

## Feedforward Closedloop Learning

Generated by Doxygen 1.8.17



<b>1 Feedforward Closedloop Learning (FCL)</b>	<b>1</b>
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Bandpass Class Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Function Documentation	8
4.1.2.1 getOutput()	8
4.2 FeedforwardClosedloopLearning Class Reference	8
4.2.1 Detailed Description	9
4.2.2 Constructor & Destructor Documentation	10
4.2.2.1 FeedforwardClosedloopLearning()	10
4.2.2.2 ~FeedforwardClosedloopLearning()	10
4.2.3 Member Function Documentation	10
4.2.3.1 doStep() [1/2]	10
4.2.3.2 doStep() [2/2]	11
4.2.3.3 getLayer()	11
4.2.3.4 getLayers()	11
4.2.3.5 getNumInputs()	11
4.2.3.6 getNumLayers()	12
4.2.3.7 getOutput()	12
4.2.3.8 getOutputLayer()	12
4.2.3.9 initWeights()	12
4.2.3.10 loadModel()	13
4.2.3.11 saveModel()	13
4.2.3.12 seedRandom()	13
4.2.3.13 setActivationFunction()	14
4.2.3.14 setBias()	14
4.2.3.15 setDecay()	14
4.2.3.16 setLearningRate()	14
4.2.3.17 setLearningRateDiscountFactor()	15
4.2.3.18 setMomentum()	15
4.3 FeedforwardClosedloopLearningWithFilterbank Class Reference	15
4.3.1 Detailed Description	16
4.3.2 Constructor & Destructor Documentation	17
4.3.2.1 FeedforwardClosedloopLearningWithFilterbank()	17
4.3.3 Member Function Documentation	17
4.3.3.1 doStep() [1/2]	17

4.3.3.2 doStep() [2/2]	18
4.4 Layer Class Reference	18
4.4.1 Detailed Description	19
4.4.2 Constructor & Destructor Documentation	20
4.4.2.1 Layer()	20
4.4.2.2 ~Layer()	20
4.4.3 Member Function Documentation	20
4.4.3.1 calcOutputs()	20
4.4.3.2 doLearning()	20
4.4.3.3 doNormaliseWeights()	21
4.4.3.4 getError()	21
4.4.3.5 getNeuron()	21
4.4.3.6 getNinputs()	21
4.4.3.7 getNneurons()	22
4.4.3.8 getOutput()	22
4.4.3.9 getWeightDistanceFromInitialWeights()	22
4.4.3.10 initWeights()	22
4.4.3.11 saveWeightMatrix()	23
4.4.3.12 setActivationFunction()	23
4.4.3.13 setBias()	23
4.4.3.14 setConvolution()	24
4.4.3.15 setDebugInfo()	24
4.4.3.16 setDecay()	24
4.4.3.17 setError() [1/2]	24
4.4.3.18 setError() [2/2]	25
4.4.3.19 setErrors()	25
4.4.3.20 setInput()	25
4.4.3.21 setInputs()	26
4.4.3.22 setLearningRate()	26
4.4.3.23 setMaxDetLayer()	26
4.4.3.24 setMomentum()	26
4.4.3.25 setNormaliseWeights()	27
4.4.3.26 setStep()	27
4.4.3.27 setUseThreads()	27
4.5 Neuron Class Reference	27
4.5.1 Detailed Description	30
4.5.2 Member Enumeration Documentation	30
4.5.2.1 WeightInitMethod	30
4.5.3 Constructor & Destructor Documentation	30
4.5.3.1 Neuron()	30
4.5.3.2 ~Neuron()	30
4.5.4 Member Function Documentation	31

4.5.4.1 calcOutputThread()	31
4.5.4.2 dActivation()	31
4.5.4.3 doLearning()	31
4.5.4.4 doLearningThread()	31
4.5.4.5 doMaxDet()	31
4.5.4.6 doMaxDetThread()	32
4.5.4.7 getAverageOfWeightVector()	32
4.5.4.8 getBiasWeight()	32
4.5.4.9 getDecay()	32
4.5.4.10 getError()	33
4.5.4.11 getEuclideanNormOfWeightVector()	33
4.5.4.12 getInfinityNormOfWeightVector()	33
4.5.4.13 getInput()	33
4.5.4.14 getMask() [1/2]	34
4.5.4.15 getMask() [2/2]	34
4.5.4.16 getMaxWeightValue()	34
4.5.4.17 getMinWeightValue()	35
4.5.4.18 getNinputs()	35
4.5.4.19 getOutput()	35
4.5.4.20 getSum()	35
4.5.4.21 getSumOfSquaredWeightVector()	36
4.5.4.22 getWeight()	36
4.5.4.23 getWeightDistanceFromInitialWeights()	36
4.5.4.24 initWeights()	36
4.5.4.25 normaliseWeights()	37
4.5.4.26 saveInitialWeights()	37
4.5.4.27 setActivationFunction()	37
4.5.4.28 setBias()	37
4.5.4.29 setBiasWeight()	38
4.5.4.30 setDebugInfo()	38
4.5.4.31 setDecay()	38
4.5.4.32 setError()	39
4.5.4.33 setGeometry()	39
4.5.4.34 setInput()	39
4.5.4.35 setLearningRate()	40
4.5.4.36 setMask() [1/2]	40
4.5.4.37 setMask() [2/2]	40
4.5.4.38 setMomentum()	40
4.5.4.39 setStep()	41
4.5.4.40 setWeight()	41



## Chapter 1

# Feedforward Closedloop Learning (FCL)

Forward propagation closed loop learning Bernd Porr, Paul Miller. Adaptive Behaviour 2019.

[Submission version](#)

For an autonomous agent, the inputs are the sensory data that inform the agent of the state of the world, and the outputs are their actions, which act on the world and consequently produce new sensory inputs. The agent only knows of its own actions via their effect on future inputs; therefore desired states, and error signals, are most naturally defined in terms of the inputs. Most machine learning algorithms, however, operate in terms of desired outputs. For example, backpropagation takes target output values and propagates the corresponding error backwards through the network in order to change the weights. In closed loop settings, it is far more obvious how to define desired sensory inputs than desired actions, however. To train a deep network using errors defined in the input space would call for an algorithm that can propagate those errors forwards through the network, from input layer to output layer, in much the same way that activations are propagated.

[Github project page](#)





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

Bandpass . . . . .	7
FeedforwardClosedloopLearning . . . . .	8
FeedforwardClosedloopLearningWithFilterbank . . . . .	15
Layer . . . . .	18
Neuron . . . . .	27



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Bandpass</a>	Creates memory traces at specified length . . . . .	7
<a href="#">FeedforwardClosedloopLearning</a>	Main class of Feedforward Closed Loop Learning . . . . .	8
<a href="#">FeedforwardClosedloopLearningWithFilterbank</a>	Derived classes of the <a href="#">FeedforwardClosedloopLearning</a> class for special functionality . . . . .	15
<a href="#">Layer</a>	<a href="#">Layer</a> which contains the neurons of one layer . . . . .	18
<a href="#">Neuron</a>	<a href="#">Neuron</a> which calculates the output and performs learning . . . . .	27



## Chapter 4

# Class Documentation

### 4.1 Bandpass Class Reference

Creates memory traces at specified length.

```
#include <bandpass.h>
```

#### Public Member Functions

- [Bandpass](#) ()  
*Constructor.*
- double [filter](#) (double v)  
*Filter.*
- void [calcPolesZeros](#) (double f, double r)  
*Calculates the coefficients The frequency is the normalized frequency in the range [0..0.5].*
- void [setParameters](#) (double frequency, double Qfactor)  
*sets the filter parameters*
- void [impulse](#) (char \*name)  
*Generates an acsii file with the impulse response of the filter.*
- void [calcNorm](#) (double f)  
*Normalises the output with f.*
- void [transfer](#) (char \*name)  
*Generates an ASCII file with the transfer function.*
- double [getOutput](#) ()  
*Gets the output of the filter.*
- void [reset](#) ()  
*Sets the output to zero again.*

#### 4.1.1 Detailed Description

Creates memory traces at specified length.

It's a 2nd order IIR filter.

## 4.1.2 Member Function Documentation

### 4.1.2.1 getOutput()

```
double Bandpass::getOutput ( ) [inline]
```

Gets the output of the filter.

Same as the return value of the function "filter()".

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

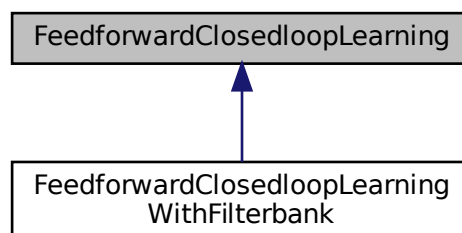
- bandpass.h
- bandpass.cpp

## 4.2 FeedforwardClosedloopLearning Class Reference

Main class of Feedforward Closed Loop Learning.

```
#include <fcl.h>
```

Inheritance diagram for FeedforwardClosedloopLearning:



## Public Member Functions

- [FeedforwardClosedloopLearning](#) (int num\_of\_inputs, int \*num\_of\_neurons\_per\_layer\_array, int \_num\_layers)  
*Constructor: FCL without any filters.*
- [~FeedforwardClosedloopLearning](#) ()  
*Destructor De-allocated any memory.*
- void [doStep](#) (double \*input, double \*error)  
*Performs the simulation step.*
- void [doStep](#) (double \*input, int n1, double \*error, int n2)  
*Python wrapper function.*
- double [getOutput](#) (int index)  
*Gets the output from one of the output neurons.*
- void [setLearningRate](#) (double learningRate)  
*Sets globally the learning rate.*
- void [setLearningRateDiscountFactor](#) (double \_learningRateDiscountFactor)  
*Sets how the learnign rate increases or decreases from layer to layer.*
- void [setDecay](#) (double decay)  
*Sets a typical weight decay scaled with the learning rate.*
- void [setMomentum](#) (double momentum)  
*Sets the global momentum for all layers.*
- void [setActivationFunction](#) ([Neuron::ActivationFunction](#) \_activationFunction)  
*Sets the activation function of the [Neuron](#).*
- void [initWeights](#) (double max=0.001, int initBias=1, [Neuron::WeightInitMethod](#) weightInitMethod=[Neuron::MAX\\_OUTPUT\\_RANDOM](#))  
*Init's the weights in all layers.*
- void [seedRandom](#) (int s)  
*Seeds the random number generator.*
- void [setBias](#) (double \_bias)  
*Sets globally the bias.*
- int [getNumLayers](#) ()  
*Gets the total number of layers.*
- [Layer](#) \* [getLayer](#) (int i)  
*Gets a pointer to a layer.*
- [Layer](#) \* [getOutputLayer](#) ()  
*Gets the output layer.*
- int [getNumInputs](#) ()  
*Gets the number of inputs.*
- [Layer](#) \*\* [getLayers](#) ()  
*Returns all Layers.*
- bool [saveModel](#) (const char \*name)  
*Saves the whole network.*
- bool [loadModel](#) (const char \*name)  
*Loads the while network.*

### 4.2.1 Detailed Description

Main class of Feedforward Closed Loop Learning.

Create an instance of this class to do the learning. It will create the whole network with an input layer, layers and an output layer. Learning is done iterative by first setting the input values and errors and then calling [doStep\(\)](#).

(C) 2017,2018-2022, Bernd Porr [bernd@glasgowneuro.tech](mailto:bernd@glasgowneuro.tech) (C) 2017,2018, Paul Miller [paul@glasgowneuro.tech](mailto:paul@glasgowneuro.tech)

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

## 4.2.2 Constructor & Destructor Documentation

### 4.2.2.1 FeedforwardClosedloopLearning()

```
FeedforwardClosedloopLearning::FeedforwardClosedloopLearning (
    int num_of_inputs,
    int * num_of_neurons_per_layer_array,
    int _num_layers )
```

Constructor: FCL without any filters.

#### Parameters

<i>num_of_inputs</i>	Number of inputs in the input layer
<i>num_of_neurons_per_layer_array</i>	Number of neurons in each layer
<i>_num_layers</i>	Number of layer (needs to match with array above)

### 4.2.2.2 ~FeedforwardClosedloopLearning()

```
FeedforwardClosedloopLearning::~~FeedforwardClosedloopLearning ( )
```

Destructor De-allocated any memory.

## 4.2.3 Member Function Documentation

### 4.2.3.1 doStep() [1/2]

```
void FeedforwardClosedloopLearning::doStep (
    double * input,
    double * error )
```

Performs the simulation step.

#### Parameters

<i>input</i>	Array with the input values
<i>error</i>	Array of the error signals



#### 4.2.3.2 doStep() [2/2]

```
void FeedforwardClosedloopLearning::doStep (
    double * input,
    int n1,
    double * error,
    int n2 )
```

Python wrapper function.

Not public.

#### 4.2.3.3 getLayer()

```
Layer* FeedforwardClosedloopLearning::getLayer (
    int i ) [inline]
```

Gets a pointer to a layer.

##### Parameters

<i>i</i>	Index of the layer.
----------	---------------------

##### Returns

A pointer to a layer class.

#### 4.2.3.4 getLayers()

```
Layer** FeedforwardClosedloopLearning::getLayers ( ) [inline]
```

Returns all Layers.

##### Returns

Returns a two dimensional array of all layers.

#### 4.2.3.5 getNumInputs()

```
int FeedforwardClosedloopLearning::getNumInputs ( ) [inline]
```

Gets the number of inputs.

##### Returns

The number of inputs

#### 4.2.3.6 getNumLayers()

```
int FeedforwardClosedloopLearning::getNumLayers ( ) [inline]
```

Gets the total number of layers.

##### Returns

The total number of all layers.

#### 4.2.3.7 getOutput()

```
double FeedforwardClosedloopLearning::getOutput (
    int index ) [inline]
```

Gets the output from one of the output neurons.

##### Parameters

<i>index</i>	The index number of the output neuron.
--------------	--

##### Returns

The output value of the output neuron.

#### 4.2.3.8 getOutputLayer()

```
Layer* FeedforwardClosedloopLearning::getOutputLayer ( ) [inline]
```

Gets the output layer.

##### Returns

A pointer to the output layer which is also a [Layer](#) class.

#### 4.2.3.9 initWeights()

```
void FeedforwardClosedloopLearning::initWeights (
    double max = 0.001,
    int initBias = 1,
    Neuron::WeightInitMethod weightInitMethod = Neuron::MAX_OUTPUT_RANDOM )
```

Initiates the weights in all layers.

## Parameters

<i>max</i>	Maximum value of the weights.
<i>initBias</i>	If the bias also should be initialised.
<i>weightInitMethod</i>	See <a href="#">Neuron::WeightInitMethod</a> for the options.

**4.2.3.10 loadModel()**

```
bool FeedforwardClosedloopLearning::loadModel (
    const char * name )
```

Loads the while network.

## Parameters

<i>name</i>	filename
-------------	----------

**4.2.3.11 saveModel()**

```
bool FeedforwardClosedloopLearning::saveModel (
    const char * name )
```

Saves the whole network.

## Parameters

<i>name</i>	filename
-------------	----------

**4.2.3.12 seedRandom()**

```
void FeedforwardClosedloopLearning::seedRandom (
    int s ) [inline]
```

Seeds the random number generator.

## Parameters

<i>s</i>	An arbitratry number.
----------	-----------------------

#### 4.2.3.13 setActivationFunction()

```
void FeedforwardClosedloopLearning::setActivationFunction (
    Neuron::ActivationFunction _activationFunction )
```

Sets the activation function of the [Neuron](#).

##### Parameters

<code>_activationFunction</code>	See <a href="#">Neuron::ActivationFunction</a> for the different options.
----------------------------------	---

#### 4.2.3.14 setBias()

```
void FeedforwardClosedloopLearning::setBias (
    double _bias )
```

Sets globally the bias.

##### Parameters

<code>_bias</code>	Sets globally the bias input to all neurons.
--------------------	--

#### 4.2.3.15 setDecay()

```
void FeedforwardClosedloopLearning::setDecay (
    double decay )
```

Sets a typical weight decay scaled with the learning rate.

##### Parameters

<code>decay</code>	The larger the faster the decay.
--------------------	----------------------------------

#### 4.2.3.16 setLearningRate()

```
void FeedforwardClosedloopLearning::setLearningRate (
    double learningRate )
```

Sets globally the learning rate.

## Parameters

<i>learningRate</i>	Sets the learning rate for all layers and neurons.
---------------------	--

**4.2.3.17 setLearningRateDiscountFactor()**

```
void FeedforwardClosedloopLearning::setLearningRateDiscountFactor (
    double _learningRateDiscountFactor ) [inline]
```

Sets how the learnign rate increases or decreases from layer to layer.

## Parameters

<i>_learningRateDiscountFactor</i>	A factor of $>1$ means higher learning rate in deeper layers.
------------------------------------	---

**4.2.3.18 setMomentum()**

```
void FeedforwardClosedloopLearning::setMomentum (
    double momentum )
```

Sets the global momentum for all layers.

## Parameters

<i>momentum</i>	Defines the inertia of the weight change over time.
-----------------	---

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

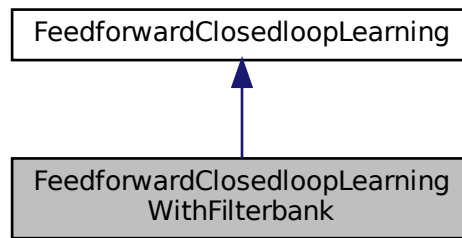
- fcl.h

**4.3 FeedforwardClosedloopLearningWithFilterbank Class Reference**

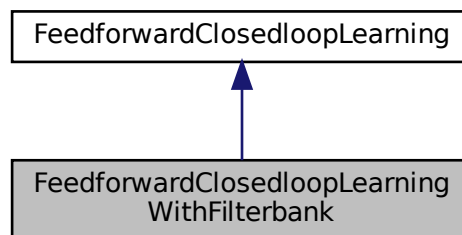
Derived classes of the [FeedforwardClosedloopLearning](#) class for special functionality.

```
#include <fcl_util.h>
```

Inheritance diagram for FeedforwardClosedloopLearningWithFilterbank:



Collaboration diagram for FeedforwardClosedloopLearningWithFilterbank:



## Public Member Functions

- [FeedforwardClosedloopLearningWithFilterbank](#) (int num\_of\_inputs, int \*num\_of\_neurons\_per\_layer\_array, int num\_layers, int num\_filtersInput, double minT, double maxT)  
*FeedforwardClosedloopLearning with Filterbank at each input.*
- [~FeedforwardClosedloopLearningWithFilterbank](#) ()  
*Destructor.*
- void [doStep](#) (double \*input, double \*error)  
*Performs the simulation step.*
- void [doStep](#) (double \*input, int n1, double \*error, int n2)  
*Python wrapper function.*
- double [getFilterOutput](#) (int inputIdx, int filterIdx)
- int [getNFiltersPerInput](#) ()

### 4.3.1 Detailed Description

Derived classes of the [FeedforwardClosedloopLearning](#) class for special functionality.

## 4.3.2 Constructor & Destructor Documentation

### 4.3.2.1 FeedforwardClosedloopLearningWithFilterbank()

```
FeedforwardClosedloopLearningWithFilterbank::FeedforwardClosedloopLearningWithFilterbank (
    int num_of_inputs,
    int * num_of_neurons_per_layer_array,
    int num_layers,
    int num_filtersInput,
    double minT,
    double maxT )
```

[FeedforwardClosedloopLearning](#) with Filterbank at each input.

Constructor: FCL with a filter bank at the input Every input feeds internally into a has a filter bank of num\_filtersInput filters. This allows for a temporal distribution of the inputs.

#### Parameters

<i>num_of_inputs</i>	Number of inputs in the input layer
<i>num_of_neurons_per_layer_array</i>	Number of neurons in each layer
<i>_num_layers</i>	Number of layer (needs to match with array above)
<i>num_filtersInput</i>	Number of filters at the input layer, 0 = no filterbank
<i>num_filters</i>	Number of filters in the hiddel layers (usually zero)
<i>_minT</i>	Minimum/first temporal duration of the 1st filter
<i>_maxT</i>	Maximum/last temporal duration of the last filter

## 4.3.3 Member Function Documentation

### 4.3.3.1 doStep() [1/2]

```
void FeedforwardClosedloopLearningWithFilterbank::doStep (
    double * input,
    double * error )
```

Performs the simulation step.

#### Parameters

<i>input</i>	Array with the input values
<i>error</i>	Array of the error signals

#### 4.3.3.2 doStep() [2/2]

```
void FeedforwardClosedloopLearningWithFilterbank::doStep (
    double * input,
    int n1,
    double * error,
    int n2 )
```

Python wrapper function.

Not public.

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

- fcl\_util.h

## 4.4 Layer Class Reference

[Layer](#) which contains the neurons of one layer.

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

### Public Types

- enum [WeightNormalisation](#) {  
**WEIGHT\_NORM\_NONE** = 0, **WEIGHT\_NORM\_LAYER\_EUCLEDIAN** = 1, **WEIGHT\_NORM\_NEURON\_**↵  
**EUCLEDIAN** = 2, **WEIGHT\_NORM\_LAYER\_MANHATTAN** = 3,  
**WEIGHT\_NORM\_NEURON\_MANHATTAN** = 4, **WEIGHT\_NORM\_LAYER\_INFINITY** = 5, **WEIGHT\_NO**↵  
**RM\_NEURON\_INFINITY** = 6 }

*Weight normalisation constants Defines if weights are normalised layer-wide or for every neuron separately.*

### Public Member Functions

- [Layer](#) (int \_nNeurons, int \_nInputs)  
*Constructor.*
- [~Layer](#) ()  
*Destructor Frees all memory.*
- void [calcOutputs](#) ()  
*Calculates the output values in all neurons.*
- void [doLearning](#) ()  
*Adjusts the weights.*
- void [setError](#) (double \_error)  
*Sets the global error for all neurons.*
- void [setError](#) (int i, double \_error)  
*sets the error individually*
- void [setErrors](#) (double \*\_errors)  
*Sets all errors from an input array.*
- double [getError](#) (int i)  
*Retrieves the error.*



- void [setBias](#) (double \_bias)  
*Sets the global bias for all neurons.*
- void [setInput](#) (int inputIndex, double input)  
*Set the input value of one input.*
- void [setInputs](#) (double \*\_inputs)  
*Sets all inputs from an input array.*
- void [setLearningRate](#) (double \_learningRate)  
*Sets the learning rate of all neurons.*
- void [setActivationFunction](#) ([Neuron::ActivationFunction](#) \_activationFunction)  
*Set the activation function.*
- void [setMomentum](#) (double \_momentum)  
*Set the momentum of all neurons in this layer.*
- void [setDecay](#) (double \_decay)  
*Sets the weight decay scaled by the learning rate.*
- void [initWeights](#) (double \_max=1, int initBiasWeight=1, [Neuron::WeightInitMethod](#) weightInitMethod=[Neuron::MAX\\_OUTPUT\\_RANDOM](#))  
*Init the weights.*
- double [getOutput](#) (int index)  
*Gets the output of one neuron.*
- [Neuron \\*](#) [getNeuron](#) (int index)  
*Gets a pointer to one neuron.*
- int [getNneurons](#) ()  
*Gets the number of neurons.*
- int [getNinputs](#) ()  
*Number of inputs.*
- void [setConvolution](#) (int width, int height)  
*Defines a 2D geometry for the input layer of widthxheight.*
- void [setMaxDetLayer](#) (int \_m)  
*Maxium detection layer.*
- void [setNormaliseWeights](#) ([WeightNormalisation](#) \_normaliseWeights)  
*Normalise the weights.*
- void [setDebugInfo](#) (int layerIndex)  
*Sets the layer index within the whole network.*
- void [setStep](#) (long int step)  
*Sets the simulation step in the layer for debug purposes.*
- double [getWeightDistanceFromInitialWeights](#) ()  
*Get weight distance from the start of the simulation.*
- void [doNormaliseWeights](#) ()  
*Performs the weight normalisation.*
- void [setUseThreads](#) (int \_useThreads)  
*Sets if threads should be used.*
- int [saveWeightMatrix](#) (char \*filename)  
*Save weight matrix for documentation and debugging.*

#### 4.4.1 Detailed Description

[Layer](#) which contains the neurons of one layer.

It performs all computations possible in a layer. In particular it calls all neurons in separate threads and triggers the computations there. These functions are all called from the parent class.

## 4.4.2 Constructor & Destructor Documentation

### 4.4.2.1 Layer()

```
Layer::Layer (
    int _nNeurons,
    int _nInputs )
```

Constructor.

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007.

#### Parameters

<code>_nNeurons</code>	Number of neurons in the layer.
<code>_nInputs</code>	Number of inputs to the <a href="#">Layer</a> .
<code>_nFilters</code>	Number of lowpass filters at each input.
<code>_minT</code>	Minimum time of the lowpass filter.
<code>_maxT</code>	Maximum time of the lowpass filter.

(C) 2017, Bernd Porr [bernd@glasgowneuro.tech](mailto:bernd@glasgowneuro.tech) (C) 2017, Paul Miller [paul@glasgowneuro.tech](mailto:paul@glasgowneuro.tech)

### 4.4.2.2 ~Layer()

```
Layer::~~Layer ( )
```

Destructor Frees all memory.

## 4.4.3 Member Function Documentation

### 4.4.3.1 calcOutputs()

```
void Layer::calcOutputs ( )
```

Calculates the output values in all neurons.

### 4.4.3.2 doLearning()

```
void Layer::doLearning ( )
```

Adjusts the weights.

#### 4.4.3.3 doNormaliseWeights()

```
void Layer::doNormaliseWeights ( )
```

Performs the weight normalisation.

#### 4.4.3.4 getError()

```
double Layer::getError (
    int i )
```

Retrieves the error.

##### Parameters

<i>i</i>	Index of the neuron
----------	---------------------

#### 4.4.3.5 getNeuron()

```
Neuron* Layer::getNeuron (
    int index ) [inline]
```

Gets a pointer to one neuron.

##### Parameters

<i>index</i>	The index number of the neuron.
--------------	---------------------------------

##### Returns

A pointer to a [Layer](#) class.

#### 4.4.3.6 getNinputs()

```
int Layer::getNinputs ( ) [inline]
```

Number of inputs.

##### Returns

The number of inputs

#### 4.4.3.7 getNneurons()

```
int Layer::getNneurons ( ) [inline]
```

Gets the number of neurons.

##### Returns

The number of neurons.

#### 4.4.3.8 getOutput()

```
double Layer::getOutput (
    int index ) [inline]
```

Gets the output of one neuron.

##### Parameters

<i>index</i>	The index number of the neuron.
--------------	---------------------------------

##### Returns

Returns the double value of the output.

#### 4.4.3.9 getWeightDistanceFromInitialWeights()

```
double Layer::getWeightDistanceFromInitialWeights ( )
```

Get weight distance from the start of the simulation.

##### Returns

The distance from the initial (random) weight setup.

#### 4.4.3.10 initWeights()

```
void Layer::initWeights (
    double _max = 1,
    int initBiasWeight = 1,
    Neuron::WeightInitMethod weightInitMethod = Neuron::MAX_OUTPUT_RANDOM )
```

Initializes the weights.

## Parameters

<i>_max</i>	Maximum value if using random init.
<i>initBiasWeight</i>	if one also the bias weight is initialised.
<i>weightInitMethod</i>	The method employed to init the weights.

**4.4.3.11 saveWeightMatrix()**

```
int Layer::saveWeightMatrix (
    char * filename )
```

Save weight matrix for documentation and debugging.

## Parameters

<i>filename</i>	The filename it should be saved to.
-----------------	-------------------------------------

**4.4.3.12 setActivationFunction()**

```
void Layer::setActivationFunction (
    Neuron::ActivationFunction _activationFunction )
```

Set the activation function.

## Parameters

<i>_activationFunction</i>	The activation function. See: <a href="#">Neuron::ActivationFunction</a>
----------------------------	--

**4.4.3.13 setBias()**

```
void Layer::setBias (
    double _bias )
```

Sets the global bias for all neurons.

## Parameters

<i>_bias</i>	The bias for all neurons
--------------	--------------------------

#### 4.4.3.14 setConvolution()

```
void Layer::setConvolution (
    int width,
    int height )
```

Defines a 2D geometry for the input layer of widthxheight.

##### Parameters

<i>width</i>	The width of the convolutional window.
<i>height</i>	The height of the convolution window.

#### 4.4.3.15 setDebugInfo()

```
void Layer::setDebugInfo (
    int layerIndex )
```

Sets the layer index within the whole network.

##### Parameters

<i>layerIndex</i>	The layer index in the whole network.
-------------------	---------------------------------------

#### 4.4.3.16 setDecay()

```
void Layer::setDecay (
    double _decay )
```

Sets the weight decay scaled by the learning rate.

##### Parameters

<i>_decay</i>	The decay rate of the weights
---------------	-------------------------------

#### 4.4.3.17 setError() [1/2]

```
void Layer::setError (
    double _error )
```

Sets the global error for all neurons.

## Parameters

<code>_error</code>	Sets the error in the whole layer
---------------------	-----------------------------------

**4.4.3.18 setError()** [2/2]

```
void Layer::setError (
    int i,
    double _error )
```

sets the error individually

## Parameters

<i>i</i>	Index of the neuron
<code>_error</code>	The error to be set

**4.4.3.19 setErrors()**

```
void Layer::setErrors (
    double * _errors )
```

Sets all errors from an input array.

## Parameters

<code>_errors</code>	is an array of errors
----------------------	-----------------------

**4.4.3.20 setInput()**

```
void Layer::setInput (
    int inputIndex,
    double input )
```

Set the input value of one input.

## Parameters

<i>inputIndex</i>	The index number of the input.
<i>input</i>	The value of the input

#### 4.4.3.21 setInputs()

```
void Layer::setInputs (
    double * _inputs )
```

Sets all inputs from an input array.

##### Parameters

<code>_inputs</code>	array of all inputs
----------------------	---------------------

#### 4.4.3.22 setLearningRate()

```
void Layer::setLearningRate (
    double _learningRate )
```

Sets the learning rate of all neurons.

##### Parameters

<code>_learningRate</code>	The learning rate
----------------------------	-------------------

#### 4.4.3.23 setMaxDetLayer()

```
void Layer::setMaxDetLayer (
    int _m ) [inline]
```

Maxium detection layer.

Experimental. This hasn't been implemented.

#### 4.4.3.24 setMomentum()

```
void Layer::setMomentum (
    double _momentum )
```

Set the momentum of all neurons in this layer.

##### Parameters

<code>_momentum</code>	The momentum for all neurons in this layer.
------------------------	---



**4.4.3.25 setNormaliseWeights()**

```
void Layer::setNormaliseWeights (
    WeightNormalisation _normaliseWeights )
```

Normalise the weights.

**Parameters**

<code>_normaliseWeights</code>	Metod of normalisation.
--------------------------------	-------------------------

**4.4.3.26 setStep()**

```
void Layer::setStep (
    long int step )
```

Sets the simulation step in the layer for debug purposes.

**Parameters**

<code>step</code>	Step number.
-------------------	--------------

**4.4.3.27 setUseThreads()**

```
void Layer::setUseThreads (
    int _useThreads ) [inline]
```

Sets if threads should be used.

**Parameters**

<code>_useThreads</code>	0 = no Threads, 1 = Threads
--------------------------	-----------------------------

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

- layer.h
- layer.cpp

**4.5 Neuron Class Reference**

[Neuron](#) which calculates the output and performs learning.

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

## Public Types

- enum [WeightInitMethod](#) { **MAX\_OUTPUT\_RANDOM** = 0, **MAX\_WEIGHT\_RANDOM** = 1, **MAX\_OUTPUT\_RANDOM\_CONST** = 2, **CONST\_WEIGHTS** = 3 }  
*Constants how to init the weights in the neuron.*
- enum [ActivationFunction](#) { **LINEAR** = 0, **TANH** = 1, **RELU** = 2, **REMAXLU** = 3, **TANHLIMIT** = 4 }  
*Activation functions on offer LINEAR: linear unit, TANH: tangens hyperbolicus, RELU: linear rectifier, REMAXLU: as RELU but limits to one.*

## Public Member Functions

- [Neuron](#) (int \_nInputs)  
*Constructor.*
- [~Neuron](#) ()  
*Destructor Tidies up any memory allocations.*
- void [calcOutput](#) ()  
*Calculate the output of the neuron This runs the filters, activation functions, sum it all up.*
- void [doLearning](#) ()  
*Performs the learning Performs ICO learning in the neuron: pre \* error.*
- void [doMaxDet](#) ()  
*Detects max of an input Switches the highest weight to 1 and the others to 0.*
- void [initWeights](#) (double \_max=1, int initBias=1, [WeightInitMethod](#) \_wm=MAX\_OUTPUT\_RANDOM)  
*Init the weights in the neuron.*
- void [setActivationFunction](#) ([ActivationFunction](#) \_activationFunction)  
*Sets the activation function.*
- double [dActivation](#) ()  
*Returns the output of the neuron fed through the derivative of the activation.*
- double [getMinWeightValue](#) ()  
*Minimum weight value.*
- double [getMaxWeightValue](#) ()  
*Maximum weight value.*
- double [getWeightDistanceFromInitialWeights](#) ()  
*Weight development.*
- double [getOutput](#) ()  
*Gets the output of the neuron.*
- double [getSum](#) ()  
*Gets the weighted sum of all inputs pre-activation function.*
- double [getWeight](#) (int \_index)  
*Gets one weight.*
- void [setWeight](#) (int \_index, double \_weight)  
*Sets one weight.*
- void [setError](#) (double \_error)  
*Sets the error in the neuron If the derivative is activated then the derivative of the error is calculated.*
- double [getError](#) ()  
*Gets the error as set by setError.*
- void [setInput](#) (int \_index, double \_value)

- Sets one input.*

  - double `getInput` (int `_index`)

*Get the value at one input.*
- double `getBiasWeight` ()

*Gets the bias weight.*
- void `setBiasWeight` (double `_biasweight`)

*Sets the bias weight.*
- void `setBias` (double `_bias`)

*Sets the bias input value.*
- void `setLearningRate` (double `_learningrate`)

*Sets the learning rate.*
- void `setMomentum` (double `_momentum`)

*Sets the momentum.*
- void `setDecay` (double `_decay`)

*Sets the weight decay over time.*
- double `getDecay` ()

*Gets the weight decay over time.*
- int `getNinputs` ()

*Get the number of inputs to the neuron.*
- void `setGeometry` (int `_width`, int `_height`)

*Tells the layer that it's been a 2D array originally to be a convolutional layer.*
- void `setMask` (int `x`, int `y`, unsigned char `c`)

*Boundary safe manipulation of the convolution mask.*
- void `setMask` (unsigned char `c`)

*Init the whole mask with a single value.*
- unsigned char `getMask` (int `x`, int `y`)

*Boundary safe return of the mask in (x,y) coordinates.*
- unsigned char `getMask` (int `index`)

*Boundary safe return of the mask in flat form.*
- double `getSumOfSquaredWeightVector` ()

*Calculates the sum of the squared weight vector values.*
- double `getEuclideanNormOfWeightVector` ()

*Calculates the Euclidian length of the weight vector.*
- double `getManhattanNormOfWeightVector` ()

*Calculates the Manhattan length of the weight vector /return Manhattan length of the weight vector.*
- double `getInfinityNormOfWeightVector` ()

*Calculates the Infinity norm of the vector.*
- double `getAverageOfWeightVector` ()

*Calculates the average of the weight values.*
- void `normaliseWeights` (double `norm`)

*Normalises the weights with a divisor.*
- void `saveInitialWeights` ()

*Save the initial weights.*
- void `setDebugInfo` (int `_layerIndex`, int `_neuronIndex`)

*Sets debug info populated from [Layer](#).*
- void `setStep` (long int `_step`)

*Sets the simulation step for debugging and logging.*

## Static Public Member Functions

- static void \* [calcOutputThread](#) (void \*object)  
*Wrapper for thread callback for output calc.*
- static void \* [doLearningThread](#) (void \*object)  
*Wrapper for thread callback for learning.*
- static void \* [doMaxDetThread](#) (void \*object)  
*Wrapper for thread callback for maxdet.*

### 4.5.1 Detailed Description

[Neuron](#) which calculates the output and performs learning.

### 4.5.2 Member Enumeration Documentation

#### 4.5.2.1 WeightInitMethod

```
enum Neuron::WeightInitMethod
```

Constants how to init the weights in the neuron.

### 4.5.3 Constructor & Destructor Documentation

#### 4.5.3.1 Neuron()

```
Neuron::Neuron (
    int _nInputs )
```

Constructor.

Parameters

<code><a href="#">_nInputs</a></code>	Number of inputs to the <a href="#">Neuron</a>
---------------------------------------	--

#### 4.5.3.2 ~Neuron()

```
Neuron::~~Neuron ( )
```

Destructor Tidies up any memory allocations.

## 4.5.4 Member Function Documentation

### 4.5.4.1 calcOutputThread()

```
static void* Neuron::calcOutputThread (
    void * object ) [inline], [static]
```

Wrapper for thread callback for output calc.

### 4.5.4.2 dActivation()

```
double Neuron::dActivation ( )
```

Returns the output of the neuron fed through the derivative of the activation.

#### Returns

Result

### 4.5.4.3 doLearning()

```
void Neuron::doLearning ( )
```

Performs the learning Performs ICO learning in the neuron:  $pre * error$ .

### 4.5.4.4 doLearningThread()

```
static void* Neuron::doLearningThread (
    void * object ) [inline], [static]
```

Wrapper for thread callback for learning.

### 4.5.4.5 doMaxDet()

```
void Neuron::doMaxDet ( )
```

Detects max of an input Switches the highest weight to 1 and the others to 0.

#### 4.5.4.6 doMaxDetThread()

```
static void* Neuron::doMaxDetThread (
    void * object ) [inline], [static]
```

Wrapper for thread callback for maxdet.

#### 4.5.4.7 getAverageOfWeightVector()

```
double Neuron::getAverageOfWeightVector ( )
```

Calculates the average of the weight values.

##### Returns

average of the weight values.

#### 4.5.4.8 getBiasWeight()

```
double Neuron::getBiasWeight ( ) [inline]
```

Gets the bias weight.

##### Returns

Bias weight value

#### 4.5.4.9 getDecay()

```
double Neuron::getDecay ( ) [inline]
```

Gets the weight decay over time.

##### Returns

The weight decay value. The larger the faster the weight decay.

#### 4.5.4.10 `getError()`

```
double Neuron::getError ( ) [inline]
```

Gets the error as set by `setError`.

##### Returns

The error value stored in the neuron

#### 4.5.4.11 `getEuclideanNormOfWeightVector()`

```
double Neuron::getEuclideanNormOfWeightVector ( ) [inline]
```

Calculates the Euclidean length of the weight vector.

##### Returns

Euclidean length of the weight vector.

#### 4.5.4.12 `getInfinityNormOfWeightVector()`

```
double Neuron::getInfinityNormOfWeightVector ( )
```

Calculates the Infinity norm of the vector.

/return Infinity norm of the vector.

#### 4.5.4.13 `getInput()`

```
double Neuron::getInput (
    int _index ) [inline]
```

Get the value at one input.

##### Parameters

<code>_index</code>	Index of the input
---------------------	--------------------

##### Returns

Returns the input value

**4.5.4.14 getMask() [1/2]**

```
unsigned char Neuron::getMask (
    int index ) [inline]
```

Boundary safe return of the mask in flat form.

**Parameters**

<i>index</i>	Mask index.
--------------	-------------

**Returns**

The mask at the index: 0 = ignore underlying value, 1 = process underlying value.

**4.5.4.15 getMask() [2/2]**

```
unsigned char Neuron::getMask (
    int x,
    int y )
```

Boundary safe return of the mask in (x,y) coordinates.

**Parameters**

<i>x</i>	Sets the mask value at coordinate x (0 .. width).
<i>y</i>	Sets the mask value at coordinate y (0 .. height).

**Returns**

The mask at x,y: 0 = ignore underlying value, 1 = process underlying value.

**4.5.4.16 getMaxWeightValue()**

```
double Neuron::getMaxWeightValue ( )
```

Maximum weight value.

**Returns**

The maximum weight value in this neuron



#### 4.5.4.17 getMinWeightValue()

```
double Neuron::getMinWeightValue ( )
```

Minimum weight value.

##### Returns

The minimum weight value in this neuron

#### 4.5.4.18 getNinputs()

```
int Neuron::getNinputs ( ) [inline]
```

Get the number of inputs to the neuron.

##### Returns

The number of inputs

#### 4.5.4.19 getOutput()

```
double Neuron::getOutput ( ) [inline]
```

Gets the output of the neuron.

##### Returns

The overall output of the neuron after the activation function

#### 4.5.4.20 getSum()

```
double Neuron::getSum ( ) [inline]
```

Gets the weighted sum of all inputs pre-activation function.

##### Returns

Weighted sum (linear)

**4.5.4.21 getSumOfSquaredWeightVector()**

```
double Neuron::getSumOfSquaredWeightVector ( )
```

Calculates the sum of the squared weight vector values.

**Returns**

The squared weight vector values.

**4.5.4.22 getWeight()**

```
double Neuron::getWeight (
    int _index ) [inline]
```

Gets one weight.

**Parameters**

<code>_index</code>	The input index
---------------------	-----------------

**Returns**

The weight value at one input and one filter

**4.5.4.23 getWeightDistanceFromInitialWeights()**

```
double Neuron::getWeightDistanceFromInitialWeights ( )
```

Weight development.

**Returns**

Returns the Euclidean distance of the weights from their starting position

**4.5.4.24 initWeights()**

```
void Neuron::initWeights (
    double _max = 1,
    int initBias = 1,
    WeightInitMethod _wm = MAX_OUTPUT_RANDOM )
```

Init's the weights in the neuron.

## Parameters

<code>_max</code>	Maximum value of the weights.
<code>initBias</code>	If one also the bias weight is initialised.
<code>_wm</code>	Method how to init the weights as defined by <code>WeightInitMethod</code> .

**4.5.4.25 normaliseWeights()**

```
void Neuron::normaliseWeights (
    double norm )
```

Normalises the weights with a divisor.

## Parameters

<code>norm</code>	Divisor which normalises the weights.
-------------------	---------------------------------------

**4.5.4.26 saveInitialWeights()**

```
void Neuron::saveInitialWeights ( )
```

Save the initial weights.

This saves the initial weights for later comparisons. For internal use.

**4.5.4.27 setActivationFunction()**

```
void Neuron::setActivationFunction (
    ActivationFunction _activationFunction ) [inline]
```

Sets the activation function.

## Parameters

<code>_activationFunction</code>	Sets the activation function according to <code>ActivationFunction</code> .
----------------------------------	---

**4.5.4.28 setBias()**

```
void Neuron::setBias (
    double _bias ) [inline]
```

Sets the bias input value.

## Parameters

<code>_bias</code>	Bias value.
--------------------	-------------

**4.5.4.29 setBiasWeight()**

```
void Neuron::setBiasWeight (
    double _biasweight ) [inline]
```

Sets the bias weight.

## Parameters

<code>_biasweight</code>	Bias value.
--------------------------	-------------

**4.5.4.30 setDebugInfo()**

```
void Neuron::setDebugInfo (
    int _layerIndex,
    int _neuronIndex ) [inline]
```

Sets debug info populated from [Layer](#).

## Parameters

<code>_layerIndex</code>	The layer the neuron is in.
<code>_neuronIndex</code>	The index of the neuron in the layer.

**4.5.4.31 setDecay()**

```
void Neuron::setDecay (
    double _decay ) [inline]
```

Sets the weight decay over time.

## Parameters

<code>_decay</code>	The larger the faster the weight decay.
---------------------	---

#### 4.5.4.32 setError()

```
void Neuron::setError (
    double _error )
```

Sets the error in the neuron. If the derivative is activated then the derivative of the error is calculated.

##### Parameters

<code>_error</code>	Sets the error of the neuron.
---------------------	-------------------------------

#### 4.5.4.33 setGeometry()

```
void Neuron::setGeometry (
    int _width,
    int _height ) [inline]
```

Tells the layer that it's been a 2D array originally to be a convolutional layer.

`_width * _height == nInputs`. Otherwise an exception is triggered. The geometry entered here is then used in the mask operations so that every neuron is able to process a subset of the input space, for example an image and thus becomes a localised receptive field.

##### Parameters

<code>_width</code>	The width of the layer
<code>_height</code>	of the layer

#### 4.5.4.34 setInput()

```
void Neuron::setInput (
    int _index,
    double _value ) [inline]
```

Sets one input.

##### Parameters

<code>_index</code>	Index of the input.
<code>_value</code>	of the input.

**4.5.4.35 setLearningRate()**

```
void Neuron::setLearningRate (
    double _learningrate ) [inline]
```

Sets the learning rate.

**Parameters**

<i>_learningrate</i>	The learning rate
----------------------	-------------------

**4.5.4.36 setMask() [1/2]**

```
void Neuron::setMask (
    int x,
    int y,
    unsigned char c )
```

Boundary safe manipulation of the convolution mask.

Sets the convolution mask using the geometry defined by setGeometry.

**Parameters**

<i>x</i>	Sets the mask value at coordinate x (0 .. width).
<i>y</i>	Sets the mask value at coordinate y (0 .. height).
<i>c</i>	Sets the mask: 0 = ignore underlying value, 1 = process underlying value.

**4.5.4.37 setMask() [2/2]**

```
void Neuron::setMask (
    unsigned char c )
```

Init the whole mask with a single value.

**Parameters**

<i>c</i>	Sets the mask for the whole array. 0 = ignore the entire input, 1 = process every input.
----------	--

**4.5.4.38 setMomentum()**

```
void Neuron::setMomentum (
    double _momentum ) [inline]
```

Sets the momentum.

Sets the inertia of the learning.

#### Parameters

<code>_momentum</code>	The new momentum
------------------------	------------------

#### 4.5.4.39 setStep()

```
void Neuron::setStep (
    long int _step ) [inline]
```

Sets the simulation step for debugging and logging.

#### Parameters

<code>_step</code>	Current simulation step.
--------------------	--------------------------

#### 4.5.4.40 setWeight()

```
void Neuron::setWeight (
    int _index,
    double _weight ) [inline]
```

Sets one weight.

#### Parameters

<code>_index</code>	The input index
<code>_weight</code>	The weight value

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

- neuron.h
- neuron.cpp





# Index

- ~FeedforwardClosedloopLearning
  - FeedforwardClosedloopLearning, [10](#)
- ~Layer
  - Layer, [20](#)
- ~Neuron
  - Neuron, [30](#)
- Bandpass, [7](#)
  - getOutput, [8](#)
- calcOutputs
  - Layer, [20](#)
- calcOutputThread
  - Neuron, [31](#)
- dActivation
  - Neuron, [31](#)
- doLearning
  - Layer, [20](#)
  - Neuron, [31](#)
- doLearningThread
  - Neuron, [31](#)
- doMaxDet
  - Neuron, [31](#)
- doMaxDetThread
  - Neuron, [31](#)
- doNormaliseWeights
  - Layer, [20](#)
- doStep
  - FeedforwardClosedloopLearning, [10](#)
  - FeedforwardClosedloopLearningWithFilterbank, [17](#)
- FeedforwardClosedloopLearning, [8](#)
  - ~FeedforwardClosedloopLearning, [10](#)
  - doStep, [10](#)
  - FeedforwardClosedloopLearning, [10](#)
  - getLayer, [11](#)
  - getLayers, [11](#)
  - getNumInputs, [11](#)
  - getNumLayers, [11](#)
  - getOutput, [12](#)
  - getOutputLayer, [12](#)
  - initWeights, [12](#)
  - loadModel, [13](#)
  - saveModel, [13](#)
  - seedRandom, [13](#)
  - setActivationFunction, [13](#)
  - setBias, [14](#)
  - setDecay, [14](#)
  - setLearningRate, [14](#)
  - setLearningRateDiscountFactor, [15](#)
  - setMomentum, [15](#)
  - FeedforwardClosedloopLearningWithFilterbank, [15](#)
  - doStep, [17](#)
  - FeedforwardClosedloopLearningWithFilterbank, [17](#)
- getAverageOfWeightVector
  - Neuron, [32](#)
- getBiasWeight
  - Neuron, [32](#)
- getDecay
  - Neuron, [32](#)
- getError
  - Layer, [21](#)
  - Neuron, [32](#)
- getEuclideanNormOfWeightVector
  - Neuron, [33](#)
- getInfinityNormOfWeightVector
  - Neuron, [33](#)
- getInput
  - Neuron, [33](#)
- getLayer
  - FeedforwardClosedloopLearning, [11](#)
- getLayers
  - FeedforwardClosedloopLearning, [11](#)
- getMask
  - Neuron, [33](#), [34](#)
- getMaxWeightValue
  - Neuron, [34](#)
- getMinWeightValue
  - Neuron, [34](#)
- getNeuron
  - Layer, [21](#)
- getNinputs
  - Layer, [21](#)
  - Neuron, [35](#)
- getNneurons
  - Layer, [21](#)
- getNumInputs
  - FeedforwardClosedloopLearning, [11](#)
- getNumLayers
  - FeedforwardClosedloopLearning, [11](#)
- getOutput
  - Bandpass, [8](#)
  - FeedforwardClosedloopLearning, [12](#)
  - Layer, [22](#)
  - Neuron, [35](#)
- getOutputLayer
  - FeedforwardClosedloopLearning, [12](#)
- getSum

- Neuron, 35
- getSumOfSquaredWeightVector
  - Neuron, 35
- getWeight
  - Neuron, 36
- getWeightDistanceFromInitialWeights
  - Layer, 22
  - Neuron, 36
- initWeights
  - FeedforwardClosedloopLearning, 12
  - Layer, 22
  - Neuron, 36
- Layer, 18
  - ~Layer, 20
  - calcOutputs, 20
  - doLearning, 20
  - doNormaliseWeights, 20
  - getError, 21
  - getNeuron, 21
  - getNinputs, 21
  - getNneurons, 21
  - getOutput, 22
  - getWeightDistanceFromInitialWeights, 22
  - initWeights, 22
  - Layer, 20
  - saveWeightMatrix, 23
  - setActivationFunction, 23
  - setBias, 23
  - setConvolution, 23
  - setDebugInfo, 24
  - setDecay, 24
  - setError, 24, 25
  - setErrors, 25
  - setInput, 25
  - setInputs, 26
  - setLearningRate, 26
  - setMaxDetLayer, 26
  - setMomentum, 26
  - setNormaliseWeights, 27
  - setStep, 27
  - setUseThreads, 27
- loadModel
  - FeedforwardClosedloopLearning, 13
- Neuron, 27
  - ~Neuron, 30
  - calcOutputThread, 31
  - dActivation, 31
  - doLearning, 31
  - doLearningThread, 31
  - doMaxDet, 31
  - doMaxDetThread, 31
  - getAverageOfWeightVector, 32
  - getBiasWeight, 32
  - getDecay, 32
  - getError, 32
  - getEuclideanNormOfWeightVector, 33
  - getInfinityNormOfWeightVector, 33
  - getInput, 33
  - getMask, 33, 34
  - getMaxWeightValue, 34
  - getMinWeightValue, 34
  - getNinputs, 35
  - getOutput, 35
  - getSum, 35
  - getSumOfSquaredWeightVector, 35
  - getWeight, 36
  - getWeightDistanceFromInitialWeights, 36
  - initWeights, 36
  - Neuron, 30
  - normaliseWeights, 37
  - saveInitialWeights, 37
  - setActivationFunction, 37
  - setBias, 37
  - setBiasWeight, 38
  - setDebugInfo, 38
  - setDecay, 38
  - setError, 38
  - setGeometry, 39
  - setInput, 39
  - setLearningRate, 39
  - setMask, 40
  - setMomentum, 40
  - setStep, 41
  - setWeight, 41
  - WeightInitMethod, 30
- normaliseWeights
  - Neuron, 37
- saveInitialWeights
  - Neuron, 37
- saveModel
  - FeedforwardClosedloopLearning, 13
- saveWeightMatrix
  - Layer, 23
- seedRandom
  - FeedforwardClosedloopLearning, 13
- setActivationFunction
  - FeedforwardClosedloopLearning, 13
  - Layer, 23
  - Neuron, 37
- setBias
  - FeedforwardClosedloopLearning, 14
  - Layer, 23
  - Neuron, 37
- setBiasWeight
  - Neuron, 38
- setConvolution
  - Layer, 23
- setDebugInfo
  - Layer, 24
  - Neuron, 38
- setDecay
  - FeedforwardClosedloopLearning, 14
  - Layer, 24
  - Neuron, 38

- setError
  - Layer, [24](#), [25](#)
  - Neuron, [38](#)
- setErrors
  - Layer, [25](#)
- setGeometry
  - Neuron, [39](#)
- setInput
  - Layer, [25](#)
  - Neuron, [39](#)
- setInputs
  - Layer, [26](#)
- setLearningRate
  - FeedforwardClosedloopLearning, [14](#)
  - Layer, [26](#)
  - Neuron, [39](#)
- setLearningRateDiscountFactor
  - FeedforwardClosedloopLearning, [15](#)
- setMask
  - Neuron, [40](#)
- setMaxDetLayer
  - Layer, [26](#)
- setMomentum
  - FeedforwardClosedloopLearning, [15](#)
  - Layer, [26](#)
  - Neuron, [40](#)
- setNormaliseWeights
  - Layer, [27](#)
- setStep
  - Layer, [27](#)
  - Neuron, [41](#)
- setUseThreads
  - Layer, [27](#)
- setWeight
  - Neuron, [41](#)
- WeightInitMethod
  - Neuron, [30](#)