Feedforward Closedloop Learning

Generated by Doxygen 1.8.17

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:

Chapter 3

Class Index

3.1 Class List

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

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:

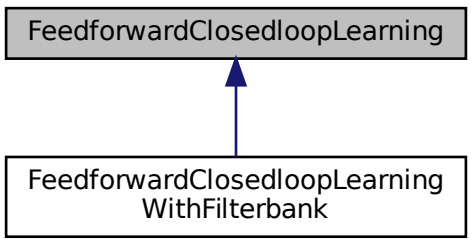
- bandpass.h
- bandpass.cpp

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)
  Inits 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 (C) 2017,2018, Paul Miller paul@glasgowneuro.↩ tech

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

num_of_inputs	Number of inputs in the input layer
num_of_neurons_per_layer_array	Number of neurons in each layer
_num_layers	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

input	Array with the input values
error	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	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

index	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 )
```

Inits the weights in all layers.

Parameters

max	Maximum value of the weights.
initBias	If the bias also should be initialised.
weightInitMethod	See Neuron::WeightInitMethod for the options.

4.2.3.10 loadModel()

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

Loads the while network.

Parameters

name	filename

4.2.3.11 saveModel()

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

Saves the whole network.

Parameters

name	filename

4.2.3.12 seedRandom()

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

Seeds the random number generator.

Parameters

s	An arbitratry number.

4.2.3.13 setActivationFunction()

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

Sets the activation function of the Neuron.

Parameters

_activationFunction	See Neuron::ActivationFunction for the different options.

4.2.3.14 setBias()

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

Sets globally the bias.

Parameters

_bias	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

decay	The larger the faster the decay.

4.2.3.16 setLearningRate()

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

Sets globally the learning rate.

Parameters

learningRate	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

_learningRateDiscountFactor	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

momentum	Defines the intertia of the weight change over time.

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

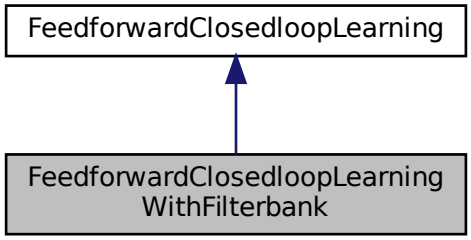
- fcl.h

4.3 FeedforwardClosedloopLearningWithFilterbank Class Reference

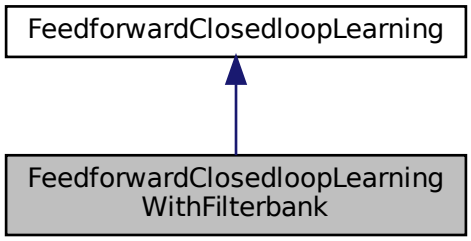
Derived classes of the FeedforwardClosedloopLearning class for special functionality.

```
#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

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

input	Array with the input values
error	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_NORM_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)

  Inits the weights.
- double getOutput (int index)

  Gets the outpuut 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 compuations 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

_nNeurons	Number of neurons in the layer.
_nInputs	Number of inputs to the Layer.
_nFilters	Number of lowpass filters at each input.
_minT	Minimum time of the lowpass filter.
_maxT	Maximum time of the lowpass filter.

(C) 2017, Bernd Porr bernd@glasgowneuro.tech (C) 2017, Paul Miller 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	Index of the neuron

4.4.3.5 getNeuron()

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

Gets a pointer to one neuron.

Parameters

index	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 outpuut of one neuron.

Parameters

index	The index number of the neuron.

Returns

Retuns the double valye 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 )
```

Inits the weights.

Parameters

_max	Maximum value if using random init.
initBiasWeight	if one also the bias weight is initialised.
weightInitMethod	The methid employed to init the weights.

4.4.3.11 saveWeightMatrix()

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

Save weight matrix for documentation and debugging.

Parameters

filename	The filename it should be saved to.

4.4.3.12 setActivationFunction()

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

Set the activation function.

Parameters

_activationFunction	The activation function. See: Neuron::ActivationFunction

4.4.3.13 setBias()

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

Sets the global bias for all neurons.

Parameters

_bias	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

width	The width of the convolutional window.
height	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

layerIndex	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

_decay	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

_error	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	Index of the neuron
_error	The error to be set

4.4.3.19 setErrors()

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

Sets all errors from an input array.

Parameters

_errors	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

inputIndex	The index number of the input.
input	The value of the input

4.4.3.21 setInputs()

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

Sets all inputs from an input array.

Parameters

_inputs	array of all inputs

4.4.3.22 setLearningRate()

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

Sets the learning rate of all neurons.

Parameters

_learningRate	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

_momentum	The momentum for all neurons in this layer.

4.4.3.25 setNormaliseWeights()

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

Normalise the weights.

Parameters

_normaliseWeights	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

step	Step number.

4.4.3.27 setUseThreads()

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

Sets if threads should be used.

Parameters

_useThreads	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_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)

  Inits 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 Eucledian 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

_nInputs	Number of inputs to the Neuron

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 Eucledian length of the weight vector.

Returns

Eucledian 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

_index	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

index	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

x	Sets the mask value at coordinate x (0 .. width).
y	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 numer 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

_index	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 )
```

Inits the weights in the neuron.

Parameters

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

4.5.4.25 normaliseWeights()

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

Normalises the weights with a divisor.

Parameters

norm	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

_activationFunction	Sets the activiation function according to ActivationFunction.

4.5.4.28 setBias()

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

Sets the bias input value.

Parameters

_bias	Bias value.

4.5.4.29 setBiasWeight()

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

Sets the bias weight.

Parameters

_biasweight	Bias value.

4.5.4.30 setDebugInfo()

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

Sets debug info populated from Layer.

Parameters

_layerIndex	The layer the neuron is in.
_neuronIndex	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

_decay	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

_error	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

_width	The width of the layer
_height	of the layer

4.5.4.34 setInput()

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

Sets one input.

Parameters

_index	Index of the input.
_value	of the input.

4.5.4.35 setLearningRate()

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

Sets the learning rate.

Parameters

_learningrate	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

x	Sets the mask value at coordinate x (0 .. width).
y	Sets the mask value at coordinate y (0 .. height).
c	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

c	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

_momentum	The new momentum

4.5.4.39 setStep()

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

Sets the simulation step for debugging and logging.

Parameters

_step	Current simulation step.

4.5.4.40 setWeight()

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

Sets one weight.

Parameters

_index	The input index
_weight	The weight value

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

- neuron.h
- neuron.cpp

Index