (p. 20).*
- void **setDisplayCursor** (int col, int row)
Sets the position where the guess output will be drawn in terminal.
- virtual void **outputWeights** ()
*Takes the weights from the **NeuralNetwork** (p. 20), and displays them on the std output.*
- virtual void **outputGuess** ()
*Takes output vector data from the **NeuralNetwork** (p. 20), and displays it on the std output.*
- virtual void **outputInput** ()
*Takes input vector data from the **NeuralNetwork** (p. 20) and displays it on the std output.*

Private Attributes

- int **displayColumn**
Stores the starting column of guess output.
- int **displayRow**
Stores the starting row of guess output.

5.11.1 Detailed Description

Show **NeuralNetwork** (p. 20) data using std output.

Displays the weights and output vector of a **NeuralNetwork** (p. 20) using the standard output.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 NNStdOutput::NNStdOutput () [inline]

Default constructor.

5.11.2.2 NNStdOutput::NNStdOutput (NeuralNetwork & nn) [inline]

Creates an std output interface and connects it to a **NeuralNetwork** (p. 20).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
-----------	---

5.11.3 Member Function Documentation

5.11.3.1 void NNStdOutput::outputGuess () [virtual]

Takes output vector data from the **NeuralNetwork** (p. 20), and displays it on the std output.

Implements **NNOutput** (p. 41).

5.11.3.2 void NNStdOutput::outputInput () [virtual]

Takes input vector data from the **NeuralNetwork** (p. 20) and displays it on the std output.

Implements **NNOutput** (p. 41).

5.11.3.3 void NNStdOutput::outputWeights () [virtual]

Takes the weights from the **NeuralNetwork** (p. 20), and displays them on the std output.

Implements **NNOutput** (p. 41).

5.11.3.4 void NNStdOutput::setDisplayCursor (int col, int row)

Sets the position where the guess output will be drawn in terminal.

Parameters

<i>col</i>	Column of terminal where the output will start.
<i>row</i>	Row of terminal where the output will start.

5.11.4 Member Data Documentation

5.11.4.1 int NNStdOutput::displayColumn [private]

Stores the starting column of guess output.

5.11.4.2 int NNStdOutput::displayRow [private]

Stores the starting row of guess output.

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

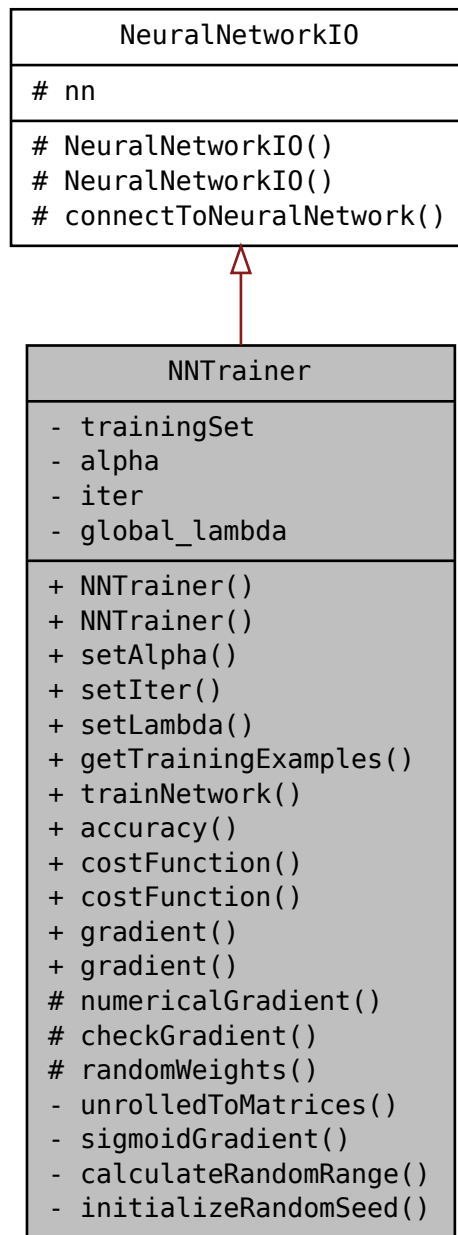
- nnstdoutput.h
- nnstdoutput.cpp

5.12 NNTrainer Class Reference

Trains the weights of a neural network given a training set.

```
#include <nntrainer.h>
```

Inheritance diagram for NNTrainer:



Public Member Functions

- **NNTrainer ()**
Default constructor.
- **NNTrainer (NeuralNetwork &nn)**
*Creates a **NNTrainer** (p. 44) and connects it to a **NeuralNetwork** (p. 20).*
- void **setAlpha** (double **alpha**)
Sets the value of gradient coefficient alpha.
- void **setIter** (int **iter**)
Sets the number of iterations of gradient descend.

- void **setLambda** (double lambda)
- void **getTrainingExamples** (std::vector< **TrainingExample** > &trainingSet)
*Obtain the training set for training the **NeuralNetwork** (p. 20).*
- void **trainNetwork** ()
*Trains the weights of the **NeuralNetwork** (p. 20).*
- double **accuracy** ()
Shows the ability of the current weights to fit a given training set.
- double **costFunction** (const double lambda=0)
Returns the cost of all the examples with the current weight set.
- double **costFunction** (const std::vector< double > theta, const double lambda=0)
Returns the cost of all the examples with the current weight set.
- std::vector< double > **gradient** (const double lambda=0)
Returns the gradient of the cost function with the current weight set.
- std::vector< double > **gradient** (const std::vector< double > theta, const double lambda=0)
Returns the gradient of the cost function with the current weight set.

Protected Member Functions

- std::vector< double > **numericalGradient** (const double lambda=0, const double epsilon=1e-4)
Calculates the numerical gradient to check the gradient implementation.
- bool **checkGradient** (const double lambda=0)
Checks the computation of gradients with backprop in a small neural network.
- void **randomWeights** ()
Generates a new random weight set for training.

Private Member Functions

- std::vector< **Matrix** * > **unrolledToMatrices** (std::vector< double > theta)
Converts unrolled weights into a suitable matrix vector to set in the network.
- double **sigmoidGradient** (double n)
Calculates the derivative of the sigmoid function at point n.
- double **calculateRandomRange** (int layer)
Calculates the range of the random weights taking into account the network dimensions.
- void **initializeRandomSeed** ()
Uses the current time to initialize a seed for random numbers.

Private Attributes

- std::vector< **TrainingExample** > * **trainingSet**
Pointer to the training set loaded into memory.
- double **alpha**
Attenuation coefficient for gradient.
- int **iter**
Number of iterations of gradient descend.
- double **global_lambda**
Regularization coefficient to be applied to gradient calculation.

5.12.1 Detailed Description

Trains the weights of a neural network given a training set.

It also provides a costFunction and a gradient that serve as a base for other **NNTrainer** (p. 44) with different training algorithms.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 NNTrainer::NNTrainer ()

Default constructor.

5.12.2.2 NNTrainer::NNTrainer (**NeuralNetwork** & *nn*)

Creates a **NNTrainer** (p. 44) and connects it to a **NeuralNetwork** (p. 20).

Parameters

<i>nn</i>	NeuralNetwork (p. 20) to connect to.
-----------	---

5.12.3 Member Function Documentation

5.12.3.1 double NNTrainer::accuracy ()

Shows the ability of the current weights to fit a given training set.

Returns

A value between 0 and 1.

5.12.3.2 double NNTrainer::calculateRandomRange (int *layer*) [private]

Calculates the range of the random weights taking into account the network dimensions.

Parameters

<i>layer</i>	Index of the current layer. Index starts at 0. Input layer is not a valid layer as it has no weights associated.
--------------	--

5.12.3.3 bool NNTrainer::checkGradient (const double *lambda* = 0) [protected]

Checks the computation of gradients with backprop in a small neural network.

Parameters

<i>lambda</i>	Regularization coefficient to avoid overfitting.
---------------	--

This function is used for debugging.

5.12.3.4 double NNTrainer::costFunction (const std::vector< double > *theta*, const double *lambda* = 0)

Returns the cost of all the examples with the current weight set.

Parameters

<i>theta</i>	Vector containing all the unrolled weights.
<i>lambda</i>	Regularization coefficient to avoid overfitting.

Returns

Cost of all the training examples with current weight set

5.12.3.5 double NNTrainer::costFunction (const double *lambda* = 0)

Returns the cost of all the examples with the current weight set.

Parameters

<i>lambda</i>	Regularization coefficient to avoid overfitting.
---------------	--

Returns

Cost of all the training examples with current weight set

5.12.3.6 `void NNTrainer::getTrainingExamples (std::vector< TrainingExample > & trainingSet)`

Obtain the training set for training the **NeuralNetwork** (p.20).

5.12.3.7 `std::vector< double > NNTrainer::gradient (const double lambda = 0)`

Returns the gradient of the cost function with the current weight set.

Parameters

<i>theta</i>	Vector containing all the unrolled weights.
<i>lambda</i>	Regularization coefficient to avoid overfitting.

Returns

Gradient of cost function given the current weight set.

5.12.3.8 `std::vector< double > NNTrainer::gradient (const std::vector< double > theta, const double lambda = 0)`

Returns the gradient of the cost function with the current weight set.

Parameters

<i>theta</i>	Vector containing all the unrolled weights.
<i>lambda</i>	Regularization coefficient to avoid overfitting.

Returns

Gradient of cost function given the current weight set.

5.12.3.9 `void NNTrainer::initializeRandomSeed () [private]`

Uses the current time to initialize a seed for random numbers.

5.12.3.10 `std::vector< double > NNTrainer::numericalGradient (const double lambda = 0, const double epsilon = 1e-4) [protected]`

Calculates the numerical gradient to check the gradient implementation.

This function is used for debugging.

Parameters

<i>lambda</i>	Regularization coefficient to avoid overfitting.
<i>epsilon</i>	Increment to use when calculating the gradient.

Warning

This way of calculating the gradient is expensive computationally. It should be used only in small test networks for testing the backpropagation implementation, not for optimization of weight set.

5.12.3.11 `void NNTrainer::randomWeights () [protected]`

Generates a new random weight set for training.

Generates an appropriate random weights set before training, taking into account the network dimensions.

5.12.3.12 `void NNTrainer::setAlpha (double alpha)`

Sets the value of gradient coefficient alpha.

Parameters

<i>alpha</i>	Value to set.
--------------	---------------

5.12.3.13 void NNTrainer::setIter (int *iter*)

Sets the number of iterations of gradient descend.

Parameters

<i>iter</i>	Number of iterations.
-------------	-----------------------

5.12.3.14 void NNTrainer::setLambda (double *lambda*)

the value of regularizaton parameter lambda.

Parameters

<i>lambda</i>	Regularization parameter lambda.
---------------	----------------------------------

5.12.3.15 double NNTrainer::sigmoidGradient (double *n*) [private]

Calculates the derivative of the sigmoid function at point n.

Note

This is not the actual derivative, only works if the input is the output of the sigmoid function, i.e. the input is output of the activation function.

5.12.3.16 void NNTrainer::trainNetwork ()

Trains the weigths of the **NeuralNetwork** (p. 20).

Uses a gradient descend algorithm to train the **NeuralNetwork** (p. 20) weights, according to the training parameters alpha, iter, and lambda and a loaded training set.

5.12.3.17 std::vector< Matrix * > NNTrainer::unrolledToMatrices (std::vector< double > *theta*) [private]

Converts unrolled weights into a suitable matrix vector to set in the network.

Warning

This function reserves memory for the matrices that has to be deallocated manually later.

Parameters

<i>theta</i>	Unrolled vector of weights.
--------------	-----------------------------

Returns

Matrix (p. 12) vector containing weights.

5.12.4 Member Data Documentation

5.12.4.1 double NNTrainer::alpha [private]

Attenuation coefficient for gradient.

5.12.4.2 double NNTrainer::global_lambda [private]

Regularization coefficient to be applied to gradient calculation.

5.12.4.3 `int NNTrainer::iter` `[private]`

Number of iterations of gradient descend.

5.12.4.4 `std::vector< TrainingExample > * NNTrainer::trainingSet` `[private]`

Pointer to the training set loaded into memory.

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

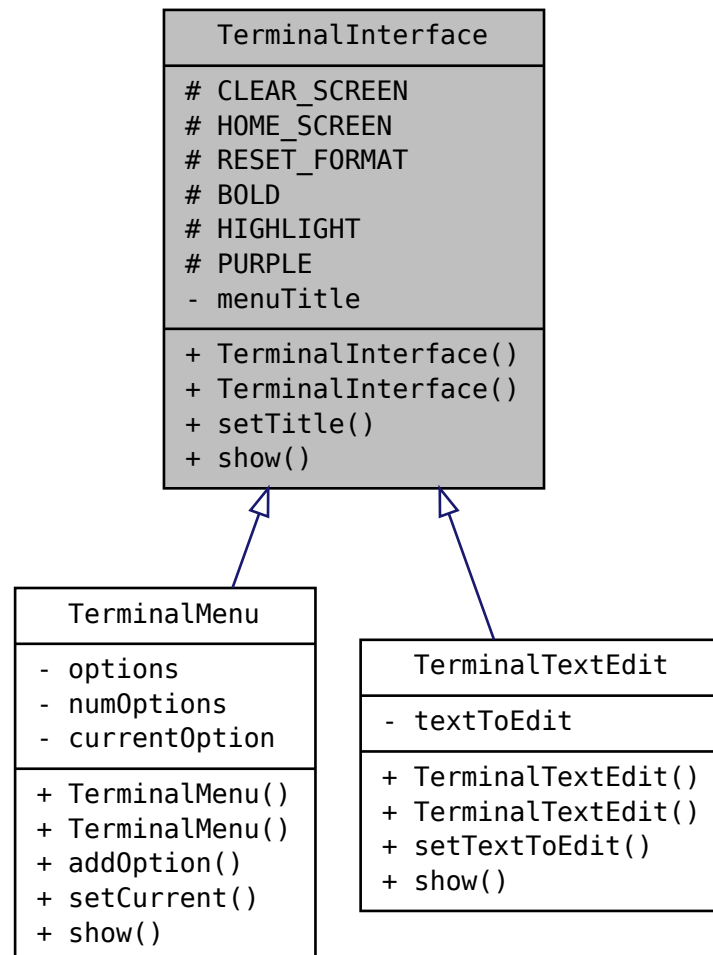
- nntrainer.h
- nntrainer.cpp

5.13 TerminalInterface Class Reference

Base class for creating simple terminal interfaces.

```
#include <terminalinterface.h>
```

Inheritance diagram for TerminalInterface:



Public Member Functions

- **TerminalInterface** ()
Default constructor.
- **TerminalInterface** (const std::string title)
*Creates a **TerminalInterface** (p. 50) with a given title.*
- void **setTitle** (const std::string newTitle)
*Sets the title for the **TerminalInterface** (p. 50).*
- virtual int **show** ()
Show the current interface on terminal.

Static Protected Attributes

- static const std::string **CLEAR_SCREEN** = "\033[2J"
String containing the ANSI escape code for clearing the terminal.
- static const std::string **HOME_SCREEN** = "\033[0;0H"
String containing the ANSI escape code for setting the cursor at position (0,0).
- static const std::string **RESET_FORMAT** = "\033[0m"
String containing the ANSI escape code for setting the default output format.
- static const std::string **BOLD** = "\033[1m"
String containing the ANSI escape code for setting the output format to bold.
- static const std::string **HIGHLIGHT** = "\033[7m"
String containing the ANSI escape code for setting the output format to highlighted text.
- static const std::string **PURPLE** = "\033[0;35m"
String containing the ANSI escape code for setting the output color to purple.

Private Attributes

- std::string **menuTitle**
Title of the interface.

5.13.1 Detailed Description

Base class for creating simple terminal interfaces.
Shows a header on top of the terminal with a given title.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 TerminalInterface::TerminalInterface ()

Default constructor.

5.13.2.2 TerminalInterface::TerminalInterface (const std::string title)

Creates a **TerminalInterface** (p. 50) with a given title.

Parameters

<i>title</i>	Title for the interface.
--------------	--------------------------

5.13.3 Member Function Documentation

5.13.3.1 void TerminalInterface::setTitle (const std::string newTitle)

Sets the title for the **TerminalInterface** (p. 50).

Parameters

<i>newTitle</i>	Title for the interface.
-----------------	--------------------------

5.13.3.2 int TerminalInterface::show () [virtual]

Show the current interface on terminal.

This interface just shows a terminal interface with just a header. On derived classes this function has to implement the main functionality of the interface.

Returns

A value indicating if it has ended without any issue.

Reimplemented in **TerminalMenu** (p. 54), and **TerminalTextEdit** (p. 56).

5.13.4 Member Data Documentation

5.13.4.1 static const std::string TerminalInterface::BOLD = "\033[1m" [static, protected]

String containing the ANSI escape code for setting the output format to bold.

5.13.4.2 static const std::string TerminalInterface::CLEAR_SCREEN = "\033[2J" [static, protected]

String containing the ANSI escape code for clearing the terminal.

5.13.4.3 static const std::string TerminalInterface::HIGHLIGHT = "\033[7m" [static, protected]

String containing the ANSI escape code for setting the output format to highlighted text.

5.13.4.4 static const std::string TerminalInterface::HOME_SCREEN = "\033[0;0H" [static, protected]

String containing the ANSI escape code for setting the cursor at position (0,0).

5.13.4.5 std::string TerminalInterface::menuTitle [private]

Title of the interface.

5.13.4.6 static const std::string TerminalInterface::PURPLE = "\033[0;35m" [static, protected]

String containing the ANSI escape code for setting the output color to purple.

5.13.4.7 static const std::string TerminalInterface::RESET_FORMAT = "\033[0m" [static, protected]

String containing the ANSI escape code for setting the default output format.

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

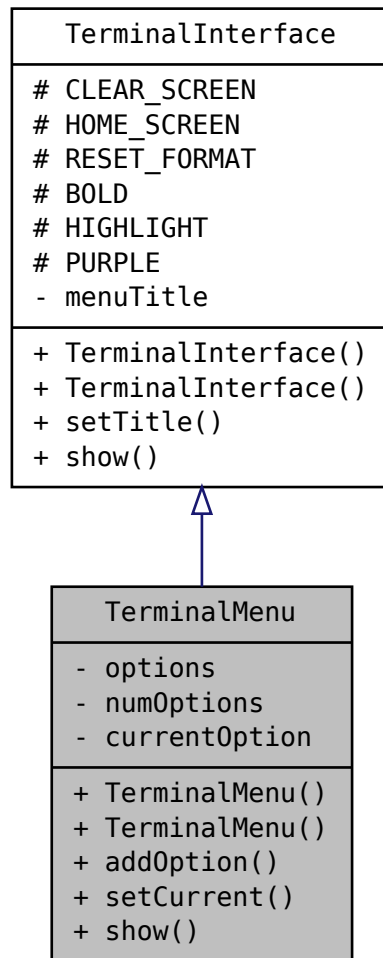
- terminalinterface.h
- terminalinterface.cpp

5.14 TerminalMenu Class Reference

Basic terminal menu interface.

```
#include <terminalmenu.h>
```

Inheritance diagram for TerminalMenu:



Public Member Functions

- **TerminalMenu** ()
Default constructor.
- **TerminalMenu** (const std::string title)
*Creates a **TerminalMenu** (p. 52) with a given title.*
- void **addOption** (const std::string newOption)
Adds an option to the interface.
- void **setCurrent** (const int num)
Sets the current selected option.
- int **show** ()
Shows the menu on terminal.

Private Attributes

- std::vector< std::string > **options**
Vector of strings containing the text of each choice of the menu.

- int **numOptions**
Number of choices in the menu.
- int **currentOption**
Option currently selected.

5.14.1 Detailed Description

Basic terminal menu interface.

Shows a terminal menu with multiple choices and returns the selected one.

This menu can be configured by setting one by one all the possible choices that the menu will show.

The function **show()** (p. 54) will return the index of the selected choice, or -1 if none was selected or an error happened.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 TerminalMenu::TerminalMenu ()

Default constructor.

5.14.2.2 TerminalMenu::TerminalMenu (const std::string *title*)

Creates a **TerminalMenu** (p. 52) with a given title.

Parameters

<i>title</i>	Title for the interface.
--------------	--------------------------

5.14.3 Member Function Documentation

5.14.3.1 void TerminalMenu::addOption (const std::string *newOption*)

Adds an option to the interface.

Parameters

<i>newOption</i>	Text of the new option.
------------------	-------------------------

5.14.3.2 void TerminalMenu::setCurrent (const int *num*)

Sets the current selected option.

Parameters

<i>num</i>	Current selected option.
------------	--------------------------

5.14.3.3 int TerminalMenu::show () [virtual]

Shows the menu on terminal.

Shows the menu on terminal, and waits for user input to select one of the choices. Then, it returns the index (starting at 0) of the selected choice, or -1 in case of error or none of them selected.

Returns

Index of selected option, starting at 0.

Reimplemented from **TerminalInterface** (p. 52).

5.14.4 Member Data Documentation

5.14.4.1 `int TerminalMenu::currentOption` [private]

Option currently selected.

5.14.4.2 `int TerminalMenu::numOptions` [private]

Number of choices in the menu.

5.14.4.3 `std::vector< std::string > TerminalMenu::options` [private]

Vector of strings containing the text of each choice of the menu.

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

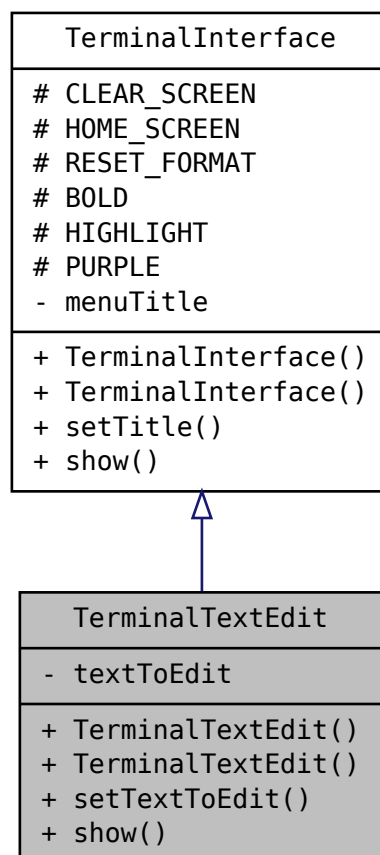
- terminalmenu.h
- terminalmenu.cpp

5.15 TerminalTextEdit Class Reference

Basic text edition on terminal interface.

```
#include <terminaltextedit.h>
```

Inheritance diagram for TerminalTextEdit:



Public Member Functions

- **TerminalTextEdit** (const std::string title)
*Creates a **TerminalTextEdit** (p. 55) with a given title.*
- **TerminalTextEdit** (const std::string title, std::string &textToEdit)
*Creates a **TerminalTextEdit** (p. 55) with a given title and a given string to modify.*
- void **setTextToEdit** (std::string &textToEdit)
*Sets the string that will be modified by the **TerminalTextEdit** (p. 55).*
- int **show** ()
Shows the text edit interface on terminal.

Private Attributes

- std::string * **textToEdit**
Pointer to the string to be modified.

5.15.1 Detailed Description

Basic text edition on terminal interface.

Shows a terminal menu showing the previous value for a string and asking the user to a new value for it.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 TerminalTextEdit::TerminalTextEdit (const std::string title)

Creates a **TerminalTextEdit** (p. 55) with a given title.

Parameters

<i>title</i>	Title for the interface.
--------------	--------------------------

5.15.2.2 TerminalTextEdit::TerminalTextEdit (const std::string title, std::string & textToEdit)

Creates a **TerminalTextEdit** (p. 55) with a given title and a given string to modify.

Parameters

<i>title</i>	Title for the interface.
<i>textToEdit</i>	String to be modified.

5.15.3 Member Function Documentation

5.15.3.1 void TerminalTextEdit::setTextToEdit (std::string & textToEdit)

Sets the string that will be modified by the **TerminalTextEdit** (p. 55).

Parameters

<i>textToEdit</i>	String to be modified.
-------------------	------------------------

5.15.3.2 int TerminalTextEdit::show () [virtual]

Shows the text edit interface on terminal.

Shows a terminal menu showing the previous value for a string and asking the user to a new value for it.

Reimplemented from **TerminalInterface** (p. 52).

5.15.4 Member Data Documentation

5.15.4.1 `std::string * TerminalTextEdit::textToEdit` [private]

Pointer to the string to be modified.

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

- terminaltextedit.h
- terminaltextedit.cpp

5.16 TrainingExample Struct Reference

A given input for the network and the expected output for that input.

```
#include <nntrainer.h>
```

Public Attributes

- `std::vector< double > x`
Input for the network.
- `std::vector< double > y`
Expected output of the network with input x.

5.16.1 Detailed Description

A given input for the network and the expected output for that input.

5.16.2 Member Data Documentation

5.16.2.1 `std::vector<double> TrainingExample::x`

Input for the network.

5.16.2.2 `std::vector<double> TrainingExample::y`

Expected output of the network with input x.

The documentation for this struct was generated from the following file:

- nntrainer.h