Deep Neuronal Filter

Generated by Doxygen 1.8.17

Chapter 1

Deep Neuronal Filter (DNF)

A noise reduction filter using deep networks in autoencoder configuration.

github: https://github.com/berndporr/deepNeuronalFilter

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Class Documentation

3.1 DNF Class Reference

Main Deep Neuronal Network main class.

`#include <dnf.h>`

Public Member Functions

- DNF (const int NLAYERS, const int numTaps, double fs, Neuron::actMethod am=Neuron::Act_Tanh)

  Constructor which sets up the delay lines, network layers and also calculates the number of neurons per layer so that the final layer always just has one neuron.
- double filter (double signal, double noise)

  Realtime sample by sample filtering operation.
- Net & getNet () const

  Returns a reference to the whole neural network.
- const int getSignalDelaySteps () const

  Returns the length of the delay line which delays the signal polluted with noise.
- const double getDelayedSignal () const

  Returns the delayed with noise polluted signal by the delay indicated by getSignalDelaySteps().
- const double getRemover () const

  Returns the remover signal.
- const double getOutput () const

  Returns the output of the DNF: the the noise free signal.
- ~DNF ()

  Frees the memory used by the DNF.

3.1.1 Detailed Description

Main Deep Neuronal Network main class.

It's designed to be as simple as possible with only a few parameters as possible.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 DNF()

```
DNF::DNF (
            const int NLAYERS,
            const int numTaps,
            double fs,
            Neuron::actMethod am = Neuron::Act_Tanh )  [inline]
```

Constructor which sets up the delay lines, network layers and also calculates the number of neurons per layer so that the final layer always just has one neuron.

Parameters

NLAYERS	Number of layers
numTaps	Number of taps for the delay line feeding into the 1st layer
fs	Sampling rate of the signals used in Hz.
am	The activation function for the neurons. Default is tanh.

3.1.3 Member Function Documentation

3.1.3.1 filter()

```
double DNF::filter (
            double signal,
            double noise )  [inline]
```

Realtime sample by sample filtering operation.

Parameters

signal	The signal contaminated with noise. Should be less than one.
noise	The reference noise. Should be less than one.

Returns

The filtered signal where the noise has been removed by the DNF.

3.1.3.2 getDelayedSignal()

```
const double DNF::getDelayedSignal ( ) const  [inline]
```

Returns the delayed with noise polluted signal by the delay indicated by getSignalDelaySteps().

Returns

The delayed noise polluted signal sample.

3.1.3.3 getNet()

```
Net& DNF::getNet ( ) const  [inline]
```

Returns a reference to the whole neural network.

Returns

A reference to Net.

3.1.3.4 getOutput()

```
const double DNF::getOutput ( ) const  [inline]
```

Returns the output of the DNF: the the noise free signal.

Returns

The current output of the DNF which is idential to filter().

3.1.3.5 getRemover()

```
const double DNF::getRemover ( ) const  [inline]
```

Returns the remover signal.

Returns

The current remover signal sample.

3.1.3.6 getSignalDelaySteps()

```
const int DNF::getSignalDelaySteps ( ) const  [inline]
```

Returns the length of the delay line which delays the signal polluted with noise.

Returns

Number of delay steps in samples.

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

- dnf.h

3.2 Layer Class Reference

This is the class for creating layers that are contained inside the Net class.

```
#include <Layer.h>
```

Public Types

- enum whichGradient { **exploding** = 0, **average** = 1, **vanishing** = 2 }
  Options for what gradient of a chosen error to monitor.

Public Member Functions

- Layer (int _nNeurons, int _nInputs, int _subject, string _trial)
  Constructor for Layer: it initialises the neurons internally.
- ~Layer ()
  Destructor De-allocated any memory.
- void initLayer (int _layerIndex, Neuron::weightInitMethod _wim, Neuron::biasInitMethod _bim, Neuron::actMethod _am)
  Initialises each layer with specific methods for weight/bias initialisation and activation function of neurons.
- void setlearningRate (double _w_learningRate, double _b_learningRate)
  Sets the learning rate.
- void setInputs (const double *_inputs, const double scale=1.0, const unsigned int offset=0, const int n=-1)
  Sets the inputs to all neurons in the first hidden layer only.
- void propInputs (int _index, double _value)
  Sets the inputs to all neurons in the deeper layers (excluding the first hidden layer)
- void calcOutputs ()
  Demands that all neurons in this layer calculate their output.
- void setError (double _error)
  Sets the error to be propagated backward at all neurons in the output layer only.
- double getGradient (whichGradient _whichGradient)
  Allows for accessing the error that propagates backward in the network.
- void updateWeights ()
  Requests that all neurons perform one iteration of learning.

- Neuron * getNeuron (int _neuronIndex)

  Allows access to a specific neuron.
- int getnNeurons ()

  Reports the number of neurons in this layer.
- double getOutput (int _neuronIndex)

  Allows for accessing the activation of a specific neuron.
- double getSumOutput (int _neuronIndex)

  Allows for accessing the sum output of any specific neuron.
- double getWeights (int _neuronIndex, int _weightIndex)

  Allows for accessing any specific weights in the layer.
- double getWeightChange ()

  Accesses the total sum of weight changes of all the neurons in this layer.
- double getWeightDistance ()

  Performs squared root on the weight change.
- double getGlobalError (int _neuronIndex)

  Reports the global error that is assigned to a specific neuron in this layer.
- double getInitWeight (int _neuronIndex, int _weightIndex)

  Reports the initial value that was assigned to a specific weight at the initialisatin of the network.
- void saveWeights ()

  Saves the temporal weight change of all weights in all neurons into files.
- void snapWeights (string prefix, string _trial, int _subject)

  Snaps the final distribution of weights in a specific layer, this is overwritten every time the function is called.
- void **snapWeightsMatrixFormat** (string prefix)
- void printLayer ()

  Prints on the console a full tree of this layer with the values of all weights and outputs for all neurons.

3.2.1 Detailed Description

This is the class for creating layers that are contained inside the Net class.

The Layer instances in turn contain neurons.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Layer()

```
Layer::Layer (
            int _nNeurons,
            int _nInputs,
            int _subject,
            string _trial )
```

Constructor for Layer: it initialises the neurons internally.

Parameters

_nNeurons	Total number of neurons in the layer
_nInputs	Total number of inputs to that layer

3.2.3 Member Function Documentation

3.2.3.1 getGlobalError()

```
double Layer::getGlobalError (
            int _neuronIndex )
```

Reports the global error that is assigned to a specific neuron in this layer.

Parameters

_neuronIndex	the neuron index

Returns

the value of the global error

3.2.3.2 getGradient()

```
double Layer::getGradient (
            whichGradient _whichGradient )
```

Allows for accessing the error that propagates backward in the network.

Parameters

_neuronIndex	The index from which the error is requested

Returns

Returns the error of the chosen neuron

3.2.3.3 getInitWeight()

```
double Layer::getInitWeight (
            int _neuronIndex,
            int _weightIndex )
```

Reports the initial value that was assigned to a specific weight at the initialisatin of the network.

Parameters

_neuronIndex	Index of the neuron containing the weight
_weightIndex	Index of the weight

Returns

3.2.3.4 getNeuron()

```
Neuron* Layer::getNeuron (
            int _neuronIndex )
```

Allows access to a specific neuron.

Parameters

_neuronIndex	The index of the neuron to access

Returns

A pointer to that neuron

3.2.3.5 getnNeurons()

```
int Layer::getnNeurons ( )
```

Reports the number of neurons in this layer.

Returns

The total number of neurons in this layer

3.2.3.6 getOutput()

```
double Layer::getOutput (
            int _neuronIndex )
```

Allows for accessing the activation of a specific neuron.

Parameters

_neuronIndex	The index of the neuron

Returns

the activation of that neuron

3.2.3.7 getSumOutput()

```
double Layer::getSumOutput (
            int _neuronIndex )
```

Allows for accessing the sum output of any specific neuron.

Parameters

_neuronIndex	The index of the neuron to access

Returns

Returns the wighted sum of the inputs to that neuron

3.2.3.8 getWeightChange()

```
double Layer::getWeightChange ( )
```

Accesses the total sum of weight changes of all the neurons in this layer.

Returns

sum of weight changes all neurons

3.2.3.9 getWeightDistance()

```
double Layer::getWeightDistance ( )
```

Performs squared root on the weight change.

Returns

The sqr of the weight changes

3.2.3.10 getWeights()

```
double Layer::getWeights (
            int _neuronIndex,
            int _weightIndex )
```

Allows for accessing any specific weights in the layer.

Parameters

_neuronIndex	The index of the neuron containing that weight
_weightIndex	The index of the input to which that weight is assigned

Returns

Returns the chosen weight

3.2.3.11 initLayer()

```
void Layer::initLayer (
            int _layerIndex,
            Neuron::weightInitMethod _wim,
            Neuron::biasInitMethod _bim,
            Neuron::actMethod _am )
```

Initialises each layer with specific methods for weight/bias initialisation and activation function of neurons.

Parameters

_layerIndex	The index that is assigned to this layer by the Net class
_wim	weights initialisation method, see Neuron::weightInitMethod for different options
_bim	biases initialisation method, see Neuron::biasInitMethod for different options
_am	activation method, see Neuron::actMethod for different options

3.2.3.12 propInputs()

```
void Layer::propInputs (
            int _index,
            double _value )
```

Sets the inputs to all neurons in the deeper layers (excluding the first hidden layer)

Parameters

_index	The index of the input
_value	The value of the input

3.2.3.13 setError()

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

Sets the error to be propagated backward at all neurons in the output layer only.

Parameters

_leadError	the error to be propagated backward

3.2.3.14 setInputs()

```
void Layer::setInputs (
            const double * _inputs,
            const double scale = 1.0,
            const unsigned int offset = 0,
            const int n = -1 )
```

Sets the inputs to all neurons in the first hidden layer only.

Parameters

_inputs	A pointer to an array of inputs

3.2.3.15 setlearningRate()

```
void Layer::setlearningRate (
            double _w_learningRate,
            double _b_learningRate )
```

Sets the learning rate.

Parameters

_learningRate	Sets the learning rate for all neurons.

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

- Layer.h

3.3 Net Class Reference

Net is the main class used to set up a neural network used for closed-loop Deep Learning.

`#include <Net.h>`

Public Member Functions

- Net (const int _nLayers, const int *const _nNeurons, const int _nInputs, const int _subject, const string _trial)
  Constructor: The neural network that performs the learning.
- ~Net ()
  Destructor De-allocated any memory.
- void initNetwork (Neuron::weightInitMethod _wim, Neuron::biasInitMethod _bim, Neuron::actMethod _am)
  Dictates the initialisation of the weights and biases and determines the activation function of the neurons.
- void setLearningRate (double _w_learningRate, double _b_learningRate)
  Sets the learning rate.
- void setInputs (const double *_inputs, const double scale=1.0, const unsigned int offset=0, const int n=-1)
  Sets the inputs to the network in each iteration of learning, needs to be placed in an infinite loop.
- void propInputs ()
  It propagates the inputs forward through the network.
- void propErrorBackward ()
  Propagates the error backward.
- void setError (double _leadError)
  Sets the error at the output layer to be propagated backward.
- double getGradient (Layer::whichGradient _whichGradient)
  It provides a measure of how the magnitude of the error changes through the layers to alarm for vanishing or exploding gradients.
- void updateWeights ()
  Requests that all layers perform one iteration of learning.
- Layer * getLayer (int _layerIndex)
  Allows Net to access each layer.
- double getOutput (int _neuronIndex)
  Allows the user to access the activation output of a specific neuron in the output layer only.
- double getSumOutput (int _neuronIndex)
  Allows the user to access the weighted sum output of a specific neuron in output layer only.
- int getnLayers ()
  Informs on the total number of hidden layers (excluding the input layer)
- int getnInputs ()
  Informs on the total number of inputs to the network.
- double getWeightDistance ()
  Allows for monitoring the overall weight change of the network.
- double getLayerWeightDistance (int _layerIndex)
  Allows for monitoring the weight change in a specific layer of the network.
- double getWeights (int _layerIndex, int _neuronIndex, int _weightIndex)
  Grants access to a specific weight in the network.
- int getnNeurons ()
  Informs on the total number of neurons in the network.
- void saveWeights ()
  Saves the temporal changes of all weights in all neurons into files.
- void snapWeights (string prefix, string _trial, int _subject)
  Snaps the final distribution of all weights in a specific layer, this is overwritten every time the function is called.
- void **snapWeightsMatrixFormat** (string prefix)
- void printNetwork ()
  Prints on the console a full tree of the network with the values of all weights and outputs for all neurons.

3.3.1 Detailed Description

Net is the main class used to set up a neural network used for closed-loop Deep Learning.

It initialises all the layers and the neurons internally.

(C) 2019,2020, Bernd Porr bernd@glasgowneuro.tech (C) 2019,2020, Sama Daryanavard 2089166d@student.gla.ac.uk

GNU GENERAL PUBLIC LICENSE

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Net()

```
Net::Net (
            const int _nLayers,
            const int *const _nNeurons,
            const int _nInputs,
            const int _subject,
            const string _trial )
```

Constructor: The neural network that performs the learning.

Parameters

_nLayers	Total number of hidden layers, excluding the input layer
_nNeurons	A pointer to an int array with number of neurons for all layers need to have the length of _nLayers.
_nInputs	Number of Inputs to the network

3.3.3 Member Function Documentation

3.3.3.1 getGradient()

```
double Net::getGradient (
            Layer::whichGradient _whichGradient )
```

It provides a measure of how the magnitude of the error changes through the layers to alarm for vanishing or exploding gradients.

Parameters

_whichError	choose what error to monitor, for more information see Neuron::whichError
_whichGradient	choose what gradient of the chosen error to monitor, for more information see Layer::whichGradient

Returns

Returns the ratio of the chosen gradient in the last layer to the the first layer

3.3.3.2 getLayer()

```
Layer* Net::getLayer (
            int _layerIndex )
```

Allows Net to access each layer.

Parameters

_layerIndex	the index of the chosen layer

Returns

A pointer to the chosen Layer

3.3.3.3 getLayerWeightDistance()

```
double Net::getLayerWeightDistance (
            int _layerIndex )
```

Allows for monitoring the weight change in a specific layer of the network.

Parameters

_layerIndex	The index of the chosen layer

Returns

returns the Euclidean wight distance of neurons in the chosen layer from their initial value

3.3.3.4 getnInputs()

```
int Net::getnInputs ( )
```

Informs on the total number of inputs to the network.

Returns

Total number of inputs

3.3.3.5 getnLayers()

```
int Net::getnLayers ( )
```

Informs on the total number of hidden layers (excluding the input layer)

Returns

Total number of hidden layers in the network

3.3.3.6 getnNeurons()

```
int Net::getnNeurons ( )
```

Informs on the total number of neurons in the network.

Returns

The total number of neurons

3.3.3.7 getOutput()

```
double Net::getOutput (
            int _neuronIndex )
```

Allows the user to access the activation output of a specific neuron in the output layer only.

Parameters

_neuronIndex	The index of the chosen neuron

Returns

The value at the output of the chosen neuron

3.3.3.8 getSumOutput()

```
double Net::getSumOutput (
            int _neuronIndex )
```

Allows the user to access the weighted sum output of a specific neuron in output layer only.

Parameters

_neuronIndex	The index of the chosen neuron

Returns

The value at the sum output of the chosen neuron

3.3.3.9 getWeightDistance()

```
double Net::getWeightDistance ( )
```

Allows for monitoring the overall weight change of the network.

Returns

returns the Euclidean wight distance of all neurons in the network from their initial value

3.3.3.10 getWeights()

```
double Net::getWeights (
            int _layerIndex,
            int _neuronIndex,
            int _weightIndex )
```

Grants access to a specific weight in the network.

Parameters

_layerIndex	Index of the layer that contains the chosen weight
_neuronIndex	Index of the neuron in the chosen layer that contains the chosen weight
_weightIndex	Index of the input to which the chosen weight is assigned

Returns

returns the value of the chosen weight

3.3.3.11 initNetwork()

```
void Net::initNetwork (
            Neuron::weightInitMethod _wim,
```

```
        Neuron::biasInitMethod _bim,
        Neuron::actMethod _am )
```

Dictates the initialisation of the weights and biases and determines the activation function of the neurons.

Parameters

_wim	weights initialisation method, see Neuron::weightInitMethod for different options
_bim	biases initialisation method, see Neuron::biasInitMethod for different options
_am	activation method, see Neuron::actMethod for different options

3.3.3.12 setError()

```
void Net::setError (
            double _leadError )
```

Sets the error at the output layer to be propagated backward.

Parameters

_leadError	The closed-loop error for learning

3.3.3.13 setInputs()

```
void Net::setInputs (
            const double * _inputs,
            const double scale = 1.0,
            const unsigned int offset = 0,
            const int n = -1 )
```

Sets the inputs to the network in each iteration of learning, needs to be placed in an infinite loop.

Parameters

_inputs	A pointer to the array of inputs

3.3.3.14 setLearningRate()

```
void Net::setLearningRate (
            double _w_learningRate,
            double _b_learningRate )
```

Sets the learning rate.

Parameters

_learningRate	Sets the learning rate for all layers and neurons.

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

- Net.h

3.4 Neuron Class Reference

This is the class for creating neurons inside the Layer class.

`#include <Neuron.h>`

Public Types

- enum biasInitMethod { **B_NONE** = 0, **B_RANDOM** = 1 }

  Options for method of initialising biases 0 for initialising all weights to zero 1 for initialising all weights to one 2 for initialising all weights to a random value between 0 and 1.
- enum weightInitMethod {
  W_ZEROS = 0, **W_ONES** = 1, **W_RANDOM** = 2, **W_ONES_NORM** = 3,
  W_RANDOM_NORM = 5 }

  Options for method of initialising weights 0 for initialising all weights to zero 1 for initialising all weights to one 2 for initialising all weights to a random value between 0 and 1.
- enum actMethod { **Act_Sigmoid** = 1, **Act_Tanh** = 2, **Act_ReLU** = 3, **Act_NONE** = 0 }

  Options for activation functions of the neuron 0 for using the logistic function 1 for using the hyperbolic tan function 2 for unity function (no activation)
- enum whichError { **onBackwardError** = 0, **onMidError** = 1, **onForwardError** = 2 }

  Options for choosing an error to monitor the gradient of 0 for monitoring the error that propagates backward 1 for monitoring the error that propagates from the middle and bilaterally 2 for monitoring the error that propagates forward.

Public Member Functions

- Neuron (int _nInputs)

  Constructor for the Neuron class: it initialises a neuron with specific number fo inputs to that neuron.
- ~Neuron ()

  Destructor De-allocated any memory.
- void initNeuron (int _neuronIndex, int _layerIndex, weightInitMethod _wim, biasInitMethod _bim, actMethod _am)

  Initialises the neuron with the given methods for weight/bias initialisation and for activation function.
- void setLearningRate (double _learningRate, double _b_learningRate)

  Sets the learning rate.
- void setInput (int _index, double _value)

  Sets the inputs to this neuron that is located in the first hidden layer.
- void propInputs (int _index, double _value)

  Sets the inputs to this neuron that can be located in any layer other than the first hidden layer.
- int calcOutput (int _layerHasReported)

  Calculates the output of the neuron by performing a weighed sum of all inputs to this neuron and activating the sum.
- void setError (double _value)

  Sets the error of the neuron in the first hidden layer that is to be propagated forward.
- double getError ()

  Allows accessing the error of this neuron.
- void updateWeights ()

Performs one iteration of learning, that is: it updates all the weights assigned to each input to this neuron.

- double doActivation (const double sum) const

  Performs the activation of the sum output of the neuron.
- double doActivationPrime (const double input) const

  Performs inverse activation on any input that is passed to this function.
- void setBackpropError (const double upstreamDeltaErrorSum)

  Sets the internal backprop error.
- double getOutput ()

  Requests the output of this neuron.
- double getSumOutput ()

  Requests the sum output of the neuron.
- double getWeights (int _inputIndex)

  Requests a specific weight.
- double getInitWeights (int _inputIndex)

  Requests a inital value of a specific weight.
- double getWeightChange ()

  Requests for overall change of all weights contained in this neuron.
- double getMaxWeight ()

  Requests for the maximum weights located in this neuron.
- double getMinWeight ()

  Requests for the minimum weights located in this neuron.
- double getSumWeight ()

  Requests for the total sum of weights located in this neuron.
- double getWeightDistance ()

  Requests the weight distance of all weighs in this neuron.
- int getnInputs ()

  Requests the total number of inputs to this neuron.
- void saveWeights ()

  Saves the temporal weight change of all weights in this neuron into a file.
- void printNeuron ()

  Prints on the console a full description of all weights, inputs and outputs for this neuron.
- void setWeight (int _index, double _weight)

  Sets the weights of the neuron.

3.4.1 Detailed Description

This is the class for creating neurons inside the Layer class.

This is the building block class of the network.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Neuron()

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

Constructor for the Neuron class: it initialises a neuron with specific number fo inputs to that neuron.

Parameters

_nInputs	

3.4.3 Member Function Documentation

3.4.3.1 calcOutput()

```
int Neuron::calcOutput (
            int _layerHasReported )
```

Calculates the output of the neuron by performing a weighed sum of all inputs to this neuron and activating the sum.

Parameters

_layerHasReported	boolean variable to indicate whether or not any neuron in this layer has reported exploding output

Returns

Returns a boolean to report whether or not this neuron has exploding output

3.4.3.2 doActivation()

```
double Neuron::doActivation (
            const double sum ) const  [inline]
```

Performs the activation of the sum output of the neuron.

Parameters

_sum	the weighted sum of all inputs

Returns

activation of the sum

3.4.3.3 doActivationPrime()

```
double Neuron::doActivationPrime (
            const double input ) const  [inline]
```

Performs inverse activation on any input that is passed to this function.

Parameters

_input	the input value

Returns

the inverse activation of the input

3.4.3.4 getError()

```
double Neuron::getError ( )
```

Allows accessing the error of this neuron.

Returns

the value of the error

3.4.3.5 getInitWeights()

```
double Neuron::getInitWeights (
            int _inputIndex )
```

Requests a inital value of a specific weight.

Parameters

_inputIndex	index of the input to which the weight is assigned

Returns

teh inital value of the weight

3.4.3.6 getMaxWeight()

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

Requests for the maximum weights located in this neuron.

Returns

Returns the max weight

3.4.3.7 getMinWeight()

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

Requests for the minimum weights located in this neuron.

Returns

Returns the min weight

3.4.3.8 getnInputs()

```
int Neuron::getnInputs ( )
```

Requests the total number of inputs to this neuron.

Returns

total number of inputs

3.4.3.9 getOutput()

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

Requests the output of this neuron.

Returns

the output of the neuron after activation

3.4.3.10 getSumOutput()

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

Requests the sum output of the neuron.

Returns

returns the sum output of the neuron before activaiton

3.4.3.11 getSumWeight()

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

Requests for the total sum of weights located in this neuron.

Returns

Returns the sum of weights

3.4.3.12 getWeightChange()

```
double Neuron::getWeightChange ( )
```

Requests for overall change of all weights contained in this neuron.

Returns

the overal weight change

3.4.3.13 getWeightDistance()

```
double Neuron::getWeightDistance ( )
```

Requests the weight distance of all weighs in this neuron.

Returns

returns the sqr of the total weight change in this neuron

3.4.3.14 getWeights()

```
double Neuron::getWeights (
            int _inputIndex )
```

Requests a specific weight.

Parameters

_inputIndex	index of the input to which the chosen weight is assigned

Returns

Returns the chosen weight

3.4.3.15 initNeuron()

```
void Neuron::initNeuron (
            int _neuronIndex,
            int _layerIndex,
            weightInitMethod _wim,
            biasInitMethod _bim,
            actMethod _am )
```

Initialises the neuron with the given methods for weight/bias initialisation and for activation function.

It also specifies the index of the neuron and the index of the layer that contains this neuron.

Parameters

_neuronIndex	The index of this neuron
_layerIndex	The index of the layer that contains this neuron
_wim	The method of initialising the weights, refer to weightInitMethod for more information
_bim	The method of initialising the biases, refer to biasInitMethod for more information
_am	The function used for activation of neurons, refer to actMethod for more information

3.4.3.16 propInputs()

```
void Neuron::propInputs (
            int _index,
            double _value )
```

Sets the inputs to this neuron that can be located in any layer other than the first hidden layer.

Parameters

_index	index of the input
_value	value of the input

3.4.3.17 setBackpropError()

```
void Neuron::setBackpropError (
            const double upstreamDeltaErrorSum )  [inline]
```

Sets the internal backprop error.

Parameters

_input	the input value

3.4.3.18 setError()

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

Sets the error of the neuron in the first hidden layer that is to be propagated forward.

Parameters

_value	value of the error

3.4.3.19 setInput()

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

Sets the inputs to this neuron that is located in the first hidden layer.

Parameters

_index	Index of the input
_value	Value of the input

3.4.3.20 setLearningRate()

```
void Neuron::setLearningRate (
            double _learningRate,
            double _b_learningRate )
```

Sets the learning rate.

Parameters

_learningRate	Sets the learning rate for this neuron.

3.4.3.21 setWeight()

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

Sets the weights of the neuron.

Parameters

_index	index of the weight
_weight	value of the weight

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

- Neuron.h

Index