SOM

0.1.0

Generated by Doxygen 1.8.14

Contents

Chapter 1

SelfOrganizingMaps

Self organizing maps implementation.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 SOM$<$ T $>$ Class Template Reference

Self-Organizing Maps implementation.

`#include <SOM.h>`

Public Member Functions

- SOM (int w, int h, int d)

  Overloaded Constructor. Weights are randomly assigned between [0,1).
- SOM (int w, int h, int d, BMDistType bmdistType, DistanceType distanceType)

  Overloaded Constructor. Weights are randomly assigned between [0,1).
- void train (const std::vector$<$ std::vector$<$ T $>>$ &samples, unsigned int iterations, double s_learn_rate, double f_learn_rate, double neighborhoodSize)

  trains the SOM. If there are less samples than th #N of iterations, then the samples are repeated cyclically.
- std::vector$<$ T $>$ cluster (const std::vector$<$ T $>$ &sample)

  clusters the input sample.
- virtual ∼SOM ()

  Empty destructor.
- T ∗const nodeAt (int i, int j) const

  get node (neuron) weights at given position.
- void setNodeAt (int i, int j, const std::vector$<$ T $>$ &val)

  assign values to weights of the neuron at given indices.
- void load (const std::string &model_path, const SOMFileFormat &ff)

  loads the trained SOM network from the file.
- void save (const std::string &model_path, const SOMFileFormat &ff)

  saves the trained SOM to the file.
- int cols ()

  get #N of columns (width) of SOM lattice
- int rows ()

  get #N of rows (height) of SOM lattice
- int dims ()

  get dimensions (codebook vector size) of SOM
- T calcBestMatchingUnit (const std::vector$<$ T $>$ &sample, int &y, int &x) const

  calculates Best Matching Unit (winning neuron).

Private Member Functions

- T euclideanDistance (const std::vector< T > &v1, const std::vector< T > &v2) const

  calculates Euclidean Distance between 2 vectors.
- T squaredEuclideanDistance (const std::vector< T > &v1, const std::vector< T > &v2) const

  calculates squared euclidean distance between 2 vectors.
- T calcGaussian (T mean, T stdDev, T x) const

  calculates Gaussian function of given x.
- T calcGaussian2D (T meanX, T meanY, T sigmaX, T sigmaY, T x, T y) const

  calculates 2D Gaussian function of given input pair (x,y).
- T calcGaussian2D (int meanX, int meanY, T sigma, int x, int y) const

  calculates 2D Gaussian function of given input pair (x,y). This method uses the same sigma for X and Y dimensions.
- T euclideanDistance (const std::vector< T > &v1, const T ∗v2) const

  calculates Euclidean Distance between 2 vectors. This method overloads () as second parameter is pointer to T for performance reasons.
- T dotProduct (const std::vector< T > &v1, const T ∗v2) const

  calculates Dot Product of 2 vectors.
- T cosineSimilarity (const std::vector< T > &v1, const T ∗v2) const

  calculates cosine similarity of 2 vectors.
- T L2norm (const std::vector< T > &v1)

  calculates L2 norm of a vector

Private Attributes

- BMDistType bmdistType

  Best Matching Unit neighbour distance update type of the SOM.
- DistanceType distanceType

  distance metric that is used when BMU is calculated see ()
- int W

  grid width
- int H

  grid height
- int D

  size of the weight vector of the each node.
- std::vector< T > weights

  weights / nodes of SOM

Friends

- YAML::Emitter & **operator<<** (YAML::Emitter &out, const SOM< T > &som)
- void **operator>>** (const YAML::Node &node, SOM< T > &som)

4.1.1 Detailed Description

template<class T>
class SOM< T >

Self-Organizing Maps implementation.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 SOM() [1/2]

```
template<class T>
SOM< T >::SOM (
              int w,
              int h,
              int d )  [inline]
```

Overloaded Constructor. Weights are randomly assigned between [0,1).

Parameters

w	width
h	height
d	#N of dimensions (codebook size).

4.1.2.2 SOM() [2/2]

```
template<class T>
SOM< T >::SOM (
              int w,
              int h,
              int d,
              BMDistType bmdistType,
              DistanceType distanceType )  [inline]
```

Overloaded Constructor. Weights are randomly assigned between [0,1).

Parameters

w	Width.
h	Height.
d	#N of Dimensions.
bmdistType	BMU update coefficients type.
distanceType	Distance metric type to use.

4.1.2.3 ∼SOM()

```
template<class T>
virtual SOM< T >::∼SOM ( )  [inline], [virtual]
```

Empty destructor.

4.1.3 Member Function Documentation

4.1.3.1 calcBestMatchingUnit()

```
template<class T>
T SOM< T >::calcBestMatchingUnit (
             const std::vector< T > & sample,
             int & y,
             int & x ) const  [inline]
```

calculates Best Matching Unit (winning neuron).

Parameters

sample	input sample
y	index of the 0th dimension (rows) of the winning neuron
x	index of the 1th dimension (columns) of the winning neuron

Returns

distance between BMU and sample

4.1.3.2 calcGaussian()

```
template<class T>
T SOM< T >::calcGaussian (
             T mean,
             T stdDev,
             T x ) const  [inline], [private]
```

calculates Gaussian function of given x.

$$f(x) = \frac{1}{(\sigma\sqrt{(2\pi)}}e^{(\frac{-(x-\mu)^2}{2\sigma^2})}$$

Parameters

mean	mean of the Gaussian distribution
stdDev	standarad deviation
x	input value

Returns

Gaussian function of the given x.

4.1.3.3 calcGaussian2D() [1/2]

```
template<class T>
T SOM< T >::calcGaussian2D (
             T meanX,
             T meanY,
             T sigmaX,
             T sigmaY,
             T x,
             T y ) const  [inline], [private]
```

calculates 2D Gaussian function of given input pair (x,y).

$f(x,y) = \frac{1}{(2\pi\sigma_x\sigma_y)} e^{(-[(x-\mu_x)^2/(2\sigma_x^2)+(y-\mu_y)^2/(2\sigma_y^2)])}$.

Parameters

meanX	mean value in X dimension.
meanY	mean value in Y dimension.
sigmaX	standarad deviation in X dimension.
sigmaY	standarad deviation in Y dimension.
x	input value (X dimension).
y	input value (Y dimension).

Returns

2d gaussian function value of the given (x,y) pair.

4.1.3.4 calcGaussian2D() [2/2]

```
template<class T>
T SOM< T >::calcGaussian2D (
             int meanX,
             int meanY,
             T sigma,
             int x,
             int y ) const  [inline], [private]
```

calculates 2D Gaussian function of given input pair (x,y). This method uses the same sigma for X and Y dimensions.

$f(x,y) = \frac{1}{(2\pi\sigma^2)} e^{(-[(x-\mu_x)^2+(y-\mu_y)^2]/(2\sigma^2))}$.

Parameters

meanX	mean value in X dimension.
meanY	mean value in Y dimension.
sigma	standarad deviation (sigmaVector=[sigma, sigma])
x	input value (X dimension).
y	input value (Y dimension).

Returns

2d gaussian function value of the given (x,y) pair.

4.1.3.5 cluster()

```
template<class T>
std::vector<T> SOM< T >::cluster (
             const std::vector< T > & sample )  [inline]
```

clusters the input sample.

Parameters

sample	input sample

Returns

Winner neuron's weight vector, which corresponds to the most similar weights to input pattern.

4.1.3.6 cols()

```
template<class T>
int SOM< T >::cols ( )  [inline]
```

get #N of columns (width) of SOM lattice

Returns

4.1.3.7 cosineSimilarity()

```
template<class T>
T SOM< T >::cosineSimilarity (
             const std::vector< T > & v1,
             const T * v2 ) const  [inline], [private]
```

calculates cosine similarity of 2 vectors.

Parameters

v1	vector 1
v2	vector 2

Returns

resulting scalar value of cosine similarity

4.1.3.8 dims()

```
template<class T>
int SOM< T >::dims ( )  [inline]
```

get dimensions (codebook vector size) of SOM

Returns

4.1.3.9 dotProduct()

```
template<class T>
T SOM< T >::dotProduct (
             const std::vector< T > & v1,
             const T * v2 ) const  [inline], [private]
```

calculates Dot Product of 2 vectors.

Parameters

v1	vector 1
v2	vector 2

Returns

resulting scalar value of dot product

4.1.3.10 euclideanDistance() [1/2]

```
template<class T>
T SOM< T >::euclideanDistance (
             const std::vector< T > & v1,
             const std::vector< T > & v2 ) const  [inline], [private]
```

calculates Euclidean Distance between 2 vectors.

Parameters

v1	vector 1
v2	vector 2

Returns

euclidean distance

4.1.3.11 euclideanDistance() [2/2]

```
template<class T>
T SOM< T >::euclideanDistance (
            const std::vector< T > & v1,
            const T * v2 ) const  [inline], [private]
```

calculates Euclidean Distance between 2 vectors. This method overloads () as second parameter is pointer to T for performance reasons.

Parameters

v1	vector 1
v2	vector 2

Returns

euclidean distance

4.1.3.12 L2norm()

```
template<class T>
T SOM< T >::L2norm (
            const std::vector< T > & v1 )  [inline], [private]
```

calculates L2 norm of a vector

Parameters

v1	input vector

Returns

scalar value of L2 norm.

4.1.3.13 load()

```
template<class T>
void SOM< T >::load (
             const std::string & model_path,
             const SOMFileFormat & ff )  [inline]
```

loads the trained SOM network from the file.

Parameters

model_path	model path

///

Parameters

ff	file format

4.1.3.14 nodeAt()

```
template<class T>
T* const SOM< T >::nodeAt (
             int i,
             int j ) const  [inline]
```

get node (neuron) weights at given position.

Parameters

i	index of the first dimension (rows) of the SOM lattice.
j	index of the second dimension (columns) of the SOM lattice.

Returns

returns the pointer to Type T, which is the first element in the weight (codebook) vector of the corresponding SOM node.

4.1.3.15 rows()

```
template<class T>
int SOM< T >::rows ( )  [inline]
```

get #N of rows (height) of SOM lattice

Returns

4.1.3.16 save()

```
template<class T>
void SOM< T >::save (
            const std::string & model_path,
            const SOMFileFormat & ff ) [inline]
```

saves the trained SOM to the file.

Parameters

model_path	model file path
ff	file format

4.1.3.17 setNodeAt()

```
template<class T>
void SOM< T >::setNodeAt (
            int i,
            int j,
            const std::vector< T > & val ) [inline]
```

assign values to weights of the neuron at given indices.

Parameters

i	index at 0th dimension (rows)
j	index at 1th dimension (columns)
val	value to set.

4.1.3.18 squaredEuclideanDistance()

```
template<class T>
T SOM< T >::squaredEuclideanDistance (
            const std::vector< T > & v1,
            const std::vector< T > & v2 ) const [inline], [private]
```

calculates squared euclidean distance between 2 vectors.

Parameters

v1	vector 1
v2	vector 2

Returns

euclidean distance

4.1.3.19 train()

```
template<class T>
void SOM< T >::train (
             const std::vector< std::vector< T >> & samples,
             unsigned int iterations,
             double s_learn_rate,
             double f_learn_rate,
             double neighborhoodSize )  [inline]
```

trains the SOM. If there are less samples than th #N of iterations, then the samples are repeated cyclically.

Parameters

samples	training samples with size of N∗D where N is the number of samples and D is the number of dimensions of SOM.
iterations	#N of iterations
s_learn_rate	starting learning_rate
f_learn_rate	ending learning_rate
neighborhoodSize	neighborhood size, currently only sqare neighborhood is supported.

4.1.4 Member Data Documentation

4.1.4.1 bmdistType

```
template<class T>
BMDistType SOM< T >::bmdistType  [private]
```

Best Matching Unit neighbour distance update type of the SOM.

4.1.4.2 D

```
template<class T>
int SOM< T >::D  [private]
```

size of the weight vector of the each node.

4.1.4.3 distanceType

```
template<class T>
DistanceType SOM< T >::distanceType  [private]
```

distance metric that is used when BMU is calculated see ()

4.1.4.4 H

```
template<class T>
int SOM< T >::H  [private]
```

grid height

4.1.4.5 W

```
template<class T>
int SOM< T >::W  [private]
```

grid width

4.1.4.6 weights

```
template<class T>
std::vector<T> SOM< T >::weights  [private]
```

weights / nodes of SOM

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

- SOM.h

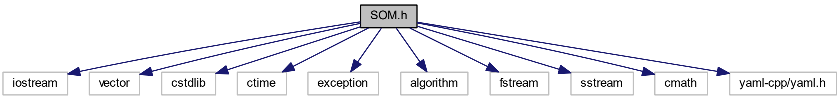
Chapter 5

File Documentation

5.1 SOM.h File Reference

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <exception>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <cmath>
#include <yaml-cpp/yaml.h>
```

Include dependency graph for SOM.h:



Classes

- class SOM< T >

  Self-Organizing Maps implementation.

Macros

- #define **ENABLE_ROCKSDB** 0
- #define **M_PI** (3.14159265358979323846)

Enumerations

- enum SOMFileFormat : unsigned char { SOMFileFormat::YAML = 0, SOMFileFormat::ROCKSDB = 1 }

  supported file formats for SOM
- enum BMDistType : unsigned char { BMDistType::Uniform = 0, BMDistType::ExpDecay = 1, BMDistType::Gaussian = 2 }

  Distribution types for BMU neighborhood update coefficients.
- enum DistanceType : unsigned char { DistanceType::Euclidean = 0, DistanceType::DotProduct = 1, DistanceType::CosineSimiarity = 2, DistanceType::SquaredEuclidean = 3 }

  Distance metrics

5.1.1 Enumeration Type Documentation

5.1.1.1 BMDistType

```
enum BMDistType : unsigned char [strong]
```

Distribution types for BMU neighborhood update coefficients.

Enumerator

Uniform	Same coeffs for BMU and all of its neighborhoods.
ExpDecay	exponential decay
Gaussian	Gaussian distribution

5.1.1.2 DistanceType

```
enum DistanceType : unsigned char [strong]
```

Distance metrics

Enumerator

Euclidean	Euclidean distance: For 2D, the distance between (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. General form (L^2 norm) for the vectors A, B with the size n is calculated as: $$\sqrt{\sum_{i=1}^{n}(A_i - B_i)^2}$$
DotProduct	Dot Product: General form for the vectors A, B with a size n: $$\sum_{i=1}^{n}(A_i * B_i)$$

Enumerator

CosineSimiarity	Cosine Simiarity: General form for the vectors A, B with a size n : $$\text{similarity} = \cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}}$$
SquaredEuclidean	Squared Euclidean: For 2D, the distance between (x_1, y_1) and (x_2, y_2) is $(x_2 - x_1)^2 + (y_2 - y_1)^2$. General form for the vectors A, B with a size n is calculated as: $$\sum_{i=1}^{n}(A_i - B_i)^2$$

5.1.1.3 SOMFileFormat

```
enum SOMFileFormat : unsigned char [strong]
```

supported file formats for SOM

Enumerator

YAML	yaml file format
ROCKSDB	RocksDB DB format.

Index