C++:Self-Organizing-Map:CUDA

0.0.1

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Namespace Documentation

3.1 SOM Namespace Reference

The namespace of SOM related classes, methods and structures.

Classes

- struct configuration

  Structure that provides the configuration of the SOM.
- class SelfOrganizingMap

  The class that implements the Self Organizing Map model.

Typedefs

- typedef std::vector< std::vector< digit > > **SOMContainer**

Functions

- int classify (std::vector< SelfOrganizingMap > &maps, const digit &sample)

  Classifies a sample image into one of the categories represented by the maps.

3.1.1 Detailed Description

The namespace of SOM related classes, methods and structures.

This namespace contains the SelfOrganizingMap class, a structure that configures the training of it, and a method used to classify query points.

3.1.2 Function Documentation

3.1.2.1 classify()

```
int SOM::classify (
            std::vector< SelfOrganizingMap > & maps,
            const digit & sample )
```

Classifies a sample image into one of the categories represented by the maps.

This method claculates the distance from the sample point to every map. The sample is then classified according to the class of the map to which it is closest.

Parameters

maps	The SOMs that were trained on distinct classes of the dataset.
sample	The query point that needs to be classified

Chapter 4

Class Documentation

4.1 SOM::configuration Struct Reference

Structure that provides the configuration of the SOM.

Public Member Functions

- void printConfiguration (std::ostream &out)

  Prints all of the configuration parameters into an output stream.

Public Attributes

- int digitW
- int digitH
- int mapW
- int mapH
- int maxT
- bool animation
- std::string animationPath
- int frameCount
- bool classification
- int classCount

4.1.1 Detailed Description

Structure that provides the configuration of the SOM.

4.1.2 Member Data Documentation

4.1.2.1 animation

```
bool SOM::configuration::animation
```

This flags that the user wants to save intermediate states during training.

4.1.2.2 animationPath

```
std::string SOM::configuration::animationPath
```

The path to save resulting animation frames to.

4.1.2.3 classCount

```
int SOM::configuration::classCount
```

The number of classes into which the images have to be categorized in.

4.1.2.4 classification

```
bool SOM::configuration::classification
```

This flags that classification procedure is enabled.

4.1.2.5 digitH

```
int SOM::configuration::digitH
```

Height of the input image in pixels

4.1.2.6 digitW

```
int SOM::configuration::digitW
```

Width of the input image in pixels

4.1.2.7 frameCount

```
int SOM::configuration::frameCount
```

The number of frames that need to be saved.

4.1.2.8 mapH

```
int SOM::configuration::mapH
```

Height of the SOM structure.

4.1.2.9 mapW

```
int SOM::configuration::mapW
```

Width of the SOM structure.

4.1.2.10 maxT

```
int SOM::configuration::maxT
```

Number of maximum iterations during training.

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

- src/SOM.cpp

4.2 digit Class Reference

Public Member Functions

- int **dimension** () const
- **digit** (int width=8, int height=8)
- **digit** (const digit &other)
- void **operator=** (const digit &other)
- int **getWidth** () const
- int **getHeight** () const
- double * **getShades** () const
- void **initrandom** (double min, double max)
- std::ostream & **appendToFile** (std::ostream &out, std::function< double(double)> &&mapped↩ Shade=[ ](double shade) { return shade;})
- digit **operator+** (const digit &other) const
- digit **minus** (const digit &other) const
- digit **operator*** (double scalar) const
- double **getMaxAbsShade** ()
- double **getMaxShade** ()
- double **getMinShade** ()
- void **minMaxNormalize** (double min, double max)
- double **operator[ ]** (int index) const
- int **getValue** () const
- double **operator-** (const digit &other) const

Friends

- std::istream & **operator>>** (std::istream &in, digit &other)
- std::ostream & **operator<<** (std::ostream &out, const digit &other)

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

- src/digit.cpp

4.3 digitSet Class Reference

Public Member Functions

- **digitSet** (int width, int height=1)
- void **add** (const digit &d)
- std::pair< double, double > **minMaxFeatureScale** ()
- **digitSet** (const digitSet &other)
- int **getWidth** () const
- int **getHeight** () const
- const digit & **getDigit** (int index) const
- const std::vector< digit > & **getDigits** () const
- int **size** () const
- int **dimension** () const

Protected Attributes

- std::vector< digit > **_digits**
- int **_width**
- int **_height**

Friends

- std::istream & **operator>>** (std::istream &in, digitSet &other)
- std::ostream & **operator<<** (std::ostream &out, digitSet &other)

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

- src/digitSet.cpp

4.4 concurrent::Output< T > Class Template Reference

Public Member Functions

- void **push** (const T &element)
- T **pop** ()
- unsigned **size** ()

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

- src/parallel.h

4.5 concurrent::PARALLEL< Func, job > Class Template Reference

Public Member Functions

- **PARALLEL** (const Func &func, std::queue< job > &jobs, int NUM_THREADS)
- void **pstart** (bool &over)
- void **pstart** ()

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

- src/parallel.h

4.6 SOM::SelfOrganizingMap Class Reference

The class that implements the Self Organizing Map model.

Public Member Functions

- SelfOrganizingMap (const digitSet &points, int mapWidth, int mapHeight)
- SelfOrganizingMap (const SelfOrganizingMap &other)
- void setup_CUDA (int sampleDim, int sampleCount)
- void copy_samples_to_device (int sampleDim)

    Copies the samples from the host memory to CUDA device memory.
- void copy_map_to_device (int dim, int mapWidth, int mapHeight)

    Copies the map from the host memory to CUDA device memory.
- void copy_map_from_device (int sampleDim, int mapWidth, int mapHeight)

    Copies the resulting trained map from the CUDA device back to the host memory.
- ∼SelfOrganizingMap ()

    Deallocates all of the allocated memory by the class on the CUDA device.
- void initializeRandomSOM (SOM::SOMContainer &som, int dimx, int dimy)

    Initializes the map in a random manner.
- void initializeSampledSOM (SOM::SOMContainer &som,, int dimx, int dimy)

    Initializes the map in a random manner.
- bool **safebound** (int i, int j)
- double normal_pdf (double x, double m, double s)
- double euclideanDistance (int i1, int j1, int i2, int j2)
- double normalNeighbourCoefficient (int i1, int j1, int i2, int j2, double radius)
- SOM::SOMContainer & getMap ()

    Returns the internal map.
- void train (int maxT, std::function< void(int, int, SelfOrganizingMap ∗)> &&everyFewIterations=[ ](int, int, SelfOrganizingMap ∗) {})

    The training method of the map. Results in a trained model on the host memory side.
- digit getClosestSample (int i, int j)

    Searches for the sample that is closest to a given neuron within the training data (host side).
- void printMap ()

    Prints the labeling of the neurons according to the category of their closest sample.
- void printMapToStream (std::ostream &out)

    Prints the map to an output stream.
- double getClosestPrototypeDistance (const digit &sample)

    Calculates the distance from a sample to its best matching unit.

4.6.1 Detailed Description

The class that implements the Self Organizing Map model.

This class encapsulates a dataset of training points on which the model is trained and the actual map. The CUDA implementation copies the trainig points and the map to the device memory and operates on them inplace. After the training finished, the trained map is copied back to the host user memory.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 SelfOrganizingMap() [1/2]

```
SOM::SelfOrganizingMap::SelfOrganizingMap (
             const digitSet & points,
             int mapWidth,
             int mapHeight )  [inline]
```

The constructor takes a set of points and the dimensions of the map as parameter. It normalizes the inputs in order to prevent the explosion of gradients. Afterwards it initializes the map randomly, allocates memory on the CUDA device for the samples and the map and copies them there.

Parameters

points	The training points on which the map will be trained.
mapWidth	The width of the map.
mapHeight	The height of the map.

4.6.2.2 SelfOrganizingMap() [2/2]

```
SOM::SelfOrganizingMap::SelfOrganizingMap (
             const SelfOrganizingMap & other )  [inline]
```

The copy constructor of the class copies the samples and map both from the host memory and the CUDA device memory.

4.6.3 Member Function Documentation

4.6.3.1 euclideanDistance()

```
double SOM::SelfOrganizingMap::euclideanDistance (
             int i1,
             int j1,
             int i2,
             int j2 )  [inline]
```

The euclidean distance between two 2D points.

4.6.3.2 initializeRandomSOM()

```
void SOM::SelfOrganizingMap::initializeRandomSOM (
             SOM::SOMContainer & som,
             int dimx,
             int dimy )  [inline]
```

Initializes the map in a random manner.

Initializes each representative node within the map using random values within the minimum and maximum values seen in the sample dataset.

Parameters

som	The map that needs to be initialized.
dimx	The width of the images in pixels.
dimy	The height of the images in pixels.

4.6.3.3 initializeSampledSOM()

```
void SOM::SelfOrganizingMap::initializeSampledSOM (
             SOM::SOMContainer & som,
             int dimx,
             int dimy )  [inline]
```

Initializes the map in a random manner.

Initializes each representative node within the map using random samples from the training dataset.

Parameters

som	The map that needs to be initialized.
dimx	The width of the images in pixels.
dimy	The height of the images in pixels.

4.6.3.4 normal_pdf()

```
double SOM::SelfOrganizingMap::normal_pdf (
             double x,
             double m,
             double s )  [inline]
```

Return the value of x in the kernel of the Gaussian Probability Density Function.

Parameters

x	The value at which the kernel needs to be evaluated.
m	The mean of the kernel.
s	The sigma of the kernel.

4.6.3.5 normalNeighbourCoefficient()

```
double SOM::SelfOrganizingMap::normalNeighbourCoefficient (
             int i1,
             int j1,
             int i2,
             int j2,
             double radius )  [inline]
```

Calculates the neighbouring coefficients with a given radius from point (i1,j1) to point (i2,j2).

Parameters

i1	The y index of the first neuron
j1	The x index of the first neuron
i2	The y index of the second neuron
j2	The x index of the second neuron
radius	The radius of the neighbouring function.

4.6.3.6 printMapToStream()

```
void SOM::SelfOrganizingMap::printMapToStream (
             std::ostream & out )  [inline]
```

Prints the map to an output stream.

Prints the dimensions of the map, afterwards it prints the actual neuron weights to an output stream.

4.6.3.7 setup_CUDA()

```
void SOM::SelfOrganizingMap::setup_CUDA (
             int sampleDim,
             int sampleCount )  [inline]
```

Allocates memory on the CUDA device for the provided samples, the map and the intermediary distances

Parameters

sampleDim	The dimensionality of an input sample
sampleCount	The number of samples provided to the map

4.6.3.8 train()

```
void SOM::SelfOrganizingMap::train (
            int maxT,
            std::function< void(int, int, SelfOrganizingMap *)> && everyFewIterations = [](int, int, SelfOrg
)  [inline]
```

The training method of the map. Results in a trained model on the host memory side.

This method trains the model for a given number of iterations that it receives as parameter. It also receives a callback function in order to facilitate the capturing the frames for the animation. Each iteration it samples a random sample from the dataset and updates the model accordingly. The best matching unit search is parallelized and so is the neighbourhood update phase. After the maximum number of iterations it copies the map from CUDA device to host memory.

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

- src/SOM.cpp