Reference Manual

Generated by Doxygen 1.3-rc2

Tue Feb 18 23:08:34 2003

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Compound Index

2.1 Compound List

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

Chapter 3

Class Documentation

3.1 ActivationFunction Struct Reference

`#include <ActivationFunction.h>`

Public Attributes

- char * name
- double(* normal)(double, double)
- double(* derivate)(double, double)

3.1.1 Detailed Description

Neural network activation function.

Complete definition of a function and its derivate.

3.1.2 Member Data Documentation

3.1.2.1 double(* ActivationFunction::derivate)(double, double)

The derivative function, used for the backpropagation algorithm.

3.1.2.2 char* ActivationFunction::name

The symbolic name of the function.

3.1.2.3 double(* ActivationFunction::normal)(double, double)

Plain function itself. The first parameter gives the x-value, the second one is the theta value, taken from the neuron.

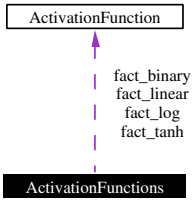
The documentation for this struct was generated from the following file:

- ActivationFunction.h

3.2 ActivationFunctions Class Reference

`#include <ActivationFunction.h>`

Collaboration diagram for ActivationFunctions:



Static Public Methods

- const ActivationFunction * resolveByName (const char *name)

Static Public Attributes

- const ActivationFunction fact_tanh
- const ActivationFunction fact_log
- const ActivationFunction fact_linear
- const ActivationFunction fact_binary

3.2.1 Detailed Description

List of activation functions that are available to the perceptron network.

3.2.2 Member Function Documentation

3.2.2.1 const ActivationFunction * ActivationFunctions::resolveByName (const char * *name*) [static]

Resolve an activation function by symbolic name.

Parameters:

name The function name, as given in the function structure.

Returns:

NULL on failure, pointer to structure on success.

3.2.3 Member Data Documentation

3.2.3.1 const ActivationFunction ActivationFunctions::fact_binary [static]

Initial value:

```
 {
        "binary",
        fact_binary_normal,
        fact_binary_derivate,
}
```

$$f_{act}(x, \Theta) = \begin{cases} 1.0 & x \geq \Theta \\ -1.0 & x < \Theta \end{cases}$$

3.2.3.2 const ActivationFunction ActivationFunctions::fact_linear [static]

Initial value:

```
 {
        "linear",
        fact_linear_normal,
        fact_linear_derivate,
}
```

$$f_{act}(x, \Theta) = x - \Theta$$

3.2.3.3 const ActivationFunction ActivationFunctions::fact_log [static]

Initial value:

```
 {
        "log",
        fact_log_normal,
        fact_log_derivate,
}
```

$$f_{act}(x, \Theta) = \frac{1}{1+e^{-(x-\Theta)}}$$

3.2.3.4 const ActivationFunction ActivationFunctions::fact_tanh [static]

Initial value:

```
 {
        "tanh",
        fact_tanh_normal,
        fact_tanh_derivate,
}
```

$$f_{act}(x, \Theta) = tanh(x - \Theta)$$

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

- ActivationFunction.h
- ActivationFunction.cpp

3.3 au_filehdr Struct Reference

`#include <AUFile.h>`

Public Methods

- uint32_t magic __attribute__ ((packed))
- uint32_t offset __attribute__ ((packed))
- uint32_t data_size __attribute__ ((packed))
- uint32_t encoding __attribute__ ((packed))
- uint32_t sample_rate __attribute__ ((packed))
- uint32_t channels __attribute__ ((packed))

3.3.1 Detailed Description

Struct of the AU file header

Note that we explicitly pack the structure elements to allow raw reads of the structure from the file and vice versa. (See http://gcc.gnu.org/onlinedocs/gcc-2.95.3/gcc_4.html#SEC90)

3.3.2 Member Function Documentation

3.3.2.1 uint32_t channels au_filehdr::__attribute__ ((packed))

Number of interleaved channels (mono = 1, stereo = 2)

3.3.2.2 uint32_t sample_rate au_filehdr::__attribute__ ((packed))

Samples per second

3.3.2.3 uint32_t encoding au_filehdr::__attribute__ ((packed))

Data encoding

3.3.2.4 uint32_t data_size au_filehdr::__attribute__ ((packed))

Audio data length in bytes

3.3.2.5 uint32_t offset au_filehdr::__attribute__ ((packed))

Absolute byte offset into file to start of audio data

3.3.2.6 uint32_t magic au_filehdr::__attribute__ ((packed))

The magic number (".snd")

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

- AUFile.h

3.4 AU_info_data Struct Reference

`#include <AUFile.h>`

Public Attributes

- unsigned char * data
- unsigned int size

3.4.1 Detailed Description

Structure for the additional, user-supplied header information

3.4.2 Member Data Documentation

3.4.2.1 unsigned char* AU_info_data::data

Byte array of data

3.4.2.2 unsigned int AU_info_data::size

Length of user data

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

- AUFile.h

3.5 AudioPreprocessing Class Reference

`#include <AudioPreprocessing.h>`

Public Methods

- AudioPreprocessing (void)
- ~AudioPreprocessing (void)
- AudioPreprocessing (AudioPreprocessing &source)
- AudioPreprocessing (double interleaving, int factor, int noWindows, int noData)
- void save (fstream &fs) const
- bool load (fstream &fs)
- void createData (AUFile *auData)
- const vector< double > & getWindows (void) const
- const vector< double > & getFFTEnergies (void) const
- void setInterleaving (double interleaving)
- double getInterleaving (void) const
- void setFactor (int factor)
- double getFactor (void) const
- void setNoWindows (int noWindows)
- int getNoWindows (void) const

Private Methods

- double sum (void)
- int sizeOfFirstWindow (void)
- void createWindows (void)
- void scaleWindows (void)
- void createFFTEnergies (void)
- int getFactorIndex (double value) const

Private Attributes

- double interleaving
- double factor
- int noWindows
- int noData
- double * data
- vector< double > window
- vector< double > energy

3.5.1 Detailed Description

The main audio processing class.

Used to convert raw audio data created by the AUFile class to a small vector. The vector holds multiple doubles that correspond to windowed energy levels in the frequency band of the input sample. The windows the vector elements are created from overlap with each other.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 AudioPreprocessing::AudioPreprocessing (void)

Main constructor.

3.5.2.2 AudioPreprocessing::~AudioPreprocessing (void)

Main destructor.

3.5.2.3 AudioPreprocessing::AudioPreprocessing (AudioPreprocessing & *source*)

Copy constructor.

Parameters:

source The source object that will be copied.

3.5.2.4 AudioPreprocessing::AudioPreprocessing (double *interleaving*, int *factor*, int *noWindows*, int *noData*)

Main constructor.

Parameters:

interleaving The percentage of interleaving (a value between zero and one).

factor Factor for multiplication of the predecessor window.

noWindows Number of windows which should be created.

noData Size of the audio data array.

3.5.3 Member Function Documentation

3.5.3.1 void AudioPreprocessing::createData (AUFile * *auData*)

Convert raw audio data to windowed vector.

The conversion is done in multiple steps:

1. Obtain the raw audio data
2. Apply fast fourier transformation (FFT)
3. Create windowed vector
4. Scale the vector

Parameters:

auData The raw audio data used as input.

3.5.3.2 void AudioPreprocessing::createFFTEnergies (void) [private]

Generate the FFT energy vector.

3.5.3.3 void AudioPreprocessing::createWindows (void) [private]

Compute the windowing of the audio data.

3.5.3.4 double AudioPreprocessing::getFactor (void) const

Getter for the enlargement factor.

Returns:
The currently used factor.

3.5.3.5 int AudioPreprocessing::getFactorIndex (double *value*) const [private]

Small value to index conversion function.

Parameters:
value The factor value to be converted.

Returns:
-1 on failure, factor index on success.

3.5.3.6 const vector< double > & AudioPreprocessing::getFFTEnergies (void) const

Getter for the FFT frequency energy vector.

Returns:
The FFT energy vector.

3.5.3.7 double AudioPreprocessing::getInterleaving (void) const

Getter for the interleaving factor.

Returns:
The current interleaving perceptage.

3.5.3.8 int AudioPreprocessing::getNoWindows (void) const

Getter function for the number of windows

Returns:
noWindows Number of Windows

3.5.3.9 const vector< double > & AudioPreprocessing::getWindows (void) const

Getter for the data array prepared for the neuronal net

Returns:
The data vector containing the individual windowed values.

3.5.3.10 bool AudioPreprocessing::load (fstream & *fs*)

Loader for preprocessor settings.

Parameters:
fs Stream the settings will be read from.

Returns:
True on success, false on failure.

3.5.3.11 void AudioPreprocessing::save (fstream & *fs*) const

Save method for the preprocessor settings.

Parameters:
fs Stream the settings are saved to.

3.5.3.12 void AudioPreprocessing::scaleWindows (void) [private]

Scale the audio data to values between -1 and 1.

3.5.3.13 void AudioPreprocessing::setFactor (int *factor*)

Setter function for the factor to enlargement factor.

Parameters:
factor Multiplier of the size of the predecessor window.

3.5.3.14 void AudioPreprocessing::setInterleaving (double *interleaving*)

Setter function for the interleaving factor.

Parameters:
interleaving The percentage of interleaving (a value between zero and one).

3.5.3.15 void AudioPreprocessing::setNoWindows (int *noWindows*)

Setter function for the number of windows

Parameters:
noWindows Number of Windows

3.5.3.16 int AudioPreprocessing::sizeOfFirstWindow (void) [private]

Compute the size of the first window.

Returns:
The size of the first window.

3.5.3.17 double AudioPreprocessing::sum (void) [private]

Compute the sum for the calculation of the first window's size

The sum is defined on page 15 of the Projekt-NN.ps project file.

3.5.4 Member Data Documentation

3.5.4.1 double* AudioPreprocessing::data [private]

Pointer to the beginning of the audio data array.

3.5.4.2 vector<double> AudioPreprocessing::energy [private]

Frequency based energy levels, as result of the amplitudes of the FFT real and imaginary parts.

3.5.4.3 double AudioPreprocessing::factor [private]

Factor to be multiplied by the size of the predecessor window in order to enlarge the actual windowsize.

3.5.4.4 double AudioPreprocessing::interleaving [private]

Percentage of interleaving (a value between zero and one).

3.5.4.5 int AudioPreprocessing::noData [private]

Size of the double array containing the audio values.

3.5.4.6 int AudioPreprocessing::noWindows [private]

Overall number of available windows (vector size).

3.5.4.7 vector<double> AudioPreprocessing::window [private]

Contains the data vector extracted from the audio stream

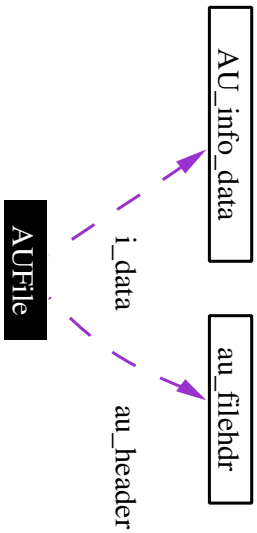
The documentation for this class was generated from the following files:

- AudioPreprocessing.h
- AudioPreprocessing.cpp

3.6 AUFile Class Reference

`#include <AUFile.h>`

Collaboration diagram for AUFile:



Public Methods

- AUFile (void)
- ~AUFile (void)
- AUFile (AUFile &source)
- bool open (const char *filename)
- bool write (const char *filename)
- bool record (const char *filename, unsigned int duration)
- bool play (void) const
- void normalize (void)
- bool cutSampleRange (unsigned int length)
- vector< double > & getData (void)
- unsigned int getSize (void)
- AU_info_data * getInformation (void)
- bool setInformation (unsigned int size, unsigned char *data)
- void setSize (unsigned int size)
- const char * getFilename (void)

Private Attributes

- char * filename
- au_filehdr au_header
- vector< double > au_data
- AU_info_data i_data
- bool if_open

Friends

- class GUIMainImpl

3.6.1 Detailed Description

One AU format sound file. Represents the entire file without information loss and provides means of changing each part of the file.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 AUFile::AUFile (void)

Standard constructor.

3.6.2.2 AUFile::~AUFile (void)

Standard destructor.

3.6.2.3 AUFile::AUFile (AUFile & *source*)

Copy constructor.

3.6.3 Member Function Documentation

3.6.3.1 bool AUFile::cutSampleRange (unsigned int *length*)

Shrink an AU file to the usable samplerange.

Parameters:
length Length of the samplerange

Returns:
true if successful, false if failed

3.6.3.2 vector< double > & AUFile::getData (void)

Getter for the audio data stream

3.6.3.3 const char * AUFile::getFilename (void)

Getter for the filename.

Returns:
filename of the au-file this object contains.

3.6.3.4 AU_info_data * AUFile::getInformation (void)

Getter for the additional header information

3.6.3.5 unsigned int AUFile::getSize (void)

Getter for the audio stream size

3.6.3.6 void AUFile::normalize (void)

Normalize the audio stream data.

3.6.3.7 bool AUFile::open (const char * *filename*)

Open an AU file and write its content to the class.

Parameters:

filename Filename of the AU file which should be opened.

Returns:

True on success, false on failure.

3.6.3.8 bool AUFile::play (void) const

Play the AU file.

Returns:

True on success, false on failure.

3.6.3.9 bool AUFile::record (const char * *filename*, unsigned int *duration*)

Record a new AU file

Parameters:

filename Filename the data is written to.

duration Length of recording in seconds. Must be in the range of 1 to 20.

Returns:

True on success, false on failure.

3.6.3.10 bool AUFile::setInformation (unsigned int *size*, unsigned char * *data*)

Setter for the additional header information.

Parameters:

size The byte-length of the new information.

data The byte array data.

Returns:
false in case the information could not be set, true when the information have been successfully updated.

3.6.3.11 void AUFile::setSize (unsigned int *size*)

Setter for audio data stream size.

Parameters:
size New size of the stream

3.6.3.12 bool AUFile::write (const char * *filename*)

Write the class data to an AU file.

Parameters:
filename Filename the data is written to.

Returns:
True in case of success, false when failed.

3.6.4 Member Data Documentation

3.6.4.1 vector<double> AUFile::au_data [private]

The audio data stream of the au-file.

3.6.4.2 au_filehdr AUFile::au_header [private]

The header of the au-file.

3.6.4.3 char* AUFile::filename [private]

Filename when loaded from file or NULL when new object.

3.6.4.4 AU_info_data AUFile::i_data [private]

Additional information which can be added at the end of the header

3.6.4.5 bool AUFile::if_open [private]

Flag, which is set to true once a file has been loaded. Otherwise, that is before any data has been loaded, it is set to false.

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

- AUFile.h
- AUFile.cpp

3.7 Graph Class Reference

`#include <Graph.h>`

Public Methods

- Graph (QWidget *parent=0, const char *name=0)
- ~Graph (void)
- void setTitle (const char *title)
- void setDescriptions (const vector< const char * > &desc)
- void setDisplayType (GraphDisplayType type)
- void clear (void)
- void setValues (const vector< vector< double > > &plots)
- void addValue (const vector< double > &plotitem)
- void setDisplaySize (unsigned int display_size)
- unsigned int getDisplaySize (void) const
- void setColor (const QColor &col)
- void setColor (const vector< QColor > &colors)
- void setMax (const double maximum)
- void setMargins (unsigned int margin_bars=2, unsigned int margin_left=4, unsigned int margin_right=4, unsigned int margin_up=4, unsigned int margin_down=4)

Protected Methods

- void paintEvent (QPaintEvent *ev)

Private Methods

- void plotFunction (const vector< double > &plot, const QColor &col, const vector< const char * > &desc)
- const QColor & selectColor (const unsigned int idx) const

Private Attributes

- GraphDisplayType type
- double max
- vector< vector< double > > plots
- vector< QColor > colors
- vector< const char * > desc
- const char * title
- unsigned int display_size
- QPainter * painter
- unsigned int margin_left
- unsigned int margin_right
- unsigned int margin_down
- unsigned int margin_up
- unsigned int margin_bars

3.7.1 Detailed Description

Function plotting widget.

Shows a number of discrete input vectors as multiple function plots within one widget. Rather unflexible and trimmed for use within the neural network result GUI part. Function plots can plot only positive values, while Barchart and Waveform objects can plot negatives, too.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Graph::Graph (QWidget * *parent* = 0, const char * *name* = 0)

Default QT-style constructor.

Parameters:

parent The QT parent widget.

name Optional name of the widget.

3.7.2.2 Graph::~Graph (void)

Default destructor.

3.7.3 Member Function Documentation

3.7.3.1 void Graph::addValue (const vector< double > & *plotitem*)

Add a plot item to the end of the current plot content.

Parameters:

plotitem A single plot item to be added.

3.7.3.2 void Graph::clear (void)

Clear the plot data and associated colors.

3.7.3.3 unsigned int Graph::getDisplaySize (void) const

Get the current amount to be plotted.

Note: Only applicable for Function type plots.

Returns:

Number of fixed items plotted.

3.7.3.4 void Graph::paintEvent (QPaintEvent * *ev*) [protected]

QT paintEvent method, used to redraw the modified screen content.

Parameters:
ev Event.

3.7.3.5 void Graph::plotFunction (const vector< double > & *plot*, const QColor & *col*, const vector< const char * > & *desc*) [private]

Plot a single graph.

Parameters:
plot Plot data to display.

col Color to display plot with.

desc (Optional) description to be displayed with each value of the graph. Only used for Barchart and BarchartPositive type of graphs.

3.7.3.6 const QColor & Graph::selectColor (const unsigned int *idx*) const [private]

Select the appropiate color.

Parameters:
idx The index of the graph that should be displayed.

Returns:
The color that was set for the graph.

3.7.3.7 void Graph::setColor (const vector< QColor > & *colors*)

Setter for individual plot colors.

Parameters:
colors Vector of individual colors for each plot.

3.7.3.8 void Graph::setColor (const QColor & *col*)

Setter for the default plot color.

Parameters:
col Default color to use in the chart.

3.7.3.9 void Graph::setDescriptions (const vector< const char * > & *desc*)

Set the bar descriptions.

The placement and meaning of the descriptions are interpreted according to the graph type. For Waveform and Function type graphs the text is displayed at the right border, after each graph trail. The array of descriptions refers to one description per graph. For Barchart and BarchartPositive type of graphs however, the descriptions refer to single bars and will be displayed right under each bar.

Parameters:

desc Ordered description strings.

3.7.3.10 void Graph::setDisplaySize (unsigned int *display_size*)

Set a fixed amount of values to plot.

Note: Only applicable for Function plots.

When more than display size items are given to plot, only the last display size items are shown, the first are discarded.

Parameters:

display_size Number of plot entries to display. Zero means automatic scaling.

3.7.3.11 void Graph::setDisplayType (GraphDisplayType *type*)

Set the display type of the widget.

Parameters:

type The visualization type.

3.7.3.12 void Graph::setMargins (unsigned int *margin_bars* = 2, unsigned int *margin_left* = 4, unsigned int *margin_right* = 4, unsigned int *margin_up* = 4, unsigned int *margin_down* = 4)

Set the margins to be used for the chart.

All margins are specified as pixels.

Parameters:

margin_bars The margin between individual bars within a Barchart type plot. For all other types, this is ignored.

margin_left Left border.

margin_right Right border.

margin_up Upper border.

margin_down Downward border.

3.7.3.13 void Graph::setMax (const double *maximum*) [inline]

Setter for the maximum display value. For the Waveform display, the negative value is used as minimum.

Parameters:
maximum The maximal displayed value.

3.7.3.14 void Graph::setTitle (const char * *title*)

Set the graph title.

Parameters:
title Title string.

3.7.3.15 void Graph::setValues (const vector< vector< double > > & *plots*)

Setter for the function plot content.

Parameters:
plots Vector of (possibly negative) values. Note that all plots have to be of the same element count.

3.7.4 Member Data Documentation

3.7.4.1 vector<QColor> Graph::colors [private]

Respective color for each plot. When .size() is zero, the default color is used (black).

3.7.4.2 vector<const char *> Graph::desc [private]

Descriptions for individual bars, used only with Barchart and BarchartPositive type of graphs.

3.7.4.3 unsigned int Graph::display_size [private]

Amount of plot items to display (only used for Function plots) Zero means scale appropiatly.

3.7.4.4 unsigned int Graph::margin_bars [private]

Margin between individual bars (only Barchart types).

3.7.4.5 unsigned int Graph::margin_down [private]

Margin to the downward widget border.

3.7.4.6 unsigned int Graph::margin_left [private]

Margin to the left widget border.

3.7.4.7 unsigned int Graph::margin_right [private]

Margin to the right widget border.

3.7.4.8 unsigned int Graph::margin_up [private]

Margin to the upward widget border.

3.7.4.9 double Graph::max [private]

The maximum displayable value. Used for all three chart types.

3.7.4.10 QPainter* Graph::painter [private]

The QT painter needed to (re-) draw the canvas. Initialized once upon object construction and used from there.

3.7.4.11 vector<vector <double> > Graph::plots [private]

The values to be displayed. Can be negative when negative is true.

3.7.4.12 const char* Graph::title [private]

The overall plot title. Once set through the constructor.

3.7.4.13 GraphDisplayType Graph::type [private]

Type of display used to visualize data.

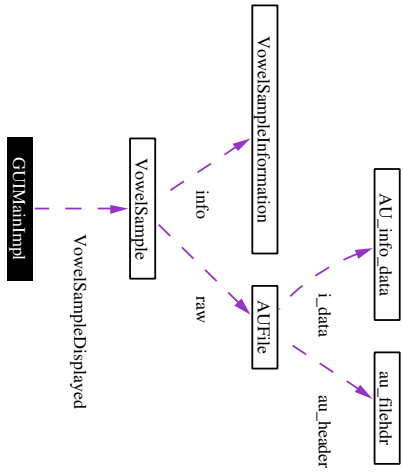
The documentation for this class was generated from the following files:

- Graph.h
- Graph.cpp

3.8 GUIMainImpl Class Reference

`#include <GUIMainImpl.h>`

Collaboration diagram for GUIMainImpl:



Public Methods

- GUIMainImpl (QWidget *parent=0, const char *name=0)
- void openAboutForm ()
- void AudioDirViewFolderSelected (QString *file)
- void AudioClassifySamplePressed (void)
- void AudioRecordPressed ()
- void AudioRecordFileSelectPressed ()
- void AudioPlayPressed ()
- void AudioProcessingInterleavingChanged (int interleaving)
- void AudioProcessingFactorChanged (int factor)
- void AudioNumberOfWindowsChanged (int nowindows)
- void SetManagerTrainingSetNewSetPressed ()
- void SetManagerTrainingSetLoadSet ()
- void SetManagerTrainingSetSaveSetPressed ()
- void SetManagerTrainingSetNameChanged (const QString &setname)
- void SetManagerAddToTrainingPressed ()
- void SetManagerRemoveFromTrainingPressed ()
- void SetManagerMoveFromTrainingToTestingPressed ()
- void SetManagerTestingSetNewSetPressed ()
- void SetManagerTestingSetLoadSet ()
- void SetManagerTestingSetSaveSetPressed ()
- void SetManagerTestingSetNameChanged (const QString &setname)
- void SetManagerAddToTestPressed ()
- void SetManagerRemoveFromTestPressed ()
- void SetManagerMoveFromTestingToTrainingPressed ()
- void NeuralNetGenerateSystemPressed (void)

- void NeuralNetLoadSystemPressed ()
- void NeuralNetSaveSystemPressed ()
- void NeuralNetNetwork0NumberOfLayersChanged (int layers)
- void NeuralNetNetwork0ChangeLayerChanged (int layer)
- void NeuralNetNetwork0NumberOfNeuronsChanged (int neurons)
- void NeuralNetNetwork1NumberOfLayersChanged (int layers)
- void NeuralNetNetwork1ChangeLayerChanged (int layer)
- void NeuralNetNetwork1NumberOfNeuronsChanged (int neurons)
- void NeuralNetNetwork2NumberOfLayersChanged (int layers)
- void NeuralNetNetwork2ChangeLayerChanged (int layer)
- void NeuralNetNetwork2NumberOfNeuronsChanged (int neurons)
- void startLearningPressed (void)
- void initializeOutputWidgetsClassify (void)
- void initializeOutputWidgetsLearning (void)
- void OutputDisplayWeightMatrix (void)
- void displayClassification (const VowelClassification &vclass)
- void displayWeightMatrix (void)
- void OutputMatrixLayer1SetValue (int value)
- void OutputMatrixLayer0SetValue (int value)
- void setOutputLayerSelectorMaximum ()

Private Slots

- void learnEpoch (void)

Private Methods

- double getMaximum (const vector< double > &val) const
- void setLearnResultPercentage (void)
- void updateAudioSettingsChanged (void)
- void reloadSets (void)

Static Private Methods

- void reloadSetsProgressHandler (void *user)

Private Attributes

- VowelSample * VowelSampleDisplayed
- vector< unsigned int > initnet [3]
- unsigned int learn_epochs
- char ostr_all_train [64]
- char ostr_net0_train [64]
- char ostr_net1_train [64]
- char ostr_net2_train [64]
- char ostr_all_test [64]
- char ostr_net0_test [64]
- char ostr_net1_test [64]

- char ostr_net2_test [64]
- bool audio_settings_changed
- unsigned int reload_count
- QProgressDialog * reimport_progress

3.8.1 Detailed Description

Main GUI Widget.

Constructs the main window application. All standard widgets set with QT-Designer are implemented in this class.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 GUIMainImpl::GUIMainImpl (QWidget * *parent* = 0, const char * *name* = 0)

Main Constructor for GUIMainImpl.

3.8.3 Member Function Documentation

3.8.3.1 void GUIMainImpl::AudioClassifySamplePressed (void)

Start the classify process for a single file.

3.8.3.2 void GUIMainImpl::AudioDirViewFolderSelected (QString * *file*)

Actualize the file-information widgets, the waveform-widget and the windowedform-widget is a new file is selected or something like interleaving, factor or the nuber of windows changed.

Parameters:
file selected file

3.8.3.3 void GUIMainImpl::AudioNumberOfWindowsChanged (int *nowindows*)

Update the vowelmanager-prep_template and actualize the drawings using AudioDirViewFolderSelected().

Parameters:
nowindows number of windows

3.8.3.4 void GUIMainImpl::AudioPlayPressed ()

Play the file selected in audiodirview

3.8.3.5 void GUIMainImpl::AudioProcessingFactorChanged (int *factor*)

Update the vowelmanager-prep_template and actualize the drawings using AudioDirViewFolderSelected().

Parameters:
factor new factor, set as an index number.

3.8.3.6 void GUIMainImpl::AudioProcessingInterleavingChanged (int *interleaving*)

Update the vowelmanager-prep_template and actualize the drawings using AudioDirViewFolderSelected().

Parameters:
interleaving new value for interleaving

3.8.3.7 void GUIMainImpl::AudioRecordFileSelectPressed ()

Open a savefiledialog, so the usercan select a filename.

3.8.3.8 void GUIMainImpl::AudioRecordPressed ()

Record a new file using the informations given in the audiorecord-groupbox.

3.8.3.9 void GUIMainImpl::displayClassification (const VowelClassification & *vclass*)

Display the results of a single-vowel classification within the output tab.

Parameters:
vclass The results to be displayed.

3.8.3.10 void GUIMainImpl::displayWeightMatrix (void)

Display the correct WeightMatrix layer from the selected network.

3.8.3.11 double GUIMainImpl::getMaximum (const vector< double > & *val*) const [private]

Find the absolute maximum of all values within a vector.

Parameters:
val The vector to be skimmed.

Returns:
The absolute maximum value within *val*.

3.8.3.12 void GUIMainImpl::initializeOutputWidgetsClassify (void)

Initialize the output widgets for use with single vowel classification.

3.8.3.13 void GUIMainImpl::initializeOutputWidgetsLearning (void)

Initialize the output widgets for use with the learning process.

3.8.3.14 void GUIMainImpl::learnEpoch (void) [private, slot]

Learn one epoch and update widgets afterwards. Called due to the idle timer.

3.8.3.15 void GUIMainImpl::NeuralNetGenerateSystemPressed (void)

Generate a new vowel classifier from the three individual network descriptions.

3.8.3.16 void GUIMainImpl::NeuralNetLoadSystemPressed ()

Open a openfiledialog and load a vowel classifier from a given filename.

3.8.3.17 void GUIMainImpl::NeuralNetNetwork0ChangeLayerChanged (int *layer*)

Update the numberofneurons-input, so the correct information is shown when then number of the layer to change is changed in network0.

Parameters:
layer number of active layer

3.8.3.18 void GUIMainImpl::NeuralNetNetwork0NumberOfLayersChanged (int *layers*)

Update the initnet and set the maxvalue of the changelayer-input when the number of layers is changed in network0.

Parameters:
layers number of layers

3.8.3.19 void GUIMainImpl::NeuralNetNetwork0NumberOfNeuronsChanged (int *neurons*)

Set the numbers of neurons in network0.

Parameters:
neurons number of neurons

3.8.3.20 void GUIMainImpl::NeuralNetNetwork1ChangeLayerChanged (int *layer*)

Update the numberofneurons-input, so the correct information is shown when then number of the layer to change is changed in network1.

Parameters:
layer number of active layer

3.8.3.21 void GUIMainImpl::NeuralNetNetwork1NumberOfLayersChanged (int *layers*)

Update the initnet and set the maxvalue of the changelayer-input when the number of layers is changed in network1.

Parameters:
layers number of layers

3.8.3.22 void GUIMainImpl::NeuralNetNetwork1NumberOfNeuronsChanged (int *neurons*)

Set the numbers of neurons in network1.

Parameters:
neurons number of neurons

3.8.3.23 void GUIMainImpl::NeuralNetNetwork2ChangeLayerChanged (int *layer*)

Update the numberofneurons-input, so the correct information is shown when then number of the layer to change is changed in network2.

Parameters:
layer number of active layer

3.8.3.24 void GUIMainImpl::NeuralNetNetwork2NumberOfLayersChanged (int *layers*)

Update the initnet and set the maxvalue of the changelayer-input when the number of layers is changed in network2.

Parameters:
layers number of layers

3.8.3.25 void GUIMainImpl::NeuralNetNetwork2NumberOfNeuronsChanged (int *neurons*)

Set the numbers of neurons in network2.

Parameters:
neurons number of neurons

3.8.3.26 void GUIMainImpl::NeuralNetSaveSystemPressed ()

Open a savefiledialog and save a vowel classifier to a given filename.

3.8.3.27 void GUIMainImpl::openAboutForm ()

opens the AboutForm-Dialog with some informations about this program. The dialog itself is made with QT-Designer.

3.8.3.28 void GUIMainImpl::OutputDisplayWeightMatrix (void)

Re-display the weight matrix.

3.8.3.29 void GUIMainImpl::OutputMatrixLayer0SetValue (int *value*)

Adjust the other value for the layernumbers, between which the weightmatrix should be shown, so that the difference is always 1

3.8.3.30 void GUIMainImpl::OutputMatrixLayer1SetValue (int *value*)

Adjust the other value for the layernumbers, between which the weightmatrix should be shown, so that the difference is always 1

3.8.3.31 void GUIMainImpl::reloadSets (void) [private]

Reload both the training and test sets by saving them to a temporary file and restoring the set from that file. Do this to reprocess all files with the updated audio preprocessing template.

3.8.3.32 void GUIMainImpl::reloadSetsProgressHandler (void * *user*) [static, private]

Handle the progress bar display while reloading all samples.

Parameters:

user The this pointer to the GUIMainImpl main object.

3.8.3.33 void GUIMainImpl::setLearnResultPercentage (void) [private]

Add the final result percentages to all output graph widgets.

3.8.3.34 void GUIMainImpl::SetManagerAddToTestPressed ()

add a selected file to the testingset.

3.8.3.35 void GUIMainImpl::SetManagerAddToTrainingPressed ()

add a selected file to the trainingset.

3.8.3.36 void GUIMainImpl::SetManagerMoveFromTestingToTrainingPressed ()

move a selected file from testingset to the trainingset.

3.8.3.37 void GUIMainImpl::SetManagerMoveFromTrainingToTestingPressed ()

move a selected file from trainingset to the testset.

3.8.3.38 void GUIMainImpl::SetManagerRemoveFromTestPressed ()

remove a selected file from the testingset.

3.8.3.39 void GUIMainImpl::SetManagerRemoveFromTrainingPressed ()

remove a selected file from the trainingset.

3.8.3.40 void GUIMainImpl::SetManagerTestingSetLoadSet ()

Open an openfiledialog and load the selected testingset.

3.8.3.41 void GUIMainImpl::SetManagerTestingSetNameChanged (const QString & *setname*)

Update the vowelmanagerobject if the setname was changed

Parameters:
setname name of the set

3.8.3.42 void GUIMainImpl::SetManagerTestingSetNewSetPressed ()

Clear the testingset and reset setname.

3.8.3.43 void GUIMainImpl::SetManagerTestingSetSaveSetPressed ()

Open a savefiledialog and save the actual set.

3.8.3.44 void GUIMainImpl::SetManagerTrainingSetLoadSet ()

Open an openfiledialog and load the selected trainingset.

3.8.3.45 void GUIMainImpl::SetManagerTrainingSetNameChanged (const QString & *setname*)

Update the vowelmanagerobject if the setname was changed

Parameters:
setname name of the set

3.8.3.46 void GUIMainImpl::SetManagerTrainingSetNewSetPressed ()

Clear the trainingset and reset setname.

3.8.3.47 void GUIMainImpl::SetManagerTrainingSetSaveSetPressed ()

Open a savefiledialog and save the actual set.

3.8.3.48 void GUIMainImpl::setOutputLayerSelectorMaximum ()

Set maximum values for layerselection of the weightmatrix if another network has been selected.

3.8.3.49 void GUIMainImpl::startLearningPressed (void)

Initialize the learn settings and start the train process.

3.8.3.50 void GUIMainImpl::updateAudioSettingsChanged (void) [private]

Update the audio_settings_changed variable when necessary.
Check if there are samples loaded within the setmanager, and update the reload flag in case there are.

3.8.4 Member Data Documentation

3.8.4.1 bool GUIMainImpl::audio_settings_changed [private]

Used to show the audio preprocessing settings have been changed.

3.8.4.2 vector<unsigned int> GUIMainImpl::initnet[3] [private]

Contains the Initialization values for the Neural Network, needed by GenerateSystem

3.8.4.3 unsigned int GUIMainImpl::learn_epochs [private]

Number of epochs to learn.

3.8.4.4 char GUIMainImpl::ostr_all_test[64] [private]

Per object static-like string objects used for the final result percentage in the output window graph widgets.

3.8.4.5 char GUIMainImpl::ostr_all_train[64] [private]

Per object static-like string objects used for the final result percentage in the output window graph widgets.

3.8.4.6 char GUIMainImpl::ostr_net0_test[64] [private]

Per object static-like string objects used for the final result percentage in the output window graph widgets.

3.8.4.7 char GUIMainImpl::ostr_net0_train[64] [private]

Per object static-like string objects used for the final result percentage in the output window graph widgets.

3.8.4.8 char GUIMainImpl::ostr_net1_test[64] [private]

Per object static-like string objects used for the final result percentage in the output window graph widgets.

3.8.4.9 char GUIMainImpl::ostr_net1_train[64] [private]

Per object static-like string objects used for the final result percentage in the output window graph widgets.

3.8.4.10 char GUIMainImpl::ostr_net2_test[64] [private]

Per object static-like string objects used for the final result percentage in the output window graph widgets.

3.8.4.11 char GUIMainImpl::ostr_net2_train[64] [private]

Per object static-like string objects used for the final result percentage in the output window graph widgets.

3.8.4.12 QProgressDialog* GUIMainImpl::reimport_progress [private]

When a sample-processing related progressbar is displayed, this object pointer is used.

3.8.4.13 unsigned int GUIMainImpl::reload_count [private]

When reloading the samples due to changed audio settings, this keeps track of the samples imported and updates the reimport_progress progressbar object.

3.8.4.14 VowelSample* GUIMainImpl::VowelSampleDisplayed [private]

The vowel sample that is displayed right now in the audio tab, or NULL when there is none.

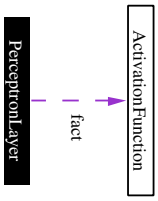
The documentation for this class was generated from the following files:

- GUIMainImpl.h
- GUIMainImpl.cpp

3.9 PerceptronLayer Class Reference

`#include <PerceptronLayer.h>`

Collaboration diagram for PerceptronLayer:



Public Methods

- PerceptronLayer (unsigned int neuron_count, const ActivationFunction *fact)
- ~PerceptronLayer (void)
- PerceptronLayer (PerceptronLayer &source)
- void randomizeParameters (PerceptronLayer *succ, const RandomFunction *weight_func, const RandomFunction *theta_func)
- void resetDiffs (void)
- void setActivationFunction (const ActivationFunction *fact)
- const ActivationFunction * getActivationFunction (void) const
- void propagate (PerceptronLayer *pred)
- void backpropagate (PerceptronLayer *succ, vector< double > &output_optimal, double opt_-tolerance)
- void postprocess (PerceptronLayer *succ, double epsilon, double weight_decay, double momterm)
- void update (void)

Public Attributes

- PerceptronLayerType type
- vector< PerceptronNeuron * > neurons

Protected Attributes

- const ActivationFunction * fact

3.9.1 Detailed Description

One layer within the perceptron network.

Most algorithm related calculation is done at the layer level.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 PerceptronLayer::PerceptronLayer (unsigned int *neuron_count*, const ActivationFunction * *fact*)

PerceptronLayer constructor, creating a layer with *neuron_count* number of neurons.

Parameters:

neuron_count Number of neurons within this layer.

fact Activation function to use for this layer.

3.9.2.2 PerceptronLayer::~PerceptronLayer (void)

PerceptronLayer destructor, mainly removing all the subneurons within this layer.

3.9.2.3 PerceptronLayer::PerceptronLayer (PerceptronLayer & *source*)

Copy constructor.

Parameters:

source Source object to be copied.

3.9.3 Member Function Documentation

3.9.3.1 void PerceptronLayer::backpropagate (PerceptronLayer * *succ*, vector< double > & *output_optimal*, double *opt_tolerance*)

Backpropagation algorithm.

Compute the delta value for all neurons within this layer.

Parameters:

succ Succeeding layer to this one, can be NULL.

output_optimal Optimal expected output for the last layer (output).

opt_tolerance The optimal difference tolerance parameter.

3.9.3.2 const ActivationFunction * PerceptronLayer::getActivationFunction (void) const

Getter for the activation function.

3.9.3.3 void PerceptronLayer::postprocess (PerceptronLayer * *succ*, double *epsilon*, double *weight_decay*, double *momterm*)

Postprocess algorithm for a single layer.

Parameters:

succ Successor layer to this one, or NULL if it is the last.

epsilon Learning parameter.

weight_decay Weight decay factor.

momterm Momentum term factor.

3.9.3.4 void PerceptronLayer::propagate (PerceptronLayer * *pred*)

Forward propagation algorithm. The input and output signals of this layer is calculated from the output signals of the *pred* layer.

Parameters:

pred the layer before this one, can be NULL.

3.9.3.5 void PerceptronLayer::randomizeParameters (PerceptronLayer * *succ*, const RandomFunction * *weight_func*, const RandomFunction * *theta_func*)

Network randomization functions.

Randomize variable network parameters: weightings and theta values. Use individual functions for greater customizeability.

Parameters:

succ Succeeding layer in network. Must be non-NULL.

weight_func Function to generate weighting parameters.

theta_func Function to generate theta values.

3.9.3.6 void PerceptronLayer::resetDiffs (void)

Reset functions for learned differences.

This function resets all parameters that are learned by the backpropagation and postprocess algorithms. It has to be called after an update has been made. This way, both online- and batch-learning can be implemented.

3.9.3.7 void PerceptronLayer::setActivationFunction (const ActivationFunction * *fact*)

Setter for the activation function.

Parameters:

fact Activation function to use for this layer.

3.9.3.8 void PerceptronLayer::update (void)

Update algorithm for a single layer.

3.9.4 Member Data Documentation

3.9.4.1 const ActivationFunction* PerceptronLayer::fact [protected]

Activation function and its derivate. Used for both propagation and backpropagation. Must be set and can be different for each layer.

3.9.4.2 vector<PerceptronNeuron *> PerceptronLayer::neurons

Every layer contains at least one neuron. This is the list of neurons within this layer.

3.9.4.3 PerceptronLayerType PerceptronLayer::type

Type of the layer within the network. The input layer is always the first, the output layer the last layer in the network. Every other layer must be a hidden layer.

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

- PerceptronLayer.h
- PerceptronLayer.cpp

3.10 PerceptronNetwork Class Reference

`#include <PerceptronNetwork.h>`

Public Methods

- PerceptronNetwork (vector< unsigned int > desc_layers, const char *network_name="unnamed", const ActivationFunction *fact=&ActivationFunctions::fact_tanh)
- ~PerceptronNetwork (void)
- PerceptronNetwork (PerceptronNetwork &source)
- void setActivationFunction (PerceptronLayerType type, const ActivationFunction *fact)
- void save (fstream &fs) const
- bool load (fstream &fs)
- void randomizeParameters (const RandomFunction *weight_func, const RandomFunction *theta_func)
- void setInput (vector< double > &in)
- void setOptimalOutput (vector< double > &optimal)
- vector< double > getOutput (void) const
- double errorTerm (void) const
- void resetDiffs (void)
- void propagate (void)
- void backpropagate (void)
- void postprocess (void)
- void update (void)
- void dumpNetworkGraph (const char *filename) const
- double getLearningParameter (void) const
- void setLearningParameter (double epsilon)
- double getOptimalTolerance (void) const
- void setOptimalTolerance (double tolerance)
- double getWeightDecayParameter (void) const
- void setWeightDecayParameter (double factor)
- double getMomentumTermParameter (void) const
- void setMomentumTermParameter (double factor)

Public Attributes

- const char * name
- vector< double > input
- vector< double > output

Protected Attributes

- vector< double > output_optimal
- vector< PerceptronLayer * > layers
- double epsilon
- double opt_tolerance
- double weight_decay
- double momentum_term

Friends

- class WeightMatrix

3.10.1 Detailed Description

One multilayer perceptron network.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 PerceptronNetwork::PerceptronNetwork (vector< unsigned int > *desc_layers*, const char * *network_name* = "unnamed", const ActivationFunction * *fact* = &ActivationFunctions::fact_tanh)

PerceptronNetwork constructor

Construct a multilayer perceptron network by layer description given through *desc_layers*. The vector size specifies the number of layers, the individual elements the number of neurons within its layer.

Parameters:

desc_layers Number of neurons within each layer, given the input layer as first and the output layer as last. Hence, the size of the vector must be at least two.

network_name Symbolic name of the network. Can be NULL. The name is pointer-copied, so we steal a pointer here.

fact Activation function to use for all neurons by default within the network. By default, its the tangens-hyperbolicus function. For individual layers, based one their type, the activation function can be changed using the changeActivation method.

3.10.2.2 PerceptronNetwork::~PerceptronNetwork (void)

PerceptronNetwork destructor

Remove all layers stored within the network.

3.10.2.3 PerceptronNetwork::PerceptronNetwork (PerceptronNetwork & *source*)

Copy constructor.

Parameters:

source Source object to be copied.

3.10.3 Member Function Documentation

3.10.3.1 void PerceptronNetwork::backpropagate (void)

Backpropagation algorithm for the entire network.

Calculate all delta error signals in every layer and their neurons. Every neuron must have a proper input/output signal assigned, and the output_optimal training target result is used for calculation.

3.10.3.2 void PerceptronNetwork::dumpNetworkGraph (const char * *filename*) const

Dump whole neural network as graph file in the GraphViz file format (www.graphviz.org).

Parameters:
filename Name of the file to write the graph data to.

3.10.3.3 double PerceptronNetwork::errorTerm (void) const

Calculate the errorterm for the network

The error value for the current output is calculated. The current optimal output must be given prior to calling this function.

The formula used is $E = \frac{1}{2} \sum_i \left(t_{p,i} - y_{p,i}^{\lambda,L} \right)^2$.

3.10.3.4 double PerceptronNetwork::getLearningParameter (void) const

Getter for the epsilon parameter.

3.10.3.5 double PerceptronNetwork::getMomentumTermParameter (void) const

Getter for the momentum term parameter.

3.10.3.6 double PerceptronNetwork::getOptimalTolerance (void) const

Getter for the optimal tolerance parameter.

3.10.3.7 vector< double > PerceptronNetwork::getOutput (void) const

Getter for the entire network output.

3.10.3.8 double PerceptronNetwork::getWeightDecayParameter (void) const

Getter for the weight decay parameter.

3.10.3.9 bool PerceptronNetwork::load (fstream & *fs*)

Load the entire network from stream *fs*.

Parameters:
fs Stream (input) to read the network from.

Returns:
True on success, false on failure.

3.10.3.10 void PerceptronNetwork::postprocess (void)

Postprocess algorithm for the entire network.

3.10.3.11 void PerceptronNetwork::propagate (void)

Propagation algorithm for the entire network.

The input levels to the network have to be set using the setInput() method, before. Afterwards the output of the network can be obtained using the getOutput() method.

3.10.3.12 void PerceptronNetwork::randomizeParameters (const RandomFunction * *weight_func*, const RandomFunction * *theta_func*)

Randomize the variable network parameters, weightings and theta values.

Parameters:

weight_func Function to randomize weighting values.

theta_func Function to randomize theta parameters.

3.10.3.13 void PerceptronNetwork::resetDiffs (void)

Reset all learned parameters.

Call after an update has been made.

3.10.3.14 void PerceptronNetwork::save (fstream & *fs*) const

Save the entire network as text into the stream *fs*.

Parameters:

fs Stream (output) to save the network to.

3.10.3.15 void PerceptronNetwork::setActivationFunction (PerceptronLayerType *type*, const ActivationFunction * *fact*)

Set the activation function for layers, based on their type.

Parameters:

type Type of layers that will use the new activation function.

fact Activation function to use.

3.10.3.16 void PerceptronNetwork::setInput (vector< double > & *in*)

Setter for the entire network input.

Parameters:

in Vector of input signal levels. The size must equal the input layer neuron count.

3.10.3.17 void PerceptronNetwork::setLearningParameter (double *epsilon*)

Setter for the epsilon parameter.

Parameters:
epsilon Learn parameter, in the range of 0.0 to 0.5.

3.10.3.18 void PerceptronNetwork::setMomentumTermParameter (double *factor*)

Setter for the momentum term parameter.

Parameters:
factor Value between 0.5 and 0.9.

3.10.3.19 void PerceptronNetwork::setOptimalOutput (vector< double > & *optimal*)

Setter for the optimal requested network output.

Must be used before any learning algorithm is called (backpropagate).

Parameters:
optimal Optimal network output for the current input.

3.10.3.20 void PerceptronNetwork::setOptimalTolerance (double *tolerance*)

Setter for the optimal tolerance parameter.

Parameters:
tolerance Value between 0.0 and 0.2.

3.10.3.21 void PerceptronNetwork::setWeightDecayParameter (double *factor*)

Setter for the weight decay parameter.

Parameters:
factor Value in the range of 0.00005 to 0.0001.

3.10.3.22 void PerceptronNetwork::update (void)

Update algorithm for the entire network.

3.10.4 Friends And Related Function Documentation

3.10.4.1 friend class WeightMatrix [friend]

The GUI visualization class is declared friend to allow access to individual layers.

3.10.5 Member Data Documentation

3.10.5.1 double PerceptronNetwork::epsilon [protected]

Generic learn parameter epsilon, should be between 0.05 and 0.5.

3.10.5.2 vector<double> PerceptronNetwork::input

Input vector, fed into the first layer of the network. The number of elements must equal the number of neurons within the input layer.

3.10.5.3 vector<PerceptronLayer *> PerceptronNetwork::layers [protected]

Left-to-right list of layers within the network. Must be at least two (one input and one output layer), but can be arbitrary large. More than four layers does not archive any improvement of the network capabilities, though.

3.10.5.4 double PerceptronNetwork::momentum_term [protected]

Momentum term parameter, should be between 0.5 and 0.9.

3.10.5.5 const char* PerceptronNetwork::name

Symbolic name of the network. Never used by the methods themselves, but convenient to use.

3.10.5.6 double PerceptronNetwork::opt_tolerance [protected]

Optimal tolerance parameter, should be between 0.0 and 0.2.

3.10.5.7 vector<double> PerceptronNetwork::output

Output vector, resulting from propagation through the entire network. The number of elements equals the number of output neurons within the network.

3.10.5.8 vector<double> PerceptronNetwork::output_optimal [protected]

Optimal expected output vector, used for the learning process. For simple propagation it is not needed.

3.10.5.9 double PerceptronNetwork::weight_decay [protected]

Weight decay parameter, should be between 0.005 and 0.03.

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

- PerceptronNetwork.h
- PerceptronNetwork.cpp

3.11 PerceptronNeuron Class Reference

`#include <PerceptronNeuron.h>`

Public Methods

- PerceptronNeuron (unsigned int in_layer_num)
- double getDelta (void)
- void setDelta (double newval)
- double getTheta (void)
- void setTheta (double newval)
- double getThetaDiff (void)
- void initializeWeightings (unsigned int succ_count)
- void resetWeights (void)
- void resetWeightDiffs (void)
- void resetDiffs (void)
- void postprocessWeight (PerceptronNeuron *succ, double epsilon, double weight_decay, double momterm)
- void postprocessTheta (double epsilon, double weight_decay, double momterm)
- void update (void)

Public Attributes

- unsigned int num
- double input
- double output
- vector< double > weight
- vector< double > weight_diff

Protected Attributes

- double delta
- double theta
- double theta_diff
- double theta_diff_last
- vector< double > weight_diff_last

3.11.1 Detailed Description

Represent one neuron within a layer of the neural network.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 PerceptronNeuron::PerceptronNeuron (unsigned int *in_layer_num*)

Main constructor.

Parameters:

in_layer_num number of neuron within its layer.

3.11.3 Member Function Documentation

3.11.3.1 double PerceptronNeuron::getDelta (void) [inline]

Getter function for the delta variable.

3.11.3.2 double PerceptronNeuron::getTheta (void) [inline]

Getter function for theta parameter.

3.11.3.3 double PerceptronNeuron::getThetaDiff (void) [inline]

Getter for the difference calculated for the theta parameter.

3.11.3.4 void PerceptronNeuron::initializeWeightings (unsigned int *succ_count*)

Initialize the weightings vectors used for each neuron to zero.

Parameters:

succ_count number of successor neurons to this neuron.

3.11.3.5 void PerceptronNeuron::postprocessTheta (double *epsilon*, double *weight_decay*, double *momterm*)

Theta postprocess algorithm. Assign the theta differences to this neuron.

Parameters:

epsilon Learning parameter.

weight_decay Weight decay factor.

momterm Momentum term factor.

3.11.3.6 void PerceptronNeuron::postprocessWeight (PerceptronNeuron * *succ*, double *epsilon*, double *weight_decay*, double *momterm*)

Weighting postprocess algorithm. Assign the weight differences to this neuron. Note that the assignment is done by adding the new delta value to the current one. Hence, both batch- and online learning can use this function. For online-learning the values must be resetted after each update using the resetWeightDiffs method. Also, only one weighting difference is calculated at a time. The computation of all differences is scheduled by PerceptronLayer::postprocess.

Parameters:

succ Successor neuron to the current neuron.

epsilon Learning parameter.

weight_decay Weight decay factor.

momterm Momentum term factor.

See also:

resetWeightDiffs ()

3.11.3.7 void PerceptronNeuron::resetDiffs (void)

Reset all learned differences.

3.11.3.8 void PerceptronNeuron::resetWeightDiffs (void)

Reset all weight deltas to zero.

3.11.3.9 void PerceptronNeuron::resetWeights (void)

Reset all weightings to zero.

3.11.3.10 void PerceptronNeuron::setDelta (double *newval*) [inline]

Setter function for the delta variable.

Parameters:
newval New value to be assigned to delta.

3.11.3.11 void PerceptronNeuron::setTheta (double *newval*) [inline]

Setter function for theta parameter.

3.11.3.12 void PerceptronNeuron::update (void)

Update algorithm. Update weightings and theta parameter by the values calculated in the postprocess step.

3.11.4 Member Data Documentation

3.11.4.1 double PerceptronNeuron::delta [protected]

Errorsignal, computed by the backpropagation algorithm. Used to compute the deltas for both the weightings and the sensitivity parameter

3.11.4.2 double PerceptronNeuron::input

Input signal level for this node, $net^{\lambda}_{p,i}$.

3.11.4.3 unsigned int PerceptronNeuron::num

Counter within the current layer (top = 0)

3.11.4.4 double PerceptronNeuron::output

Output signal level for this node, $y^{\lambda}_{p,i}$.

3.11.4.5 double PerceptronNeuron::theta [protected]

Sensitivity parameter to the activation function

3.11.4.6 double PerceptronNeuron::theta_diff [protected]

Delta-value for the sensitivity, calculated by the Processing algorithm, used to update theta within the Update algorithm

3.11.4.7 double PerceptronNeuron::theta_diff_last [protected]

The previous theta difference, used for momentum term calculation.

3.11.4.8 vector<double> PerceptronNeuron::weight

Weightings to neurons in the next layer. Hence, the size of the vector must equal the number of neurons in the next layer

3.11.4.9 vector<double> PerceptronNeuron::weight_diff

Delta-value for the individual weightings the node has to its successor nodes. Computed by the Processing algorithm

Note this is within the public space due to the from-file constructor of PerceptronNetwork, which push_back's zeroes here.

3.11.4.10 vector<double> PerceptronNeuron::weight_diff_last [protected]

The previous weight differences, used for momentum term.

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

- PerceptronNeuron.h
- PerceptronNeuron.cpp

3.12 RandomFunction Struct Reference

`#include <RandomFunction.h>`

Public Attributes

- const double(∗ func)(void ∗)
- void ∗ user

3.12.1 Detailed Description

Random number generation function definition.

3.12.2 Member Data Documentation

3.12.2.1 const double(∗ RandomFunction::func)(void ∗)

The function that generates a random number. The first parameter is the user parameter.

3.12.2.2 void∗ RandomFunction::user

The first parameter to the function.

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

- RandomFunction.h

3.13 RandomFunctions Class Reference

`#include <RandomFunction.h>`

Static Public Methods

- void rand_init (void)
- const double rand_normal (RandomInterval *range)

3.13.1 Detailed Description

Multiple random generator functions and one common initialization function.

3.13.2 Member Function Documentation

3.13.2.1 void RandomFunctions::rand_init (void) [static]

Initialize all random number generators.

3.13.2.2 const double RandomFunctions::rand_normal (RandomInterval * *range*) [static]

The canonical random number generator.

Generate one random floating point number within the given range.

Parameters:

range Range of numbers where the random one will lie in.

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

- RandomFunction.h
- RandomFunction.cpp

3.14 RandomInterval Struct Reference

`#include <RandomFunction.h>`

Public Attributes

- double low
- double high

3.14.1 Detailed Description

Range specification for random number generation.

3.14.2 Member Data Documentation

3.14.2.1 double RandomInterval::high

The high interval border.

3.14.2.2 double RandomInterval::low

The low interval border.

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

- RandomFunction.h

3.15 SplashScreen Class Reference

`#include <SplashScreen.h>`

Public Methods

- SplashScreen (const QPixmap &pm)
- void finish (QWidget *mainWin)
- void repaint (void)

Protected Methods

- void mousePressEvent (QMouseEvent *ev)

Private Attributes

- QPixmap pix

3.15.1 Detailed Description

Splashscreen class: Display an image for a short time and let the program continue afterwards.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 SplashScreen::SplashScreen (const QPixmap & *pm*)

Default constructor.

Parameters:
pm Pixmap of the image to be displayed.

3.15.3 Member Function Documentation

3.15.3.1 void SplashScreen::finish (QWidget * *mainWin*)

Logical destructor, has to be called when the splash screen has done its duty.

Parameters:
mainWin The main widget to be in control after the splash screen vanishes.

3.15.3.2 void SplashScreen::mousePressEvent (QMouseEvent * *ev*) `[protected]`

Close the splash image when the user clicks on it.

3.15.3.3 void SplashScreen::repaint (void)

The QT repaint method.

3.15.4 Member Data Documentation

3.15.4.1 QPixmap SplashScreen::pix [private]

Copy of the image passed to the constructor. Nothing fancy.

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

- SplashScreen.h
- SplashScreen.cpp

3.16 VowelClassification Struct Reference

`#include <VowelClassifier.h>`

Public Attributes

- vector< double > vstat
- vector< vector< double > > net_result
- VowelSampleType guessed_type

3.16.1 Detailed Description

Structure to return when a classification has been made.

3.16.2 Member Data Documentation

3.16.2.1 VowelSampleType VowelClassification::guessed_type

The type the sample was guessed as.

When the system is certain enough about the input vowel, the guessed_type will resemble the highest rating within the vstat array.

3.16.2.2 vector<vector<double> > VowelClassification::net_result

Individual network results.

net_result[0] holds the three results of the first network, net_result[1] and net_result[2] the other network results.

3.16.2.3 vector<double> VowelClassification::vstat

Vowel classification statistics for each vowel.

Tells a bit about how sure the expert system is about its guess. Can be anywhere between 0.0 (completely chaotic decision) and 1.0 (absolutely sure classification).

The index into the vector is the VowelSampleType, as in certainty_u = vstat[(VowelSampleType) U];

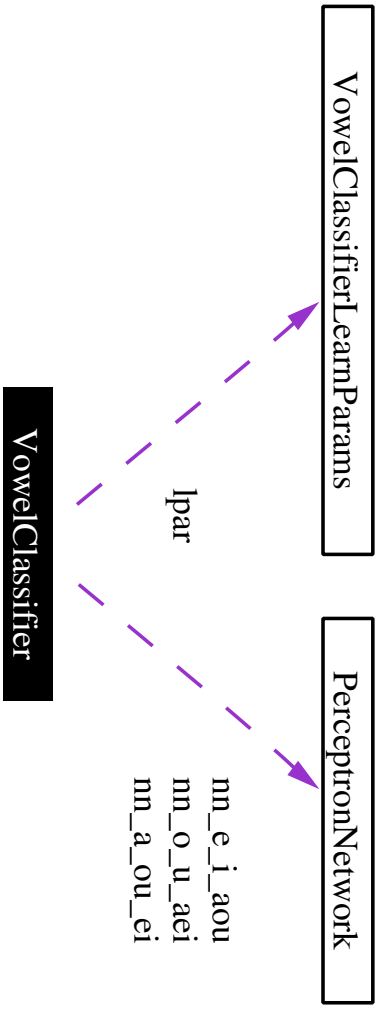
The documentation for this struct was generated from the following file:

- VowelClassifier.h

3.17 VowelClassifier Class Reference

`#include <VowelClassifier.h>`

Collaboration diagram for VowelClassifier:



Public Methods

- VowelClassifier (unsigned int input_dim, const VowelClassifierLayout *layout)
- ~VowelClassifier (void)
- VowelClassifier (VowelClassifier &source)
- bool load (fstream &fs)
- void save (fstream &fs) const
- VowelClassification classifyVowel (const VowelSample *input)
- void learnVowel (const VowelSample *input)
- void updateLearned (void)
- void resetLearned (void)

- void setLearningParameters (VowelClassifierLearnParams params)
- VowelClassifierLearnParams getLearningParameters (void) const
- const char * getExpertName (unsigned int expert_number)
- void randomizeExpert (unsigned int expert)
- void setRandomizationParameters (unsigned int expert, double weight_low, double weight_high, double theta_low, double theta_high)
- bool setActivationFunction (unsigned int expert_number, PerceptronLayerType type, const char *fact_name)
- PerceptronNetwork * getNetwork (unsigned int expert_number) const

Static Public Attributes

- vector< double > correct_results [3][(VowelSampleType) Unknown+1]

Private Methods

- vector< double > getExpertResults (PerceptronNetwork *exp, vector< double > in)
- void singleExpertLearn (PerceptronNetwork *exp, vector< double > input, vector< double > optimal)
- void initializeCorrectResults (void)

Private Attributes

- PerceptronNetwork * nn_a_ou_ei
- PerceptronNetwork * nn_o_u_aei
- PerceptronNetwork * nn_e_i_aou
- VowelClassifierLearnParams lpar
- vector< RandomInterval > weight_initializations
- vector< RandomInterval > theta_initializations

3.17.1 Detailed Description

Vowel classification system.

Classifies and learns VowelSample's.

3.17.2 Constructor & Destructor Documentation

3.17.2.1 VowelClassifier::VowelClassifier (unsigned int *input_dim*, const VowelClassifierLayout * *layout*)

Basic constructor.

Construct the VowelClassifier, given its layout through the parameters.

Parameters:

input_dim The input dimension for each sample the classifier should process.

layout Part of the neural network layout, the dimensions of each layer, except the first and the last one.

3.17.2.2 VowelClassifier::~VowelClassifier (void)

Default destructor.

3.17.2.3 VowelClassifier::VowelClassifier (VowelClassifier & *source*)

Copy constructor.

Parameters:
source Source object to be copied.

3.17.3 Member Function Documentation

3.17.3.1 VowelClassification VowelClassifier::classifyVowel (const VowelSample * *input*)

Classify a vowel.

Parameters:
input The vowel sample the system should try to classify.

3.17.3.2 const char* VowelClassifier::getExpertName (unsigned int *expert_number*)

Get the symbolic name of the expert network associated with *expert_number*.

Parameters:
expert_number The number of expert the name should be obtained from. Can range from 0 to 2.

Returns:
The name of the expert.

3.17.3.3 vector< double > VowelClassifier::getExpertResults (PerceptronNetwork * *exp*, vector< double > *in*) [private]

Ask one expert individually.

Processes a single neural network by feeding input *in*, and returning the networks output layer.

Parameters:
exp The expert to be asked.

in The network input.

3.17.3.4 VowelClassifierLearnParams VowelClassifier::getLearningParameters (void) const

Get the currently used learning parameters.

3.17.3.5 PerceptronNetwork * VowelClassifier::getNetwork (unsigned int *expert_number*) const

Get the PerceptronNetwork object behind an expert.

Parameters:
expert_number The network to be returned.

Returns:
The network.

3.17.3.6 void VowelClassifier::initializeCorrectResults (void) [private]

Create the initialization maps in correct_results.

3.17.3.7 void VowelClassifier::learnVowel (const VowelSample * *input*)

Learn a vowel *input* using the current learning parameters.

Note that while the internal system state changes due to the learning, the effective network does not. Hence, if you want to make the changes effective, you have to call updateLearned () afterwards. By choosing to call it after each learnVowel () call, you implement online-learning. Using a number of learnVowel () calls and then calling updateLearned () does all the accumulated changes at once, hence you implement batch-learning.

Parameters:
input The vowel, including its type_correct member, that the system should learn to classify.

3.17.3.8 bool VowelClassifier::load (fstream & *fs*)

Load three networks from the stream *fs*.

Parameters:
fs The stream the system is loaded from.

Returns:
True on success, false on failure.

3.17.3.9 void VowelClassifier::randomizeExpert (unsigned int *expert*)

Randomize the network parameters.

Parameters:
expert The expert index (0-2) that is initialized using its current settings.

3.17.3.10 void VowelClassifier::resetLearned (void)

Reset the currently learned parameter differences.

After having called updateLearned (), the differences are still active. To reset the learned differences when starting a new learning epoch, you have to call this method.

3.17.3.11 void VowelClassifier::save (fstream & *fs*) const

Save the entire classification system.

Parameters:

fs The already open write stream the system is saved to.

3.17.3.12 bool VowelClassifier::setActivationFunction (unsigned int *expert_number*, PerceptronLayerType *type*, const char * *fact_name*)

Set the activation function name.

Parameters:

expert_number The number (0-2) of the expert you want to change.

type The layer type you want to change the activation function of.

fact_name The symbolic name of the activation function.

Returns:

True on success, false on failure.

3.17.3.13 void VowelClassifier::setLearningParameters (VowelClassifierLearnParams *params*)

Set the learning parameters.

Change the internal learning parameters of the system. To keep this extendable we use a structure here.

Parameters:

params The parameters to apply.

3.17.3.14 void VowelClassifier::setRandomizationParameters (unsigned int *expert*, double *weight_low*, double *weight_high*, double *theta_low*, double *theta_high*)

Set the randomization parameters for an expert.

Parameters:

expert The expert index (0-2).

weight_low The lower randomization boundary for the weights.

weight_high The higher boundary for the weights.

theta_low The lower theta-initialization boundary.

theta_high The higher theta-initialization boundary.

3.17.3.15 void VowelClassifier::singleExpertLearn (PerceptronNetwork * *exp*, vector< double > *input*, vector< double > *optimal*) [private]

Propagate and backpropagate for a single expert.

Parameters:

exp The expert network to be processed.

input The input vector.

optimal The optimal output vector used for the backprogation process.

3.17.3.16 void VowelClassifier::updateLearned (void)

Update the effective system states.

Updating the states make the changes the learnVowel () method produced effective, hence increasing the classification success, hopefully.

3.17.4 Member Data Documentation

3.17.4.1 vector< double > VowelClassifier::correct_results [static]

The training map to teach the experts.

Each expert has its own translation map. The map itself maps a vowel type to the optimal expert results. For example, the first expert categorizes into the groups { A, OU, EI }, and would have this entries for the A vowel type:

correct_results[0][(VowelSampleType) A][0] = 1.0; correct_results[0][(VowelSampleType) A][1] = 0.0; correct_results[0][(VowelSampleType) A][2] = 0.0;

This is what it wants to tell you: "When the correct type is an A, the first output neuron should have level 1.0, the second and third the level 0.0". (For the first expert).

Note, that before using the correct_results map you have to initialize it using initializeCorrectResults. Also, the array should be used read-only from outside this class.

3.17.4.2 VowelClassifierLearnParams VowelClassifier::lpar [private]

The system internal learning parametrization.

3.17.4.3 PerceptronNetwork* VowelClassifier::nn_a_ou_ei [private]

One MLP expert that groups the input vowel into three disjoint sets: { a }, { o, u } and { e, i }. Named "network 0".

3.17.4.4 PerceptronNetwork* VowelClassifier::nn_e_i_aou [private]

One MLP expert that groups the input vowel into three disjoint sets: { e }, { i } and { a, o, u }. Named "network 2".

3.17.4.5 PerceptronNetwork* VowelClassifier::nn_o_u_aei [private]

One MLP expert that groups the input vowel into three disjoint sets: { o }, { u } and { a, e, i }. Named "network 1".

3.17.4.6 vector<RandomInterval> VowelClassifier::theta_initializations [private]

The theta initialization array, containing three intervals.

3.17.4.7 vector<RandomInterval> VowelClassifier::weight_initializations [private]

The weight initialization array, containing three intervals.

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

- VowelClassifier.h
- VowelClassifier.cpp

3.18 VowelClassifier::VScompare Struct Reference

Public Methods

- bool operator() (const VowelSampleType s1, const VowelSampleType s2) const

3.18.1 Detailed Description

Helper structure for use with the map template.

3.18.2 Member Function Documentation

3.18.2.1 bool VowelClassifier::VScompare::operator() (const VowelSampleType *s1*, const VowelSampleType *s2*) const [inline]

Standard operator to be used with iterators, implementing a simple order for the vowel types.

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

- VowelClassifier.h

3.19 VowelClassifierLayout Struct Reference

`#include <VowelClassifier.h>`

Public Attributes

- vector< unsigned int > net1
- vector< unsigned int > net2
- vector< unsigned int > net3

3.19.1 Detailed Description

Structure to specify the layout of the VowelClassifier neural networks.

For each network, there must be at least one number given. The size of the first and the last layers of each network is fixed by the other parameters (for the Input layer its the dimension of the input vector, for the Output layer its always three).

3.19.2 Member Data Documentation

3.19.2.1 vector<unsigned int> VowelClassifierLayout::net1

The first networks size (a, ou, ei).

3.19.2.2 vector<unsigned int> VowelClassifierLayout::net2

The second networks size (o, u, aei).

3.19.2.3 vector<unsigned int> VowelClassifierLayout::net3

The third networks size (e, i, aou).

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

- VowelClassifier.h

3.20 VowelClassifierLearnParams Struct Reference

`#include <VowelClassifier.h>`

Public Attributes

- double epsilon
- double optimal_tolerance
- double weight_decay
- double momentum_term

3.20.1 Detailed Description

Parameter structure to set the learning behaviour of the system.

For now, it only contains one parameter, but we use a structure nevertheless, to make it extendable, in case further learn parameters are required.

3.20.2 Member Data Documentation

3.20.2.1 double VowelClassifierLearnParams::epsilon

Epsilon, the main weight and theta value learn parameter.

3.20.2.2 double VowelClassifierLearnParams::momentum_term

Momentum term parameter. (0.5 to 0.9).

3.20.2.3 double VowelClassifierLearnParams::optimal_tolerance

Optimal tolerance value. (0.0 to 0.2).

3.20.2.4 double VowelClassifierLearnParams::weight_decay

Weight decay factor. (0.00005 to 0.0001).

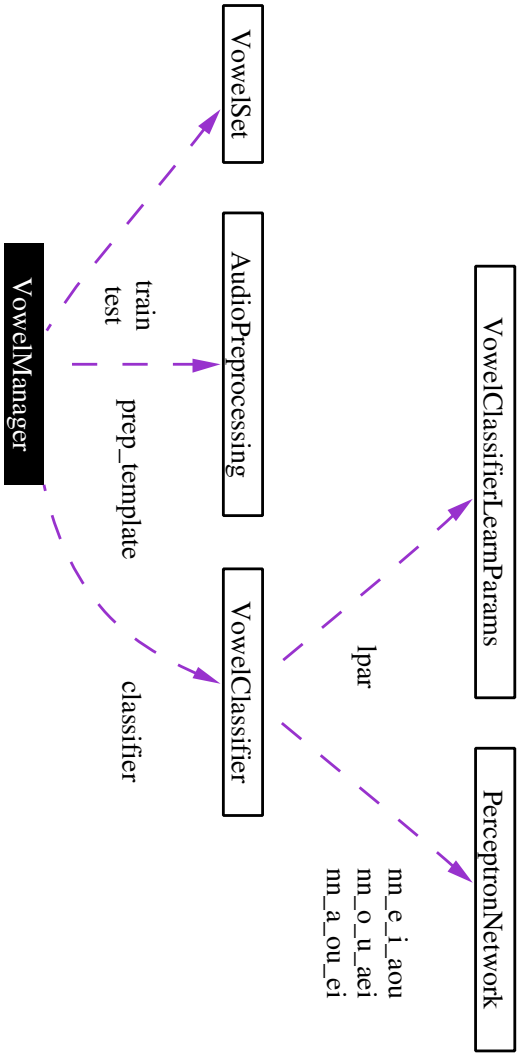
The documentation for this struct was generated from the following file:

- VowelClassifier.h

3.21 VowelManager Class Reference

`#include <VowelManager.h>`

Collaboration diagram for VowelManager:



Public Methods

- VowelManager (void)
- ~VowelManager (void)
- void learnEpochOnline (void)
- void learnEpochBatch (void)
- vector< double > getErrorMedian (VowelSet *set)
- unsigned int getEpochCounter (void)
- void resetEpochCounter (void)

Public Attributes

- VowelSet * train
- VowelSet * test
- VowelClassifier * classifier
- AudioPreprocessing * prep_template

Private Methods

- bool checkExpertCorrect (unsigned int expert, vector< double > &exp_output, VowelSampleType correct)
- unsigned int getMostActiveIndex (vector< double > &res)

Private Attributes

- unsigned int epoch_count

3.21.1 Detailed Description

Vowel management, set handling and learning algorithms.

Handles the learning, classification and management of vowels throughout the program.

Note that this class is the main interaction point for the GUI. The GUI sets and visualizes parameters to the sets ('train' and 'test'), does handle the learning parameters of 'classifier' and uses the methods to draw the errorterm graphs for each network and the entire system. Also, the class is quite open to external manipulation, and this is done intentionally. As the GUI has to know the structure of this class anyway to visualize the concepts of the system, it could also just manipulate it directly.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 VowelManager::VowelManager (void)

Default constructor.

3.21.2.2 VowelManager::~VowelManager (void)

Default destructor.

3.21.3 Member Function Documentation

3.21.3.1 bool VowelManager::checkExpertCorrect (unsigned int *expert*, vector< double > & *exp_output*, VowelSampleType *correct*) [private]

Check one single expert result for correctness.

Parameters:

expert The index for the expert that should be verified.

exp_output The expert output.

correct The known-correct type of the sample that produced the expert results.

Returns:
True when the expert was correct, false if it was not.

3.21.3.2 unsigned int VowelManager::getEpochCounter (void)

Getter function for the learning epoch counter.

Returns:
The number of epochs already learned in this system.

3.21.3.3 vector< double > VowelManager::getErrorMedian (VowelSet * *set*)

Get the median error for all samples within the *set*.

All the vowels in the given *set* are tested using the classifier, testing a single epoch. All error terms are calculated against the number of vowels in the set. The individual errorterms for each network are seperatly calculated. Thus, four errorterms are returned by the method.

Parameters:
set The set of vowels to be tested by the system.

Returns:
A vector of four errorterms, where the first represents the overall error of the system and the second to fourth represent the errors for each individual network, coined ”net0”, ”net1” and ”net2”. The names of the networks can be obtained by classifier->getExpertName (number).

3.21.3.4 unsigned int VowelManager::getMostActiveIndex (vector< double > & *res*) [private]

Utility function, finding the maximum out of a double vector.

Parameters:
res Vector of double values.

Returns:
Index whose element within the vector is maximal.

3.21.3.5 void VowelManager::learnEpochBatch (void)

Learn one epoch using the batch method.

The same description as to the learnEpochOnline method applies, except that the batch method is used and all the summed mistakes for each classification is reduced at the end.

3.21.3.6 void VowelManager::learnEpochOnline (void)

Learn one epoch using the online method.

All parameters to the learning process have to be set already within the classifier. For the learning process, the train set is used. One epoch is learned, that means every sample of the training set is learned exactly once. The online method is used, so after each classification the mistake is reduced for that sample.

3.21.3.7 void VowelManager::resetEpochCounter (void)

Reset the epoch counter.

3.21.4 Member Data Documentation

3.21.4.1 VowelClassifier* VowelManager::classifier

The classifier used to classify both the training and test sets.

Additionally, the train set is used to teach the classifiers internal expert systems.

The GUI can also use the classifier to check a single vowel, using classifier->classifyVowel ().

3.21.4.2 unsigned int VowelManager::epoch_count [private]

Counter for the epochs learned. For each call to learnEpoch* it is incremented. The epoch counter is resetted only on creation of a new manager, or when the resetEpochCounter method is called.

3.21.4.3 AudioPreprocessing* VowelManager::prep_template

The audio preprocessor used to build the vowel sets.

Vowel sets can be build either from existing VowelSample object or from audio files directly. The GUI does give the filenames to each set, but the preprocessor used to convert the AUFile objects to VowelSample objects is this one. The parameters to the preprocessor are set by the GUI, and this is the template copied for each preprocessing done. This object is never used to preprocess any file, but used as runtime template.

3.21.4.4 VowelSet* VowelManager::test

The test set of vowels.

See the note for the training set.

3.21.4.5 VowelSet* VowelManager::train

The training set of vowels.

Both the training and test sets can and should be used from external classes (the GUI). But the use should be done exclusivly over the methods the VowelSet class provides.

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

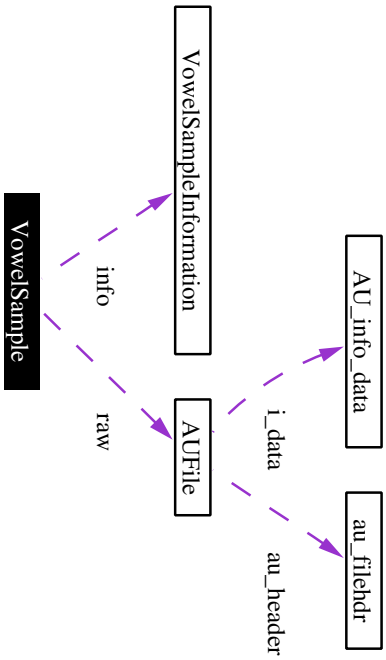
- VowelManager.h

- VowelManager.cpp

3.22 VowelSample Class Reference

`#include <VowelSample.h>`

Collaboration diagram for VowelSample:



Public Methods

- VowelSample (bool record=false)
- ~VowelSample (void)
- bool open (const char *filename, AudioPreprocessing &prep_template)
- VowelSample (VowelSample &source)
- void setCorrectVowel (VowelSampleType type)
- bool saveRawSample (const char *filename=0)
- bool isValidUserInformation (void)

Public Attributes

- VowelSampleType type_correct
- vector< double > sample
- vector< double > fft_energy
- char * filename
- AUFile * raw

Private Methods

- bool convertUserinfoToC (void)
- void convertCtoUserinfo (void)

Private Attributes

- bool info_success
- VowelSampleInformation info

3.22.1 Detailed Description

Representation of one vowel.

Used from the VowelSet class. Does the handling of audio files from the raw audio data to the neural network input.

3.22.2 Constructor & Destructor Documentation

3.22.2.1 VowelSample::VowelSample (bool *record* = false)

Standard constructor.

Parameters:

record When set to true, the VowelSample object is prepared for recording, i.e. its raw member is initialized.

3.22.2.2 VowelSample::~VowelSample (void)

Basic destructor.

3.22.2.3 VowelSample::VowelSample (VowelSample & *source*)

Copy constructor.

3.22.3 Member Function Documentation

3.22.3.1 void VowelSample::convertCtoUserinfo (void) [private]

Convert the class elements into the packed 'info' structure.

3.22.3.2 bool VowelSample::convertUserinfoToC (void) [private]

Convert the user information within the 'info' structure to the corresponding class elements.

Returns:

True on success, false on failure.

3.22.3.3 bool VowelSample::isValidUserInformation (void) [inline]

Getter for the user information structure success.

Returns:
True when there was a userinfo structure in the file, false if there was none.

3.22.3.4 bool VowelSample::open (const char * *filename*, AudioPreprocessing & *prep_template*)

Open an audio file

Convert raw data *raw* using the audio preprocessor *prep_template* as processing template. All the conversion is done using the active settings of the template. Also extract the raw information structure stored with the AU file into a C level structure (VowelSampleInformation).

Parameters:
filename The raw audio input filename.
prep_template The conversion template which transforms the raw data into a windows energy vector.

Returns:
True on success, false on failure.

3.22.3.5 bool VowelSample::saveRawSample (const char * *filename* = 0)

Save the raw data back to an AU file.

Parameters:
filename When non-zero, the filename the AU file will be written to. When zero, the original filename is used.

Returns:
True when the operation was successful, otherwise false.

3.22.3.6 void VowelSample::setCorrectVowel (VowelSampleType *type*)

Set the correct vowel type for this sample.

Parameters:
type The correct vowel for this sample.

3.22.4 Member Data Documentation

3.22.4.1 vector<double> VowelSample::fft_energy

The FFT energy array, showing the frequency based amplitude energy for the input sample.

3.22.4.2 char* VowelSample::filename

The original AU filename that produced the sample.

3.22.4.3 VowelSampleInformation VowelSample::info [private]

Userdata part of the AU file converted to a real structure. Only used temporarily, no real processing takes place here except conversion before and after AUFile read/write operations.

3.22.4.4 bool VowelSample::info_success [private]

Flag, when true there is valid userdata.

3.22.4.5 AUFile* VowelSample::raw

The original AU file associated with this sample.

Warning: can be NULL.

3.22.4.6 vector<double> VowelSample::sample

The processed audio sample.

Stored as windows vectors of energy levels in the frequency windows.

3.22.4.7 VowelSampleType VowelSample::type_correct

The correct type of vowel this sample represents. In case the vowel should be determined or is otherwise unknown, it can be set to 'unknown'.

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

- VowelSample.h
- VowelSample.cpp

3.23 VowelSampleInformation Struct Reference

`#include <VowelSample.h>`

Public Methods

- uint32_t magic __attribute__ ((packed))
- uint32_t sample_type __attribute__ ((packed))

3.23.1 Detailed Description

Low level structure that is stored within the AU file but holds information on vowel-level (hence placed here and not within the Audio module).

XXX: Note that the elements of this structure have to be packed so that the structure can be moved directly from the raw file. The little/big endian conversions have to be done on the word sized types, still.

3.23.2 Member Function Documentation

3.23.2.1 uint32_t sample_type VowelSampleInformation::__attribute__ ((packed))

VowelSampleType as number representation

3.23.2.2 uint32_t magic VowelSampleInformation::__attribute__ ((packed))

MAGIC word ('ntic', 0x6e746963)

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

- VowelSample.h

3.24 VowelSet Class Reference

`#include <VowelManager.h>`

Public Methods

- VowelSet (const char *name="SetName")
- ~VowelSet (void)
- VowelSet (VowelSet &source)
- bool load (fstream &fs, AudioPreprocessing &prep, void(*cback)(void *)=NULL, void *user=NULL)
- void save (fstream &fs)
- void setName (const char *name)
- bool addSample (VowelSample *sample)
- bool addSample (const char *filename, AudioPreprocessing &prep)
- bool removeSample (VowelSample *sample)
- void clearList (void)
- VowelSample * findSample (const char *filename)
- vector< VowelSample * > & getSampleList (void)

Public Attributes

- char * name

Private Attributes

- vector< VowelSample * > vowels

3.24.1 Detailed Description

VowelSet class, management of multiple VowelSample elements.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 VowelSet::VowelSet (const char * *name* = "SetName")

Basic constructor.

Parameters:

name The name of the set, optionally given. The name is pointer-copied, so we steal a pointer here.

3.24.2.2 VowelSet::~VowelSet (void)

Standard destructor.

3.24.2.3 VowelSet::VowelSet (VowelSet & *source*)

Copy constructor.

Parameters:

source Source object to be copied.

3.24.3 Member Function Documentation

3.24.3.1 bool VowelSet::addSample (const char * *filename*, AudioPreprocessing & *prep*)

Add a sample to the set from AU file.

Parameters:

filename Filename of the AU file to open and add to the set.

prep The AudioPreprocessor template to use for conversion to a VowelSample. It is copied to a new processor, so its internal state is preserved.

Returns:

True on success, otherwise false.

3.24.3.2 bool VowelSet::addSample (VowelSample * *sample*)

Add a sample to the set.

Parameters:

sample The sample to be added to the list. Note that no dupe checking takes place, so its possible to add the same sample multiple times.

Returns:

True on success, otherwise false.

3.24.3.3 void VowelSet::clearList (void)

Clear the entire sample list.

3.24.3.4 VowelSample * VowelSet::findSample (const char * *filename*)

Find the Sample containing data for a specific filename

Parameters:

filename for search

Returns:

VowelSample containing data for filename, NULL if unknown

3.24.3.5 vector< VowelSample * > & VowelSet::getSampleList (void)

Get the entire list of samples stored in this set.

Returns:
The list of samples in this set.

3.24.3.6 bool VowelSet::load (fstream & *fs*, AudioPreprocessing & *prep*, void(* *cback*)(void *) = NULL, void * *user* = NULL)

Load method.

Note that this method destructs all previous samples automatically.

Parameters:
fs The file stream the set will be created from.

prep The audio preprocessing template used to process all the samples.

cback An optional callback function called after each imported sample. As the import process can take a while, you can use it to update events and the like.

user An optional parameter passed to the callback function.

Returns:
True on success, false on failure.

3.24.3.7 bool VowelSet::removeSample (VowelSample * *sample*)

Remove a sample from the set.

Note: Does not delete the sample object itself.

Parameters:
sample The sample that will be removed in case its in the list. When multiple samples of the same type are in the list, only the first occurance will be removed.

Returns:
Return true in case it was removed, otherwise return false.

3.24.3.8 void VowelSet::save (fstream & *fs*)

Save method.

Parameters:
fs The file stream the set will be saved to.

3.24.3.9 void VowelSet::setName (const char * *name*)

Set the name of the set.

Parameters:
name The new name of the set.

3.24.4 Member Data Documentation

3.24.4.1 char* VowelSet::name

The name of this set. Ensured not to be NULL at any time.

3.24.4.2 vector<VowelSample *> VowelSet::vowels [private]

The list of vowels stored in this set.

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

- VowelManager.h
- VowelManager.cpp

3.25 WaveForm Class Reference

`#include <GUIAudio.h>`

Public Methods

- WaveForm (VowelSample *sample, QWidget *parent=0, const char *name=0)
- ~WaveForm (void)
- QSizePolicy sizePolicy () const
- QSize minimumSiteHint () const

Protected Methods

- void paintEvent (QPaintEvent *)

Private Attributes

- QPainter * painter

3.25.1 Detailed Description

Wrapper class from audio raw data to a GUI visualization

The audio data handling is encapsulated using this class, providing a widget for the GUI to use. Hence, the GUI is relieved from the effort to understand the audio class.

3.25.2 Constructor & Destructor Documentation

3.25.2.1 WaveForm::WaveForm (VowelSample * *sample*, QWidget * *parent* = 0, const char * *name* = 0)

Constructor which draws the waveform diagramm for a given *sample*.

Parameters:

sample The VowelSample the diagramm is created from.

parent The standard QT parent widget.

name The widgets' name.

3.25.2.2 WaveForm::~WaveForm (void)

Default destructor.

3.25.3 Member Function Documentation

3.25.3.1 QSize WaveForm::minimumSiteHint () const

Specify the minimum size necessary to display the waveform.

3.25.3.2 void WaveForm::paintEvent (QPaintEvent *) [protected]

QT paintEvent method, used to redraw the modified screen content.

3.25.3.3 QSizePolicy WaveForm::sizePolicy () const

The size policy for this widget, denying resizing.

3.25.4 Member Data Documentation

3.25.4.1 QPainter* WaveForm::painter [private]

The QT painter needed to (re-) draw the canvas. Initialized once upon object construction and used from there.

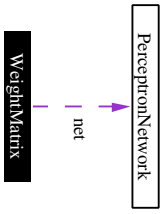
The documentation for this class was generated from the following file:

- GUIAudio.h

3.26 WeightMatrix Class Reference

`#include <GUINeuralNet.h>`

Collaboration diagram for WeightMatrix:



Public Methods

- WeightMatrix (QWidget *parent=0, const char *name=0)
- ~WeightMatrix (void)
- void setNet (PerceptronNetwork *net)
- void setLayerNum (unsigned int layernum)
- unsigned int getLayerMax (void) const

Protected Methods

- void paintEvent (QPaintEvent *)

Private Methods

- void paintSquare (unsigned int wx, unsigned int wy, const QColor &col)

Private Attributes

- unsigned int weight_fromcount
- unsigned int weight_tocount
- unsigned int layernum
- unsigned int x_needed
- unsigned int y_needed
- PerceptronNetwork * net
- vector< double > thetas
- double theta_min
- double theta_max
- double theta_mid
- double theta_rng
- vector< vector< double > > weights
- double weight_min
- double weight_max
- double weight_mid
- double weight_rng
- QPainter * painter

- unsigned int margin_up
- unsigned int margin_down
- unsigned int margin_left
- unsigned int margin_right
- int x_box
- int y_box
- int square_size_x
- int square_size_y

3.26.1 Detailed Description

Wrapper class from a neural network weight matrix to a GUI visualization.

The neural network related code is encapsulated within this class, so the GUI does use only this widget and does not have to cope with the PerceptronNetwork directly.

3.26.2 Constructor & Destructor Documentation

3.26.2.1 WeightMatrix::WeightMatrix (QWidget * *parent* = 0, const char * *name* = 0)

Constructor which draws the weight diagramm for the weights going from layer *layernum* within network *net.*

Parameters:

parent The standard QT parent widget.

name The widgets' name.

3.26.2.2 WeightMatrix::~WeightMatrix (void)

Standard destructor.

3.26.3 Member Function Documentation

3.26.3.1 unsigned int WeightMatrix::getLayerMax (void) const

Give the maximum layer number.

Returns:

The maximum layer number plus one, i.e. 2 means: 0-1 is allowed.

3.26.3.2 void WeightMatrix::paintEvent (QPaintEvent *) [protected]

QT paintEvent method, used to redraw the modified screen content.

3.26.3.3 void WeightMatrix::paintSquare (unsigned int *wx*, unsigned int *wy*, const QColor & *col*) [private]

Paint a single rectangle on the rectangle plane.

Parameters:

wx The virtual x position on the plane.

wy The virtual y position on the plane.

col The color the rectangle is drawn in.

3.26.3.4 void WeightMatrix::setLayerNum (unsigned int *layernum*)

Set the layer which outgoing weights will be shown

Parameters:

layernum The layer index into the network.

3.26.3.5 void WeightMatrix::setNet (PerceptronNetwork * *net*)

Set the PerceptronNetwork

Parameters:

net The neural network the weight diagramm is created from.

3.26.4 Member Data Documentation

3.26.4.1 unsigned int WeightMatrix::layernum [private]

Layer which outgoing weights will be shown

3.26.4.2 unsigned int WeightMatrix::margin_down [private]

Margin to the downside border of the widget, in pixels

3.26.4.3 unsigned int WeightMatrix::margin_left [private]

Margin to the left border of the widget, in pixels

3.26.4.4 unsigned int WeightMatrix::margin_right [private]

Margin to the right border of the widget, in pixels

3.26.4.5 unsigned int WeightMatrix::margin_up [private]

Margin to the upper border of the widget, in pixels

3.26.4.6 PerceptronNetwork* WeightMatrix::net [private]

The neural network the weight diagram is created from.

3.26.4.7 QPainter* WeightMatrix::painter [private]

The QT painter needed to (re-) draw the canvas. Initialized once upon object construction and used from there.

3.26.4.8 int WeightMatrix::square_size_x [private]

The size of one square in x direction

3.26.4.9 int WeightMatrix::square_size_y [private]

The size of one square in y direction

3.26.4.10 double WeightMatrix::theta_max [private]

Maximum theta value found in current theta vector.

3.26.4.11 double WeightMatrix::theta_mid [private]

Middle of theta_min and theta_max.

3.26.4.12 double WeightMatrix::theta_min [private]

Minimum theta value found in current theta vector.

3.26.4.13 double WeightMatrix::theta_rng [private]

Absolute range between theta_min and theta_max.

3.26.4.14 vector<double> WeightMatrix::thetas [private]

The theta column values. Extracted from the neurons of the displayed layer.

3.26.4.15 unsigned int WeightMatrix::weight_fromcount [private]

Hold the number of weight origins.

3.26.4.16 double WeightMatrix::weight_max [private]

The maximum weight found in the current weight matrix.

3.26.4.17 double WeightMatrix::weight_mid [private]

Middle between weight_min and weight_max.

3.26.4.18 double WeightMatrix::weight_min [private]

The minimum weight found in the current weight matrix.

3.26.4.19 double WeightMatrix::weight_rng [private]

Absolute distance between weight_min and weight_max.

3.26.4.20 unsigned int WeightMatrix::weight_tocount [private]

Hold the number of weight targets.

3.26.4.21 vector<vector<double> > WeightMatrix::weights [private]

The array of weight values displayed. Extracted from both the individual weights of the neurons of the current layer.

3.26.4.22 int WeightMatrix::x_box [private]

The actual available pixelsize for the squares in x direction

3.26.4.23 unsigned int WeightMatrix::x_needed [private]

Number of squares needed in the widget in horizontal direction. Calculated as $x_{needed} = 2 + weight_tocount$.

3.26.4.24 int WeightMatrix::y_box [private]

The actual available pixelsize for the squares in y direction

3.26.4.25 unsigned int WeightMatrix::y_needed [private]

Number of squares needed in the widget in vertical direction. Calculated as $y_{needed} = weight_fromcount$.

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

- GUINeuralNet.h
- GUINeuralNet.cpp

Index