Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 DatabaseOps Class Reference

`#include <DatabaseOps.h>`

Static Public Member Functions

- static void ExportData (InputLayer *inputLayer, OutputLayer *outputLayer, string fileName)
- static void ImportData (InputLayer *inputLayer, OutputLayer *outputLayer, string fileName)

3.1.1 Detailed Description

This class contains methods to perform I/O operations on a SQLite 3 database

3.1.2 Member Function Documentation

3.1.2.1 ExportData()

```
void DatabaseOps::ExportData (
            InputLayer * inputLayer,
            OutputLayer * outputLayer,
            string fileName )  [static]
```

Generates a SQLite database at the specified path and exports all the weights and hyperparameters to it

Parameters

inputLayer	The input layer
outputLayer	The output layer
fileName	The output file name

3.1.2.2 ImportData()

```
void DatabaseOps::ImportData (
            InputLayer * inputLayer,
            OutputLayer * outputLayer,
            string fileName )  [static]
```

Reads and imports the weights of the network from the specified SQLite database

Parameters

inputLayer	The input layer
outputLayer	The output layer
fileName	The input file name

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

- DatabaseOps.h
- DatabaseOps.cpp

3.2 InputLayer Class Reference

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

Public Member Functions

- InputLayer ()
- void AddTrain (array< bool, T > &train)
- void ResetTrains ()
- array< array< double, T-1 >, CLASSES *NEURONS_IN > ApplyAlphas ()
- void UpdateAlphas (array< array< double, T >, CLASSES > &errors)

Public Attributes

- array< array< double, TYI >, CLASSES *NEURONS_IN > w

Protected Attributes

- array< array< bool, T >, NEURONS_IN > trains
- array< array< double, Ka >, TYI > basis
- short index = 0

3.2.1 Detailed Description

This class represents an input layer in the neural network and contains all the methods to perform operations on it

3.2.2 Constructor & Destructor Documentation

3.2.2.1 InputLayer()

```
InputLayer::InputLayer ( )
```

The cosntructor initialises the values of the weights and the basis matrix

3.2.3 Member Function Documentation

3.2.3.1 AddTrain()

```
void InputLayer::AddTrain (
            array< bool, T > & train )
```

Adds a train of spikes to the array, as position [index]

Parameters

train	The train of spikes

3.2.3.2 ApplyAlphas()

```
array< array< double, T-1 >, CLASSES *NEURONS_IN > InputLayer::ApplyAlphas ( )
```

Convolve alphas with the trains and apply the basis matrix. This is one part of the process to compute the potential of the output neuron

Returns

The preprocessed spike trains

3.2.3.3 ResetTrains()

```
void InputLayer::ResetTrains ( )
```

Reset the values of the trains and index

3.2.3.4 UpdateAlphas()

```
void InputLayer::UpdateAlphas (
            array< array< double, T >, CLASSES > & errors )
```

Updates the weights w using the errors errors The errors for updating the weights

3.2.4 Member Data Documentation

3.2.4.1 basis

```
array<array<double, Ka>, TYI> InputLayer::basis  [protected]
```

The basis matrix A contains the basis functions to apply to the weights w

3.2.4.2 index

```
short InputLayer::index = 0  [protected]
```

Index of next train in the array

3.2.4.3 trains

```
array<array<bool, T>, NEURONS_IN> InputLayer::trains  [protected]
```

The trains of spikes generated by rate encoding process

3.2.4.4 w

```
array<array<double, TYI>, CLASSES*NEURONS_IN> InputLayer::w
```

The weights w for every synapse

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

- Layer.h
- Layer.cpp

3.3 MatrixOps Class Reference

`#include <MatrixOps.h>`

Static Public Member Functions

- static array< double, T-1 > Conv (array< bool, T > const &f, array< double, TYI > const &g)
- static array< double, T-1 > SumColumns (array< array< double, T-1 >, CLASSES *NEURONS_IN > &vect, const short cl)
- template<std::size_t SIZE>
  static void Multiply (double c, array< double, SIZE > &vect)
- template<std::size_t ROWS, std::size_t COLS>
  static array< double, ROWS > Dot (array< array< double, COLS >, ROWS > &basis, array< double, COLS > &weight)
- template<std::size_t SIZE>
  static double Sum (const array< double, SIZE > &vect)
- template<std::size_t ROWS, std::size_t COLS>
  static array< array< double, ROWS >, COLS > Transpose (array< array< double, COLS >, ROWS > &input)
- template<std::size_t SIZE>
  static array< double, SIZE > SumArrays (array< double, SIZE > a, array< double, SIZE > b)

3.3.1 Detailed Description

This class contains custom matrix operations

3.3.2 Member Function Documentation

3.3.2.1 Conv()

```
array< double, T-1 > MatrixOps::Conv (
            array< bool, T > const & f,
            array< double, TYI > const & g )  [static]
```

Performs a partial convolution to aggregate the trains and alpha

Parameters

f	The spike train
g	The weights vector

Returns

The result of the convolution

3.3.2.2 Dot()

```
template<std::size_t ROWS, std::size_t COLS>
array< double, ROWS > MatrixOps::Dot (
            array< array< double, COLS >, ROWS > & basis,
            array< double, COLS > & weight )  [inline], [static]
```

Performs the dot product

Parameters

ROWS	The number of rows
COLS	The number of columns
basis	The basis matrix
weight	The weight vector

Returns

the result of the operation

3.3.2.3 Multiply()

```
template<std::size_t SIZE>
void MatrixOps::Multiply (
            double c,
            array< double, SIZE > & vect )  [inline], [static]
```

Multiplies an array by a constant

Parameters

SIZE	The length of the array
c	The constant
vect	The array

3.3.2.4 Sum()

```
template<std::size_t SIZE>
double MatrixOps::Sum (
            const array< double, SIZE > & vect )  [inline], [static]
```

Sums all the values in a vector

Parameters

SIZE	The length of the array

Parameters

ROWS	The number of rows
COLS	The number of columns
vect	The vector

Returns

The sum of the elements

3.3.2.5 SumArrays()

```
template<std::size_t SIZE>
array< double, SIZE > MatrixOps::SumArrays (
            array< double, SIZE > a,
            array< double, SIZE > b )  [inline], [static]
```

Performs element-wise sum of the elements in the arrays

Parameters

SIZE	The length of the arrays
a	Array 1
b	Array 2

Returns

The element-wise sum of the arrays

3.3.2.6 SumColumns()

```
array< double, T-1 > MatrixOps::SumColumns (
            array< array< double, T-1 >, CLASSES *NEURONS_IN > & vect,
            const short cl )  [static]
```

Sum the matrix along the columns, but only every cl rows

Parameters

vect	The matrix
cl	The frequency of rows to be summed, i.e. cl=2 half of the rows are summed

Returns

The sum

3.3.2.7 Transpose()

```
template<std::size_t ROWS, std::size_t COLS>
array< array< double, ROWS >, COLS > MatrixOps::Transpose (
            array< array< double, COLS >, ROWS > & input )  [inline], [static]
```

Performs a matrix transpose

Parameters

ROWS	The number of rows
COLS	The number of columns
input	The input matrix

Returns

The transposed matrix

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

- MatrixOps.h
- MatrixOps.cpp

3.4 Network Class Reference

```
#include <Network.h>
```

Public Member Functions

- Network ()
- char Run (array< unsigned char, NEURONS_IN > image)
- template<std::size_t FILTER_SIZE>
  void Train (short epochs, int trainingImages=60000, vector< pair< array< unsigned char, NEURONS_IN >, unsigned char >> *trainingData=nullptr, array< unsigned char, FILTER_SIZE > *filter=nullptr)
- void ImportData (string fileName="data.db")
- void ExportData (string fileName="data.db")
- template<std::size_t FILTER_SIZE>
  int Validate (int testImages=10000, bool testSet=true, array< unsigned char, FILTER_SIZE > *filter=NULL)
- int ValidateDataset (vector< pair< array< unsigned char, NEURONS_IN >, unsigned char >> &trainingSet)
- int CrossValidate ()

Protected Member Functions

- void ResetNetwork ()

Protected Attributes

- InputLayer inputLayer
- OutputLayer outputLayer

3.4.1 Detailed Description

The Network class contains the methods to perform all the possible operations on the network

3.4.2 Constructor & Destructor Documentation

3.4.2.1 Network()

```
Network::Network ( )
```

The constructor instantiates the layers

3.4.3 Member Function Documentation

3.4.3.1 CrossValidate()

```
int Network::CrossValidate ( )
```

Performs 10-folds cross-validation on the entire training set - single epoch

Returns

The average of correct classification - Max 6000

3.4.3.2 ExportData()

```
void Network::ExportData (
            string fileName = "data.db" )
```

Export network weights to a database file

Parameters

fileName	The file name of the database to which export the weights

3.4.3.3 ImportData()

```
void Network::ImportData (
            string fileName = "data.db" )
```

Import network weights from a database file

Parameters

fileName	The file name of the database from which import the weights

3.4.3.4 ResetNetwork()

```
void Network::ResetNetwork ( )  [protected]
```

To reset both layers after processing every image

3.4.3.5 Run()

```
char Network::Run (
            array< unsigned char, NEURONS_IN > image )
```

Run the network on an image and return the predicted class

Parameters

image	The input image to be classified

Returns

The predicted class of the image

3.4.3.6 Train()

```
template<std::size_t FILTER_SIZE>
void Network::Train (
```

```
            short epochs,
            int trainingImages = 60000,
            vector< pair< array< unsigned char, NEURONS_IN >, unsigned char >> * training↩
Data = nullptr,
            array< unsigned char, FILTER_SIZE > * filter = nullptr )
```

Trains the network

Parameters

FILTER_SIZE	The number of elements in the filter
epochs	The number of iterations to perform
trainingImages	The number of images to train the network on
trainingData	The data to train the network on - if it's not nullptr, it overrides the trainingImages parameter
filter	The filter to limit the labels in the training - if nullptr, all 10 digits are considered

3.4.3.7 Validate()

```
template<std::size_t FILTER_SIZE>
int Network::Validate (
            int testImages = 10000,
            bool testSet = true,
            array< unsigned char, FILTER_SIZE > * filter = NULL )
```

Validates the network on any data

Parameters

FILTER_SIZE	The number of elements in the filter
testImages	The number of images used for validation. The range should be [1 10000] for the test set and [1 60000] for the training one
testSet	If true the test set is used, otherwise the training set
filter	The filter to limit the labels in the training - if nullptr, all 10 digits are considered

Returns

The number of correct classifications

3.4.3.8 ValidateDataset()

```
int Network::ValidateDataset (
            vector< pair< array< unsigned char, NEURONS_IN >, unsigned char >> & training↩
Set )
```

Validates the network on the given images

Parameters

trainingSet	The images that should be used to validate the network on

Returns

The number of correct classifications

3.4.4 Member Data Documentation

3.4.4.1 inputLayer

```
InputLayer Network::inputLayer  [protected]
```

The input layer

3.4.4.2 outputLayer

```
OutputLayer Network::outputLayer  [protected]
```

The output layer

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

- Network.h
- Network.cpp

3.5 OutputLayer Class Reference

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

Public Member Functions

- OutputLayer ()
- void Reset ()
- void ComputeOutput (array< array< double, T-1 >, CLASSES *NEURONS_IN > &synapsesOut)
- array< array< double, T >, CLASSES > ComputeErrors (unsigned char label) const
- char ComputeWinner () const
- void UpdateBetas (array< array< double, T >, CLASSES > &errors)
- void UpdateGammas (array< array< double, T >, CLASSES > &errors)

Public Attributes

- array< array< double, TYO >, CLASSES > v
- array< double, CLASSES > gammas

Protected Attributes

- array< array< double, T >, CLASSES > u
- array< array< bool, T >, CLASSES > y
- array< array< double, Kb >, TYO > basis

3.5.1 Detailed Description

This class represents an output layer in the neural network and contains all the methods to perform operations on it

3.5.2 Constructor & Destructor Documentation

3.5.2.1 OutputLayer()

```
OutputLayer::OutputLayer ( )
```

The cosntructor initialises the values of the weights and the basis matrix

3.5.3 Member Function Documentation

3.5.3.1 ComputeErrors()

```
array< array< double, T >, CLASSES > OutputLayer::ComputeErrors (
            unsigned char label ) const
```

Computes the errors at every time t for the given class label The class for which compute the errors

Returns

An array of errors for every class label

3.5.3.2 ComputeOutput()

```
void OutputLayer::ComputeOutput (
            array< array< double, T-1 >, CLASSES *NEURONS_IN > & synapsesOut )
```

Computes the potential and output starting from the preprocessed trains

Parameters

synapsesOut	The preprocessed spike trains from the input layer

3.5.3.3 ComputeWinner()

```
char OutputLayer::ComputeWinner ( ) const
```

Returns the "Winner" of the network: uses rate decoding to determine the class with the most spikes

Returns

The prediction of the network

3.5.3.4 Reset()

```
void OutputLayer::Reset ( )
```

Reset the values of the potential and output

3.5.3.5 UpdateBetas()

```
void OutputLayer::UpdateBetas (
            array< array< double, T >, CLASSES > & errors )
```

Updates the weights v using the errors errors The errors for updating the weights

See also

ComputeErrors

3.5.3.6 UpdateGammas()

```
void OutputLayer::UpdateGammas (
            array< array< double, T >, CLASSES > & errors )
```

Updates the biases using the errors errors The errors for updating the biases

See also

ComputeErrors

3.5.4 Member Data Documentation

3.5.4.1 basis

```
array<array<double, Kb>, TYO> OutputLayer::basis  [protected]
```

The basis matrix B contains the basis functions to apply to the weights v

3.5.4.2 gammas

```
array<double, CLASSES> OutputLayer::gammas
```

The biases of every class

3.5.4.3 u

```
array<array<double, T>, CLASSES> OutputLayer::u  [protected]
```

The potential of every output neuron at every time t

3.5.4.4 v

```
array<array<double, TYO>, CLASSES> OutputLayer::v
```

The weights of the feedback kernel

3.5.4.5 y

```
array<array<bool, T>, CLASSES> OutputLayer::y  [protected]
```

The output spike trains of every class

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

- Layer.h
- Layer.cpp

3.6 TrainingPool Class Reference

```
#include <TrainingPool.h>
```

Static Public Member Functions

- static void TrainAsync (short epochs, int trainingImages, string dbOut, bool validate)
- static void Join ()

Static Protected Member Functions

- static void InitTraining (short epochs, int trainingImages, string dbOut, bool validate)

Static Protected Attributes

- static vector< thread > runningInstances

3.6.1 Detailed Description

This calss contains methods for performing multiple training sessions on networks in parallel and exporting the results

3.6.2 Member Function Documentation

3.6.2.1 InitTraining()

```
void TrainingPool::InitTraining (
            short epochs,
            int trainingImages,
            string dbOut,
            bool validate )  [static], [protected]
```

Method that makes the call to the Train function of the Network class

Parameters

epochs	The number of epochs
trainingImages	The number of images to train the network on. The number should be 1-60000
dbOut	The path of the database where the data should be exported
validate	True if validation on the entire test set should be performed after the training

3.6.2.2 Join()

```
void TrainingPool::Join ( )  [static]
```

Wait for all running threads to finish executing

3.6.2.3 TrainAsync()

```
void TrainingPool::TrainAsync (
            short epochs,
            int trainingImages,
            string dbOut,
            bool validate )  [static]
```

Trains a network in a parallel thread and exports the training data

Parameters

epochs	The number of epochs
trainingImages	The number of images to train the network on. The number should be 1-60000
dbOut	The path of the database where the data should be exported
validate	True if validation on the entire test set should be performed after the training

3.6.3 Member Data Documentation

3.6.3.1 runningInstances

```
vector< thread > TrainingPool::runningInstances  [static], [protected]
```

A list of threads that are currently running

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

- TrainingPool.h
- TrainingPool.cpp

3.7 Utils Class Reference

```
#include <Utils.h>
```

Static Public Member Functions

- static array< float, NEURONS_IN > RateEncode (array< unsigned char, NEURONS_IN > &image)
- static array< bool, T > GenerateSpikes (float probability)
- template<std::size_t FILTER_SIZE>
  static vector< pair< array< unsigned char, NEURONS_IN >, unsigned char > > GetTrainingData (int NumberOfImages, array< unsigned char, FILTER_SIZE > *filter=nullptr)
- template<std::size_t FILTER_SIZE>
  static vector< pair< array< unsigned char, NEURONS_IN >, unsigned char > > GetTestData (int Number↩
  OfImages, array< unsigned char, FILTER_SIZE > *filter=nullptr)
- static array< array< double, Ka >, TYI > GenerateAlphaBasis ()
- static array< array< double, Kb >, TYO > GenerateBetaBasis ()
- static void PrintLine (string &&str)

3.7.1 Detailed Description

This class contains general methods used in the programme

3.7.2 Member Function Documentation

3.7.2.1 GenerateAlphaBasis()

```
array< array< double, Ka >, TYI > Utils::GenerateAlphaBasis ( )  [static]
```

Generates the basis matrix A

Returns

The basis matrix A

3.7.2.2 GenerateBetaBasis()

```
array< array< double, Kb >, TYO > Utils::GenerateBetaBasis ( )  [static]
```

Generates the basis matrix B

Returns

The basis matrix B

3.7.2.3 GenerateSpikes()

```
array< bool, T > Utils::GenerateSpikes (
            float probability )  [static]
```

Generate spikes trains from probabilities

Parameters

probability	The probability of spiking

Returns

The train of spikes

3.7.2.4 GetTestData()

```
template<std::size_t FILTER_SIZE>
vector< pair< array< unsigned char, NEURONS_IN >, unsigned char > > Utils::GetTestData (
            int NumberOfImages,
            array< unsigned char, FILTER_SIZE > * filter = nullptr )  [static]
```

Returns the test data

Parameters

FILTER_SIZE	The number of elements in the filter
NumberOfImages	Number of images to read from the database
filter	The filter to limit the labels in the training - if nullptr, all 10 digits are considered

Returns

The a list of <image, label> pairs

3.7.2.5 GetTrainingData()

```
template<std::size_t FILTER_SIZE>
vector< pair< array< unsigned char, NEURONS_IN >, unsigned char > > Utils::GetTrainingData (
            int NumberOfImages,
            array< unsigned char, FILTER_SIZE > * filter = nullptr )  [static]
```

Returns the training data

Parameters

FILTER_SIZE	The number of elements in the filter
NumberOfImages	Number of images to read from the database
filter	The filter to limit the labels in the training - if nullptr, all 10 digits are considered

Returns

The a list of <image, label> pairs

3.7.2.6 PrintLine()

```
void Utils::PrintLine (
            string && str )  [static]
```

Thread-safe method for printing to terminal

Parameters

str	The string to be printed to terminal

3.7.2.7 RateEncode()

```
array< float, NEURONS_IN > Utils::RateEncode (
            array< unsigned char, NEURONS_IN > & image )  [static]
```

Perform rate encode on the pixels of the image

Parameters

image	The input image

Returns

An array of probabilities of spiking

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

- Utils.h
- Utils.cpp

Chapter 4

File Documentation

4.1 Constants.h File Reference

```
#include "stdafx.h"
#include <cmath>
#include <fstream>
```

Variables

- const short CLASSES = 10
- const short NEURONS_IN = 784
- const short T = 4
- const short TYI = 3
- const short Ka = TYI
- const short TYO = 2
- const short Kb = TYO
- const float LEARNING_RATE = 0.00005
- const std::pair< float, float > P_RANGE = { 0, 0.5 }
- const string TRAIN_IMAGES_PATH = "D:\\train-images.idx3-ubyte"
- const string TRAIN_LABELS_PATH = "D:\\train-labels.idx1-ubyte"
- const string TEST_IMAGES_PATH = "D:\\t10k-images.idx3-ubyte"
- const string TEST_LABELS_PATH = "D:\\t10k-labels.idx1-ubyte"

4.1.1 Variable Documentation

4.1.1.1 CLASSES

```
const short CLASSES = 10
```

Number of classes

4.1.1.2 Ka

```
const short Ka = TYI
```

Number of alpha bases

4.1.1.3 Kb

```
const short Kb = TYO
```

Number of beta bases

4.1.1.4 LEARNING_RATE

```
const float LEARNING_RATE = 0.00005
```

Learning rate

4.1.1.5 NEURONS_IN

```
const short NEURONS_IN = 784
```

Number of input neurons

4.1.1.6 P_RANGE

```
const std::pair<float, float> P_RANGE = { 0, 0.5 }
```

Range of probabilities for spike decoding

4.1.1.7 T

```
const short T = 4
```

Length of the spike train

4.1.1.8 TEST_IMAGES_PATH

```
const string TEST_IMAGES_PATH = "D:\\t10k-images.idx3-ubyte"
```

Test images database path

4.1.1.9 TEST_LABELS_PATH

```
const string TEST_LABELS_PATH = "D:\\t10k-labels.idx1-ubyte"
```

Test labels database path

4.1.1.10 TRAIN_IMAGES_PATH

```
const string TRAIN_IMAGES_PATH = "D:\\train-images.idx3-ubyte"
```

Training images database path

4.1.1.11 TRAIN_LABELS_PATH

```
const string TRAIN_LABELS_PATH = "D:\\train-labels.idx1-ubyte"
```

Training labels database path

4.1.1.12 TYI

```
const short TYI = 3
```

Length of short-term memory window in presynaptic phase

4.1.1.13 TYO

```
const short TYO = 2
```

Length of short-term memory window in feedback phase

4.2 SNN.cpp File Reference

```
#include "stdafx.h"
#include "Network.h"
#include "Utils.h"
#include <time.h>
#include <opencv2/opencv.hpp>
#include "TrainingPool.h"
```

Functions

- int main (int argc, char **argv)

4.2.1 Function Documentation

4.2.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

This is where your program should be written

Index