

# NNetCpp

Generated by Doxygen 1.8.6

Thu Aug 23 2018 20:21:01



# Contents

<b>1</b>	<b>Namespace Index</b>	<b>1</b>
1.1	Namespace List	1
<b>2</b>	<b>Hierarchical Index</b>	<b>3</b>
2.1	Class Hierarchy	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List	5
<b>4</b>	<b>Namespace Documentation</b>	<b>7</b>
4.1	NNetUtil Namespace Reference	7
4.1.1	Detailed Description	7
4.1.2	Function Documentation	7
4.1.2.1	sigmoid	7
4.1.2.2	squareError	7
4.1.2.3	unitStep	8
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	ConnectedNet Class Reference	9
5.1.1	Detailed Description	9
5.1.2	Constructor & Destructor Documentation	9
5.1.2.1	ConnectedNet	9
5.1.2.2	~ConnectedNet	10
5.1.3	Member Function Documentation	10
5.1.3.1	getDifference	10
5.1.3.2	getOutput	10
5.1.3.3	train	10
5.2	ConstantWeightGenerator Class Reference	10
5.2.1	Detailed Description	11
5.2.2	Constructor & Destructor Documentation	11
5.2.2.1	ConstantWeightGenerator	11
5.2.3	Member Function Documentation	11
5.2.3.1	getWeight	11

5.3	Edge Class Reference	11
5.3.1	Detailed Description	12
5.3.2	Constructor & Destructor Documentation	12
5.3.2.1	Edge	12
5.3.3	Member Function Documentation	12
5.3.3.1	getWeightedDelta	12
5.3.3.2	getWeightedOutput	12
5.3.3.3	setInput	12
5.3.3.4	setOutput	12
5.3.3.5	updateWeight	13
5.4	InputNode Class Reference	13
5.4.1	Detailed Description	13
5.4.2	Constructor & Destructor Documentation	13
5.4.2.1	InputNode	13
5.4.2.2	InputNode	13
5.4.3	Member Function Documentation	14
5.4.3.1	getOutput	14
5.4.3.2	setValue	14
5.5	IterationCondition Class Reference	14
5.5.1	Detailed Description	14
5.5.2	Constructor & Destructor Documentation	15
5.5.2.1	IterationCondition	15
5.5.3	Member Function Documentation	16
5.5.3.1	check	16
5.6	NetTraining Class Reference	16
5.6.1	Detailed Description	16
5.6.2	Constructor & Destructor Documentation	16
5.6.2.1	NetTraining	16
5.6.3	Member Function Documentation	17
5.6.3.1	run	17
5.7	Neuron Class Reference	17
5.7.1	Detailed Description	18
5.7.2	Constructor & Destructor Documentation	18
5.7.2.1	Neuron	18
5.7.2.2	Neuron	18
5.7.3	Member Function Documentation	18
5.7.3.1	calcDelta	18
5.7.3.2	calcOutput	18
5.7.3.3	getDelta	18
5.7.3.4	getOutput	18

5.7.3.5	setInputs	19
5.7.3.6	setOutputs	19
5.7.3.7	sigmoidDelta	19
5.7.4	Member Data Documentation	19
5.7.4.1	delta	19
5.8	Node Class Reference	19
5.8.1	Detailed Description	20
5.8.2	Member Function Documentation	20
5.8.2.1	getOutput	20
5.9	OutputNeuron Class Reference	20
5.9.1	Detailed Description	20
5.9.2	Constructor & Destructor Documentation	21
5.9.2.1	OutputNeuron	21
5.9.2.2	OutputNeuron	21
5.9.2.3	OutputNeuron	21
5.9.3	Member Function Documentation	21
5.9.3.1	calcDelta	21
5.9.3.2	getDifference	21
5.9.3.3	setTarget	21
5.10	RandomWeightGenerator Class Reference	22
5.10.1	Detailed Description	22
5.10.2	Constructor & Destructor Documentation	22
5.10.2.1	RandomWeightGenerator	22
5.10.3	Member Function Documentation	22
5.10.3.1	getWeight	22
5.11	StopCondition Class Reference	23
5.11.1	Detailed Description	23
5.11.2	Member Function Documentation	23
5.11.2.1	check	23
5.12	TrainingData Class Reference	23
5.12.1	Detailed Description	24
5.12.2	Constructor & Destructor Documentation	24
5.12.2.1	TrainingData	24
5.12.3	Member Function Documentation	24
5.12.3.1	getInput	24
5.12.3.2	getTarget	24
5.13	WeightGenerator Class Reference	24
5.13.1	Detailed Description	25
5.13.2	Member Function Documentation	25
5.13.2.1	getWeight	25

<a href="#">Index</a>	26
-----------------------	----

# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">NNetUtil</a> . . . . .	7
------------------------------------	---





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ConnectedNet . . . . .	9
Edge . . . . .	11
NetTraining . . . . .	16
Node . . . . .	19
InputNode . . . . .	13
Neuron . . . . .	17
OutputNeuron . . . . .	20
StopCondition . . . . .	23
IterationCondition . . . . .	14
TrainingData . . . . .	23
WeightGenerator . . . . .	24
ConstantWeightGenerator . . . . .	10
RandomWeightGenerator . . . . .	22



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">ConnectedNet</a>	9
<a href="#">ConstantWeightGenerator</a>	10
<a href="#">Edge</a>	11
<a href="#">InputNode</a>	13
<a href="#">IterationCondition</a>	14
<a href="#">NetTraining</a>	16
<a href="#">Neuron</a>	17
<a href="#">Node</a>	19
<a href="#">OutputNeuron</a>	20
<a href="#">RandomWeightGenerator</a>	22
<a href="#">StopCondition</a>	23
<a href="#">TrainingData</a>	23
<a href="#">WeightGenerator</a>	24



## Chapter 4

# Namespace Documentation

### 4.1 NNetUtil Namespace Reference

#### Functions

- double [unitStep](#) (double x)
- double [sigmoid](#) (double x)
- double [squareError](#) (const std::vector< double > &v1, const std::vector< double > &v2)

#### 4.1.1 Detailed Description

Utility functions for neural networks

#### 4.1.2 Function Documentation

##### 4.1.2.1 double NNetUtil::sigmoid ( double x )

Sigmoid function

$$y(x) = 1 / (1 + \exp(-x))$$

Parameters

x	input to sigmoid
---	------------------

Returns

output of sigmoid

##### 4.1.2.2 double NNetUtil::squareError ( const std::vector< double > &v1, const std::vector< double > &v2 )

Calculate the square error of two vectors Scaled with (1/2) to normalize derivative

$$E(v1, v2) = (1/2) |v2 - v1|^2$$

Parameters

---

<i>v1</i>	first vector
<i>v2</i>	second vector

**Returns**

the square error of the two vectors

**4.1.2.3 double NNetUtil::unitStep ( double *x* )**

Unit step function

$y(x) = 0$  if  $x < 0$

$y(x) = 1$  if  $x \geq 0$

**Parameters**

<i>x</i>	input to step function
----------	------------------------

**Returns**

step function of input

## Chapter 5

# Class Documentation

### 5.1 ConnectedNet Class Reference

```
#include <ConnectedNet.h>
```

#### Public Member Functions

- [ConnectedNet](#) (int inputNodes, std::vector< int > hiddenLayers, int outputNodes, [WeightGenerator](#) \*weightGen=nullptr)
- [~ConnectedNet](#) ()
- std::vector< double > [getOutput](#) (std::vector< double > inputValues)
- void [train](#) ([TrainingData](#) &tData, double learningRate)
- std::vector< double > [getDifference](#) ()

#### 5.1.1 Detailed Description

A fully connected Neural Network

#### 5.1.2 Constructor & Destructor Documentation

5.1.2.1 [ConnectedNet::ConnectedNet](#) ( int *inputNodes*, std::vector< int > *hiddenLayers*, int *outputNodes*, [WeightGenerator](#) \* *weightGen* = `nullptr` )

Create a new [ConnectedNet](#) of given dimensions.

For example: For a 3x5x4x4x2 network one would put

inputNodes - 3

hiddenLayers - [5, 4, 4]

outputNodes - 2

#### Parameters

<i>inputNodes</i>	amount of inputs to the network
<i>hiddenLayers</i>	vector of dimensions of hidden layers, number at each index becomes amount of neurons in hidden layer at that index

<i>outputNodes</i>	amount of outputs from the network
<i>weightGen</i>	generator to use for initializing weights in the network, if left a nullptr it is replaced with a default random generator

#### 5.1.2.2 ConnectedNet::~~ConnectedNet ( )

Destructor for [ConnectedNet](#), clears all memory allocated

### 5.1.3 Member Function Documentation

#### 5.1.3.1 std::vector< double > ConnectedNet::getDifference ( )

Get a vector of difference between target and output

Returns

the difference vector

#### 5.1.3.2 std::vector< double > ConnectedNet::getOutput ( std::vector< double > *inputValues* )

Present an input to the [ConnectedNet](#) and get the output

Parameters

<i>inputValues</i>	input vector to the net
--------------------	-------------------------

Returns

the output vector of the net

#### 5.1.3.3 void ConnectedNet::train ( TrainingData & *tData*, double *learningRate* )

Train the net on one input - target vector pair with speed learningrate returns the error

Parameters

<i>tData</i>	a piece of training data to use for training the net
<i>learningRate</i>	rate to adjust weights in training

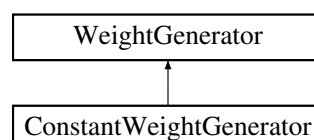
The documentation for this class was generated from the following files:

- src/ConnectedNet.h
- src/ConnectedNet.cpp

## 5.2 ConstantWeightGenerator Class Reference

```
#include <ConstantWeightGenerator.h>
```

Inheritance diagram for ConstantWeightGenerator:





## Public Member Functions

- [ConstantWeightGenerator](#) (double weight)
- double [getWeight](#) ()

### 5.2.1 Detailed Description

[WeightGenerator](#) that always outputs a constant weight

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 [ConstantWeightGenerator::ConstantWeightGenerator](#) ( double *weight* )

Create a new [ConstantWeightGenerator](#) that always outputs weight

Parameters

<i>weight</i>	the constant weight to output
---------------	-------------------------------

### 5.2.3 Member Function Documentation

#### 5.2.3.1 double [ConstantWeightGenerator::getWeight](#) ( ) [virtual]

Get the constant weight

Inherits:

Get a new weight from the generator

Returns

the new weight

Implements [WeightGenerator](#).

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

- src/netbuilding/weightgenerators/ConstantWeightGenerator.h
- src/netbuilding/weightgenerators/ConstantWeightGenerator.cpp

## 5.3 Edge Class Reference

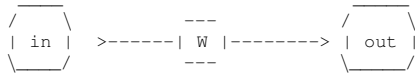
```
#include <Edge.h>
```

## Public Member Functions

- [Edge](#) (double startWeight, [Node](#) \*input=nullptr, [Neuron](#) \*output=nullptr)
- void [setInput](#) ([Node](#) \*input)
- void [setOutput](#) ([Neuron](#) \*output)
- void [updateWeight](#) (double learningRate)
- double [getWeightedOutput](#) ()
- double [getWeightedDelta](#) ()

### 5.3.1 Detailed Description

[Edge](#) between two nodes with weight  $W$



### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 `Edge::Edge ( double startWeight, Node * input = nullptr, Neuron * output = nullptr )`

Create a new [Edge](#) between 2 Nodes. Start with given weight

Parameters

<i>startWeight</i>	initial weight of the edge
<i>input</i>	start node of the edge
<i>output</i>	end node of the edge

### 5.3.3 Member Function Documentation

#### 5.3.3.1 `double Edge::getWeightedDelta ( )`

Get the delta from the output [Neuron](#) weighted for back propagation

Returns

the delta of output node weighted with the current edge weight

#### 5.3.3.2 `double Edge::getWeightedOutput ( )`

Get the output from the edge

Returns

the input weighted with the current edge weight

#### 5.3.3.3 `void Edge::setInput ( Node * input )`

Set the [Node](#) this [Edge](#) starts at

Parameters

<i>input</i>	node to set for input of edge
--------------	-------------------------------

#### 5.3.3.4 `void Edge::setOutput ( Neuron * output )`

Set the [Neuron](#) this [Edge](#) ends at Note: must be [Neuron](#) (not just [Node](#)) to allow for retrieving delta for training

## Parameters

<i>output</i>	node to set for output of edge, must be a neuron not just a node to allow for retrieving delta for training
---------------	---

5.3.3.5 void Edge::updateWeight ( double *learningRate* )

Update the weight for the edge based on the backpropagation algorithm

## Parameters

<i>learningRate</i>	distance to update weight
---------------------	---------------------------

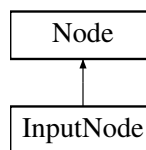
The documentation for this class was generated from the following files:

- src/netbuilding/Edge.h
- src/netbuilding/Edge.cpp

## 5.4 InputNode Class Reference

```
#include <InputNode.h>
```

Inheritance diagram for InputNode:



## Public Member Functions

- [InputNode](#) ()
- [InputNode](#) (double value)
- double [getOutput](#) ()
- void [setValue](#) (double value)

### 5.4.1 Detailed Description

A node to be used for input to networks. Its output value can be set.

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 InputNode::InputNode ( )

Create a new input [Node](#)

#### 5.4.2.2 InputNode::InputNode ( double *value* )

Create a new input [Node](#) with the given value

## Parameters

<i>value</i>	the value for the <a href="#">InputNode</a> to output
--------------	---

### 5.4.3 Member Function Documentation

#### 5.4.3.1 `double InputNode::getOutput ( )` [virtual]

Get the output value of the input node

inherits:

Get the output of the node

## Returns

the node's output

Implements [Node](#).

#### 5.4.3.2 `void InputNode::setVal ( double value )`

Set value of the input node

## Parameters

<i>value</i>	the value for the <a href="#">InputNode</a> to output
--------------	---

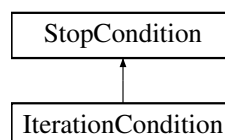
The documentation for this class was generated from the following files:

- `src/netbuilding/InputNode.h`
- `src/netbuilding/InputNode.cpp`

## 5.5 IterationCondition Class Reference

```
#include <IterationCondition.h>
```

Inheritance diagram for IterationCondition:



### Public Member Functions

- [IterationCondition](#) (int allowedIterations)
- bool [check](#) (int iteration, double error)

#### 5.5.1 Detailed Description

Condition for stopping Neural Network training after a fixed amount of training iterations

## 5.5.2 Constructor & Destructor Documentation

### 5.5.2.1 IterationCondition::IterationCondition ( int *allowedIterations* )

Create a new [IterationCondition](#) that will allow given amount of iterations

## Parameters

<i>allowedIterations</i>	amount of iterations to allow
--------------------------	-------------------------------

### 5.5.3 Member Function Documentation

#### 5.5.3.1 bool IterationCondition::check ( int *iteration*, double *error* ) [virtual]

Control if the last iteration has been met

Inherits:

Control if the condition has been met

## Parameters

<i>iteration</i>	amount of training iterations ran
<i>error</i>	current error from evaluation

## Returns

if training is done

Implements [StopCondition](#).

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

- src/stopconditions/IterationCondition.h
- src/stopconditions/IterationCondition.cpp

## 5.6 NetTraining Class Reference

```
#include <NetTraining.h>
```

## Public Member Functions

- [NetTraining](#) ([ConnectedNet](#) \*net, std::vector< [TrainingData](#) > trainingData, std::vector< [TrainingData](#) > evalData, [StopCondition](#) \*stopper, double learningRate, int trainingPerIteration=1)
- void [run](#) ()

### 5.6.1 Detailed Description

Training session of a neural net. Consists of a collection of data, net and some stop condition for how long to keep training.

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 NetTraining::NetTraining ( [ConnectedNet](#) \* *net*, std::vector< [TrainingData](#) > *trainingData*, std::vector< [TrainingData](#) > *evalData*, [StopCondition](#) \* *stopper*, double *learningRate*, int *trainingPerIteration* = 1 )

Create a new [NetTraining](#) session for the given net with the given data and stop condition

## Parameters

<i>net</i>	the neural net to train
<i>trainingData</i>	vector of all data pieces to use for training
<i>evalData</i>	data pieces to use for evaluating performance of the net
<i>stopper</i>	condition for stopping training
<i>learningRate</i>	rate to adjust weights in backpropagation algorithm
<i>trainingPer-Iteration</i>	times training data should be iterated through before performance is evaluated

## 5.6.3 Member Function Documentation

## 5.6.3.1 void NetTraining::run ( )

Start the training and run until the stop condition is met

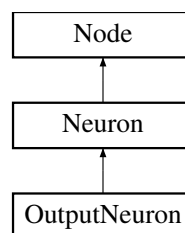
The documentation for this class was generated from the following files:

- src/NetTraining.h
- src/NetTraining.cpp

## 5.7 Neuron Class Reference

```
#include <Neuron.h>
```

Inheritance diagram for Neuron:



## Public Member Functions

- [Neuron](#) (std::function< double(double)> activationFunc, std::vector< [Edge](#) \* > inputs, std::vector< [Edge](#) \* > outputs)
- [Neuron](#) (std::function< double(double)> activationFunc)
- void [setInputs](#) (std::vector< [Edge](#) \* > inputs)
- void [setOutputs](#) (std::vector< [Edge](#) \* > outputs)
- void [calcOutput](#) ()
- void [calcDelta](#) ()
- double [getOutput](#) ()
- double [getDelta](#) ()

## Protected Member Functions

- double [sigmoidDelta](#) ()

## Protected Attributes

- double [delta](#) = 0

### 5.7.1 Detailed Description

A neuron, part of a neural network.

### 5.7.2 Constructor & Destructor Documentation

**5.7.2.1** `Neuron::Neuron ( std::function< double(double)> activationFunc, std::vector< Edge * > inputs, std::vector< Edge * > outputs )`

Create a new [Neuron](#) with the given in- and outputs

Parameters

<i>activationFunc</i>	function to use for neuron activation
<i>inputs</i>	vector of pointers to incoming edges
<i>outputs</i>	vector of pointers to outgoing edges

**5.7.2.2** `Neuron::Neuron ( std::function< double(double)> activationFunc )`

Create a new [Neuron](#)

Parameters

<i>activationFunc</i>	function to use for neuron activation
-----------------------	---------------------------------------

### 5.7.3 Member Function Documentation

**5.7.3.1** `void Neuron::calcDelta ( )`

Calculate and store the delta from this [Neuron](#) according to the GDR. All output Neurons should have their deltas updated before calculating this.

**5.7.3.2** `void Neuron::calcOutput ( )`

Calculate and store the output from this [Neuron](#). All input Neurons should have their outputs updated before calculating this.

**5.7.3.3** `double Neuron::getDelta ( )`

Get delta for backpropagation

Returns

the delta of the [Neuron](#)

**5.7.3.4** `double Neuron::getOutput ( )` `[virtual]`

Get the output of this [Neuron](#)

Inherits:

Get the output of the node



**Returns**

the node's output

Implements [Node](#).

**5.7.3.5** `void Neuron::setInputs ( std::vector< Edge * > inputs )`

Set the edges in to this [Neuron](#)

**Parameters**

<i>inputs</i>	vector of pointers to incoming edges
---------------	--------------------------------------

**5.7.3.6** `void Neuron::setOutputs ( std::vector< Edge * > outputs )`

Set the edges out from this [Neuron](#)

**Parameters**

<i>outputs</i>	vector of pointers to outgoing edges
----------------	--------------------------------------

**5.7.3.7** `double Neuron::sigmoidDelta ( )` [protected]

Get the derivative of a sigmoid activation function for this node

**Returns**

the derivative of the sigmoid

**5.7.4 Member Data Documentation**

**5.7.4.1** `double Neuron::delta = 0` [protected]

Delta for backpropagation

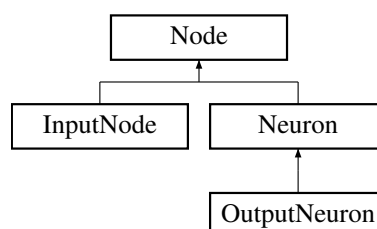
The documentation for this class was generated from the following files:

- src/netbuilding/Neuron.h
- src/netbuilding/Neuron.cpp

**5.8 Node Class Reference**

```
#include <Node.h>
```

Inheritance diagram for Node:



## Public Member Functions

- virtual double [getOutput](#) ()=0

### 5.8.1 Detailed Description

[Node](#) of network

### 5.8.2 Member Function Documentation

#### 5.8.2.1 virtual double [Node::getOutput](#) ( ) [pure virtual]

Get the output of the node

#### Returns

the node's output

Implemented in [Neuron](#), and [InputNode](#).

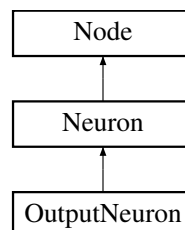
The documentation for this class was generated from the following file:

- `src/netbuilding/Node.h`

## 5.9 OutputNeuron Class Reference

```
#include <OutputNeuron.h>
```

Inheritance diagram for OutputNeuron:



## Public Member Functions

- [OutputNeuron](#) (std::function< double(double)> activationFunc)
- [OutputNeuron](#) (std::function< double(double)> activationFunc, std::vector< [Edge](#) \* > inputs)
- [OutputNeuron](#) (std::function< double(double)> activationFunc, double targetValue)
- void [calcDelta](#) ()
- void [setTarget](#) (double targetValue)
- double [getDifference](#) ()

## Additional Inherited Members

### 5.9.1 Detailed Description

[Neuron](#) at end of Neural Network outputting the nets output

## 5.9.2 Constructor & Destructor Documentation

### 5.9.2.1 OutputNeuron::OutputNeuron ( std::function< double(double)> *activationFunc* )

Create a new [OutputNeuron](#) using the given activation function

Parameters

<i>activationFunc</i>	function to use for neuron activation
-----------------------	---------------------------------------

### 5.9.2.2 OutputNeuron::OutputNeuron ( std::function< double(double)> *activationFunc*, std::vector< [Edge](#) \* > *inputs* )

Create a new [OutputNeuron](#) using the given activation function with given inputs

Parameters

<i>activationFunc</i>	function to use for neuron activation
<i>inputs</i>	vector of pointers to incoming edges

### 5.9.2.3 OutputNeuron::OutputNeuron ( std::function< double(double)> *activationFunc*, double *targetValue* )

Create a new [OutputNeuron](#) using the given activation function and target value

Parameters

<i>activationFunc</i>	function to use for neuron activation
<i>targetValue</i>	target output value of the neuron

## 5.9.3 Member Function Documentation

### 5.9.3.1 void OutputNeuron::calcDelta ( )

Calculate delta for output [Neuron](#) according to the GDR

Inherits:

Calculate and store the delta from this [Neuron](#) according to the GDR. All output Neurons should have their deltas updated before calculating this.

### 5.9.3.2 double OutputNeuron::getDifference ( )

Get the difference between actual output and target

Returns

the difference for the Neurons current values

### 5.9.3.3 void OutputNeuron::setTarget ( double *targetValue* )

Set the target value of the [OutputNeuron](#)

Parameters

<i>targetValue</i>	value to target as output
--------------------	---------------------------

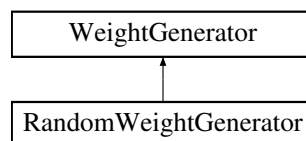
The documentation for this class was generated from the following files:

- `src/netbuilding/OutputNeuron.h`
- `src/netbuilding/OutputNeuron.cpp`

## 5.10 RandomWeightGenerator Class Reference

```
#include <RandomWeightGenerator.h>
```

Inheritance diagram for RandomWeightGenerator:



### Public Member Functions

- [RandomWeightGenerator](#) (double *weightMin*=RandomWeightGenerator::DEFAULT\_MIN, double *weightMax*=RandomWeightGenerator::DEFAULT\_MAX)
- double [getWeight](#) ()

### 5.10.1 Detailed Description

Weight Generator for generating random weights

### 5.10.2 Constructor & Destructor Documentation

**5.10.2.1** `RandomWeightGenerator::RandomWeightGenerator ( double weightMin = RandomWeightGenerator::DEFAULT_MIN, double weightMax = RandomWeightGenerator::DEFAULT_MAX )`

Create new [RandomWeightGenerator](#) for generating weights in the interval [*weightMin*, *weightMax*)

Parameters

<i>weightMin</i>	bottom limit of random interval
<i>weightMax</i>	upper limit of random interval

### 5.10.3 Member Function Documentation

**5.10.3.1** `double RandomWeightGenerator::getWeight ( ) [virtual]`

Generate a new random weight

Inherits:

Get a new weight from the generator

**Returns**

the new weight

Implements [WeightGenerator](#).

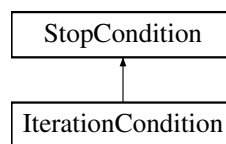
The documentation for this class was generated from the following files:

- src/netbuilding/weightgenerators/RandomWeightGenerator.h
- src/netbuilding/weightgenerators/RandomWeightGenerator.cpp

## 5.11 StopCondition Class Reference

```
#include <StopCondition.h>
```

Inheritance diagram for StopCondition:

**Public Member Functions**

- virtual bool [check](#) (int iteration, double error)=0

### 5.11.1 Detailed Description

Interface for classes describing conditions for whether to continue training Neural Nets

### 5.11.2 Member Function Documentation

5.11.2.1 virtual bool StopCondition::check ( int *iteration*, double *error* ) [pure virtual]

Control if the condition has been met

**Parameters**

<i>iteration</i>	amount of training iterations ran
<i>error</i>	current error from evaluation

**Returns**

if training is done

Implemented in [IterationCondition](#).

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

- src/stopconditions/StopCondition.h

## 5.12 TrainingData Class Reference

```
#include <TrainingData.h>
```

## Public Member Functions

- [TrainingData](#) (std::vector< double > input, std::vector< double > target)
- std::vector< double > [getInput](#) ()
- std::vector< double > [getTarget](#) ()

### 5.12.1 Detailed Description

Data to be used for training a neural net. Consist of a pair of input data and wanted output

### 5.12.2 Constructor & Destructor Documentation

#### 5.12.2.1 TrainingData::TrainingData ( std::vector< double > *input*, std::vector< double > *target* )

Create a new training data input-target pair

Parameters

<i>input</i>	the input to the neural net
<i>target</i>	the wanted output from the net

### 5.12.3 Member Function Documentation

#### 5.12.3.1 std::vector< double > TrainingData::getInput ( )

Get the input vector of the training data

Returns

the input vector

#### 5.12.3.2 std::vector< double > TrainingData::getTarget ( )

Get the known target vector of the training data

Returns

the (target) output vector

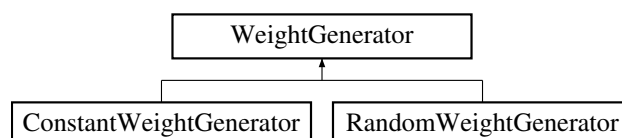
The documentation for this class was generated from the following files:

- src/TrainingData.h
- src/TrainingData.cpp

## 5.13 WeightGenerator Class Reference

```
#include <WeightGenerator.h>
```

Inheritance diagram for WeightGenerator:



## Public Member Functions

- virtual double [getWeight](#) ()=0

### 5.13.1 Detailed Description

Generator of default weights for edges

### 5.13.2 Member Function Documentation

#### 5.13.2.1 virtual double WeightGenerator::getWeight ( ) [pure virtual]

Get a new weight from the generator

#### Returns

the new weight

Implemented in [RandomWeightGenerator](#), and [ConstantWeightGenerator](#).

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

- src/netbuilding/weightgenerators/WeightGenerator.h

# Index

- ~ConnectedNet
  - ConnectedNet, 10
- calcDelta
  - Neuron, 18
  - OutputNeuron, 21
- calcOutput
  - Neuron, 18
- check
  - IterationCondition, 16
  - StopCondition, 23
- ConnectedNet, 9
  - ~ConnectedNet, 10
  - ConnectedNet, 9
  - ConnectedNet, 9
  - getDifference, 10
  - getOutput, 10
  - train, 10
- ConstantWeightGenerator, 10
  - ConstantWeightGenerator, 11
  - ConstantWeightGenerator, 11
  - getWeight, 11
- delta
  - Neuron, 19
- Edge, 11
  - Edge, 12
  - getWeightedDelta, 12
  - getWeightedOutput, 12
  - setInput, 12
  - setOutput, 12
  - updateWeight, 13
- getDelta
  - Neuron, 18
- getDifference
  - ConnectedNet, 10
  - OutputNeuron, 21
- getInput
  - TrainingData, 24
- getOutput
  - ConnectedNet, 10
  - InputNode, 14
  - Neuron, 18
  - Node, 20
- getTarget
  - TrainingData, 24
- getWeight
  - ConstantWeightGenerator, 11
  - RandomWeightGenerator, 22
  - WeightGenerator, 25
- getWeightedDelta
  - Edge, 12
- getWeightedOutput
  - Edge, 12
- InputNode, 13
  - getOutput, 14
  - InputNode, 13
  - InputNode, 13
  - setValue, 14
- IterationCondition, 14
  - check, 16
  - IterationCondition, 15
  - IterationCondition, 15
- NNetUtil, 7
  - sigmoid, 7
  - squareError, 7
  - unitStep, 8
- NetTraining, 16
  - NetTraining, 16
  - NetTraining, 16
  - run, 17
- Neuron, 17
  - calcDelta, 18
  - calcOutput, 18
  - delta, 19
  - getDelta, 18
  - getOutput, 18
  - Neuron, 18
  - setInputs, 19
  - setOutputs, 19
  - sigmoidDelta, 19
- Node, 19
  - getOutput, 20
- OutputNeuron, 20
  - calcDelta, 21
  - getDifference, 21
  - OutputNeuron, 21
  - OutputNeuron, 21
  - setTarget, 21
- RandomWeightGenerator, 22
  - getWeight, 22
  - RandomWeightGenerator, 22
  - RandomWeightGenerator, 22
- run



- NetTraining, [17](#)
- setInput
  - Edge, [12](#)
- setInputs
  - Neuron, [19](#)
- setOutput
  - Edge, [12](#)
- setOutputs
  - Neuron, [19](#)
- setTarget
  - OutputNeuron, [21](#)
- setValue
  - InputNode, [14](#)
- sigmoid
  - NNetUtil, [7](#)
- sigmoidDelta
  - Neuron, [19](#)
- squareError
  - NNetUtil, [7](#)
- StopCondition, [23](#)
  - check, [23](#)
- train
  - ConnectedNet, [10](#)
- TrainingData, [23](#)
  - getInput, [24](#)
  - getTarget, [24](#)
  - TrainingData, [24](#)
  - TrainingData, [24](#)
- unitStep
  - NNetUtil, [8](#)
- updateWeight
  - Edge, [13](#)
- WeightGenerator, [24](#)
  - getWeight, [25](#)