Deep Learning Library - Philipp Bock

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

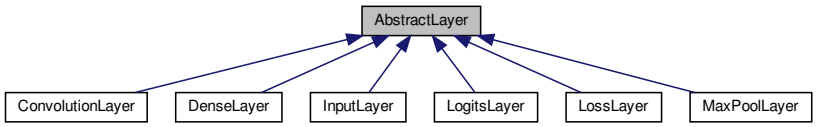
Chapter 3

Class Documentation

3.1 AbstractLayer Class Reference

`#include <AbstractLayer.hpp>`

Inheritance diagram for AbstractLayer:



Public Member Functions

- AbstractLayer (std::shared_ptr< AbstractLayer > input, std::shared_ptr< Graph > computeGraph)
- AbstractLayer (std::shared_ptr< Graph > computeGraph)
- const std::shared_ptr< Node > & getInputNode () const

  the computational graph
- const std::shared_ptr< Node > & **getOutputNode** () const
- void **setOutputNode** (const std::shared_ptr< Node > &outputNode)
- int **getOutputChannels** () const
- void **setOutputChannels** (int outputChannels)
- int **getInputChannels** () const
- int **getOutputSize** () const
- void **setOutputSize** (int outputSize)
- int **getBatchSize** () const
- void **setBatchSize** (int batchSize)
- int **getInputSize** ()
- const std::shared_ptr< Graph > & **getComputeGraph** () const

3.1.1 Detailed Description

Layer Base class, defines what each Layer needs to implement. The main concept is, that each layer has an output node. this way we can connect all nodes of the computational graph by taking the outputNode of the previous Layer as an input node

3.1.2 Constructor & Destructor Documentation

3.1.2.1 AbstractLayer() [1/2]

```
AbstractLayer::AbstractLayer (
            std::shared_ptr< AbstractLayer > input,
            std::shared_ptr< Graph > computeGraph )
```

Creates a new Layer storing the input Layer and the computeGraph using the inputLayer, the input Node, Channels and Batch Size are evaluated

Parameters

input	
computeGraph	

3.1.2.2 AbstractLayer() [2/2]

```
AbstractLayer::AbstractLayer (
            std::shared_ptr< Graph > computeGraph )  [explicit]
```

Creates a new Layer only storing the computeGraph

Parameters

computeGraph	

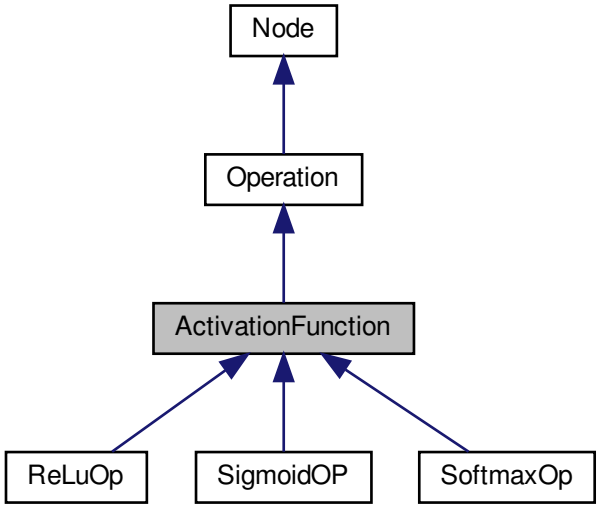
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/include/AbstractLayer.hpp
- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/src/AbstractLayer.cpp

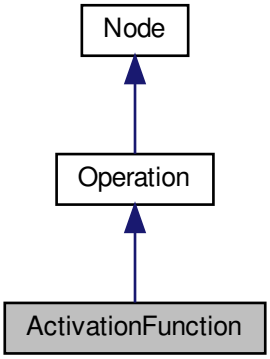
3.2 ActivationFunction Class Reference

```
#include <ActivationFunction.hpp>
```

Inheritance diagram for ActivationFunction:



Collaboration diagram for ActivationFunction:



Public Member Functions

- ActivationFunction (std::shared_ptr< Node > X)

3.2.1 Detailed Description

An ActivationFunction operation object introduces non linearity into the computational graph

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ActivationFunction()

```
ActivationFunction::ActivationFunction (
            std::shared_ptr< Node > X )
```

- creates the operation Object from the input Node
- the channels are simply forwarded from the input Node

Parameters

X	input Node

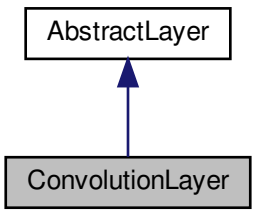
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/ActivationFunction.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/ActivationFunction.cpp

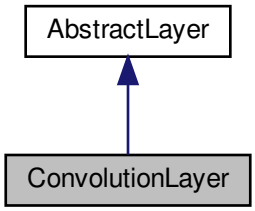
3.3 ConvolutionLayer Class Reference

```
#include <ConvolutionLayer.hpp>
```

Inheritance diagram for ConvolutionLayer:



Collaboration diagram for ConvolutionLayer:



Public Member Functions

- ConvolutionLayer (std::shared_ptr< AbstractLayer > input, std::shared_ptr< Graph > computeGraph, ActivationType activationFunction, int amountFilters, int kernelDim, int stride=1, InitializationType initializationType=InitializationType::Xavier)

3.3.1 Detailed Description

Creates a Convolutional LAyer

3.3.2 Constructor & Destructor Documentation

3.3.2.1 ConvolutionLayer()

```
ConvolutionLayer::ConvolutionLayer (
            std::shared_ptr< AbstractLayer > input,
            std::shared_ptr< Graph > computeGraph,
            ActivationType activationFunction,
            int amountFilters,
            int kernelDim,
            int stride = 1,
            InitializationType initializationType = InitializationType::Xavier )
```

Creates the layer

Parameters

input	the previous Layer
computeGraph	
activationFunction	check commonDatatypes.h for possibilities
amountFilters	
kernelDim	
stride	
initializationType	check commonDatatypes.h for possibilities

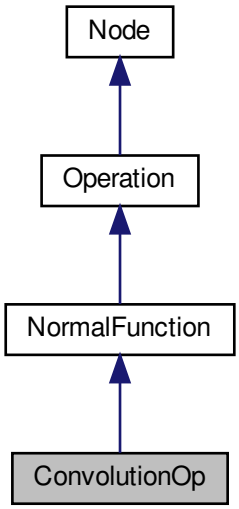
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/include/ConvolutionLayer.hpp
- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/src/ConvolutionLayer.cpp

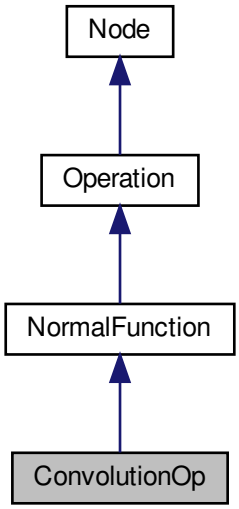
3.4 ConvolutionOp Class Reference

```
#include <ConvolutionOp.hpp>
```

Inheritance diagram for ConvolutionOp:



Collaboration diagram for ConvolutionOp:



Public Member Functions

- ConvolutionOp (std::shared_ptr< Node > X, std::shared_ptr< Parameter > filter, int stride=1)
- void forwardPass () override

  default destructor
- void backwardPass () override
- const Matrix & getIm2Col () const

  caches the calculated _im2Col matrix
- void **setIm2Col** (const Matrix &im2Col)

Static Public Member Functions

- static void forwardsConvolution (const Matrix &miniBatch, const Matrix &filter, Matrix &im2Col, Matrix &outputMatrix, const int stride, const int channels)
- static void backwardsConvolution (Matrix &dX, Matrix &dFilter, const Matrix &filter, const Matrix &dout, const Matrix &im2ColM, int batchSize, int inputDimX, int stride, int channels)
- static void im2col (Matrix &output, const Matrix &input, int kernelSize, int stride, int channel, int batchSize)
- static void col2im (Matrix &output, const Matrix &input, int kernelSize, int origDim, int stride, int channel, int batchSize)

3.4.1 Detailed Description

Implements a Convolution Operation

3.4.2 Constructor & Destructor Documentation

3.4.2.1 ConvolutionOp()

```
ConvolutionOp::ConvolutionOp (
            std::shared_ptr< Node > X,
            std::shared_ptr< Parameter > filter,
            int stride = 1 )
```

- creates a Normal Function and sets the outputChannel to the amount of Filters passed
- checks whether X and filter do have the same amount of channels if not a exception is thrown

Parameters

X	input Noode
filter	all kernels that should be applied. Each row of the forward pass represents one kernel
stride	stride

3.4.3 Member Function Documentation

3.4.3.1 backwardPass()

```
void ConvolutionOp::backwardPass ( )  [override], [virtual]
```

- calculates the gradients of the convolution using getPreviousGradients() and the previous calculated im2Col Matrix

Reimplemented from Operation.

3.4.3.2 backwardsConvolution()

```
void ConvolutionOp::backwardsConvolution (
            Matrix & dX,
            Matrix & dFilter,
            const Matrix & filter,
            const Matrix & dout,
            const Matrix & im2ColM,
            int batchSize,
            int inputDimX,
            int stride,
            int channels )  [static]
```

- calculates the gradients w.r.t X and filter
- dX = filter.transpose() ∗ dout;
- dFilter = dout ∗ im2ColM.transpose();

Parameters

dX	reference to Matrix which should store the gradients w.r.t X
dFilter	reference to Matrix which should store the gradients w.r.t filter
filter	the original filter
dout	reference to Matrix which holds the previously calculated forwardpass
im2ColM	reference to Matrix which holds the previously calculated im2Col Matrix
batchSize	batchSize of X
inputDimX	the dimensions of one sample of the input Node X
stride	original stride
channels	the original channels of X and filter

3.4.3.3 col2im()

```
void ConvolutionOp::col2im (
            Matrix & output,
            const Matrix & input,
            int kernelSize,
            int origDim,
            int stride,
            int channel,
            int batchSize )  [static]
```

Implements the col2Im function see also https://leonardoaraujosantos.gitbooks.io/artificial-inteligen
_faster.html

Parameters

output	reference to Matrix which should store the calculated col2Im matrix
input	a im2Col matrix
kernelSize	
origDim	the original Dimensions of a sample of the im2Col inputMatrix
stride	
channel	
batchSize	

3.4.3.4 forwardPass()

```
void ConvolutionOp::forwardPass ( )  [override], [virtual]
```

default destructor

- executes a convolution of the input Nodes X and filter

Reimplemented from Operation.

3.4.3.5 forwardsConvolution()

```
void ConvolutionOp::forwardsConvolution (
            const Matrix & miniBatch,
            const Matrix & filter,
            Matrix & im2Col,
            Matrix & outputMatrix,
            const int stride,
            const int channels )  [static]
```

/

- calculates the dimensions of each sample of the minibatch and each kernel of the filter with the amount of channels
- transforms the miniBatch into the im2Col format and stores it in im2Col
- executes a convolution by multiplying the filter with im2Col and stores it in outputMatrix

Parameters

miniBatch	reference to miniBatch matrix where each row corresponds to one sample, channels are stored consecutively , miniBatch.row(i)=sample = [channel 0]...[channel 1]
filter	rference to miniBatch matrix where each row corresponds to one kernel, channels are stored consecutively filter.row(i) = kernel = [channel 0]...[channel 1]
im2Col	reference to Matrix which should store the im2Col output
outputMatrix	reference to Matrix which should store the convolution output of miniBatch with filter
stride	stride
channels	input channels of filter /miniBatch

3.4.3.6 im2col()

```
void ConvolutionOp::im2col (
            Matrix & output,
            const Matrix & input,
            int kernelSize,
            int stride,
            int channel,
            int batchSize )  [static]
```

Implements the im2Col function see also https://leonardoaraujosantos.gitbooks.io/artificial-inteligen
_faster.html

Parameters

output	reference to Matrix which should store the calculated im2Col matrix
input	the matrix that should be transformed
kernelSize	
stride	
channel	
batchSize	

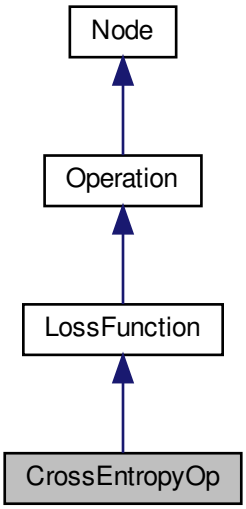
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/ConvolutionOp.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/ConvolutionOp.cpp

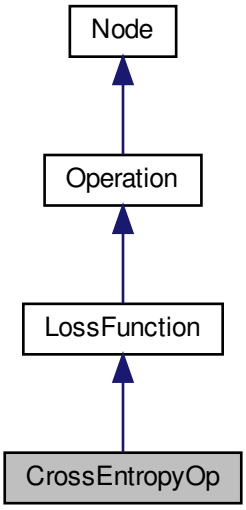
3.5 CrossEntropyOp Class Reference

```
#include <CrossEntropyOp.hpp>
```

Inheritance diagram for CrossEntropyOp:



Collaboration diagram for CrossEntropyOp:



Public Member Functions

- CrossEntropyOp (std::shared_ptr< Node > X, std::shared_ptr< Placeholder > labels)
- void forwardPass () override

  default destructor
- void backwardPass () override

3.5.1 Detailed Description

Implements the CrossEntropy Loss

3.5.2 Constructor & Destructor Documentation

3.5.2.1 CrossEntropyOp()

```
CrossEntropyOp::CrossEntropyOp (
            std::shared_ptr< Node > X,
            std::shared_ptr< Placeholder > labels )
```

creates an LossFunction object stores the labels which are need to calculate the loss

Parameters

X	softmax classified samples
labels	groundTruth for initial samples

3.5.3 Member Function Documentation

3.5.3.1 backwardPass()

```
void CrossEntropyOp::backwardPass ( )  [override], [virtual]
```

calculates the softmax-crossEntropy-classifier gradient and passes it further to its input Node which must be a Softmax operation see also: https://deepnotes.io/softmax-crossentropy

Reimplemented from Operation.

3.5.3.2 forwardPass()

```
void CrossEntropyOp::forwardPass ( )  [override], [virtual]
```

default destructor

calculates the cross-entropy loss and set the output as forward pass value

Reimplemented from Operation.

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

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/CrossEntropyOp.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/CrossEntropyOp.cpp

3.6 DataInitialization Class Reference

Static Public Member Functions

- static Eigen::MatrixXf **generateRandomMatrix** (int rows, int cols)
- static Eigen::MatrixXf **generateRandomMatrixNormalDistribution** (float mean, float stdev, int rows, int cols)

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

- /home/pbo/CLionProjects/libdl/library/Utils/include/DataInitialization.hpp
- /home/pbo/CLionProjects/libdl/library/Utils/src/DataInitialization.cpp

3.7 DataSet Struct Reference

`#include <commonDatatypes.hpp>`

Public Member Functions

- **DataSet** (std::vector< Matrix > trainingSamples, std::vector< Matrix > trainingLabels)
- **DataSet** (std::vector< Matrix > trainingSamples, std::vector< Matrix > trainingLabels, std::vector< Matrix > validationSamples, std::vector< Matrix > validationLabels)

Public Attributes

- std::vector< Matrix > **_trainingSamples** {}
- std::vector< Matrix > **_trainingLabels** {}
- std::vector< Matrix > **_validationSamples** {}
- std::vector< Matrix > **_validationLabels** {}

3.7.1 Detailed Description

This struct holds four vectors. The first two holding the data and labels of the training set The second two hild the data and labels of the validation set

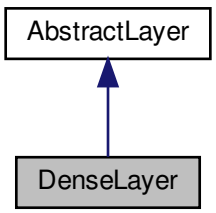
The documentation for this struct was generated from the following file:

- /home/pbo/CLionProjects/libdl/library/Utils/include/commonDatatypes.hpp

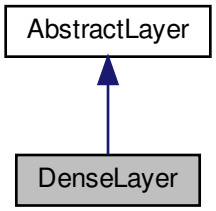
3.8 DenseLayer Class Reference

`#include <DenseLayer.hpp>`

Inheritance diagram for DenseLayer:



Collaboration diagram for DenseLayer:



Public Member Functions

- DenseLayer (std::shared_ptr< AbstractLayer > input, std::shared_ptr< Graph > computeGraph, ActivationType activationFunction, int amountNeurons, InitializationType initializationType=InitializationType::Xavier)

3.8.1 Detailed Description

Creates a DenseLayer

3.8.2 Constructor & Destructor Documentation

3.8.2.1 DenseLayer()

```
DenseLayer::DenseLayer (
            std::shared_ptr< AbstractLayer > input,
            std::shared_ptr< Graph > computeGraph,
            ActivationType activationFunction,
            int amountNeurons,
            InitializationType initializationType = InitializationType::Xavier )
```

Create the Layer

Parameters

input	
computeGraph	
activationFunction	check commonDatatypes.h for possibilities
amountNeurons	
initializationType	check commonDatatypes.h for possibilities

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

- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/include/DenseLayer.hpp
- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/src/DenseLayer.cpp

3.9 Graph Class Reference

`#include <Graph.hpp>`

Public Member Functions

- void addParameter (std::shared_ptr< Parameter > parameters)
- void addOperation (std::shared_ptr< Operation > operation)
- void computeForward ()
- void computeBackwards ()
- void updateParameters (HyperParameters params=HyperParameters())
- bool writeParameters (std::string dir)
- bool readParameters (std::string dir)
- const std::shared_ptr< Placeholder > & getInput () const

  contains all operations of the graph ordered according to their position in the graph
- void **setInput** (const std::shared_ptr< Placeholder > &input)
- const std::shared_ptr< Placeholder > & **getLabels** () const
- void **setLabels** (const std::shared_ptr< Placeholder > &labels)

3.9.1 Detailed Description

This class controls the Behaviour of our list of Nodes aka Computational Graph.

3.9.2 Member Function Documentation

3.9.2.1 addOperation()

```
void Graph::addOperation (
            std::shared_ptr< Operation > operation )
```

Adds an parameter to the list of parameters ! the input Node of the passed operation must already be in _operations List or stored as label. Otherwise an exception is thrown!

Parameters

operation	

3.9.2.2 addParameter()

```
void Graph::addParameter (
            std::shared_ptr< Parameter > parameters )
```

Adds an parameter to the list of parameters

Parameters

variable	

3.9.2.3 computeBackwards()

```
void Graph::computeBackwards ( )
```

First sets the previousGradient for the last Operation to One. Second it iterates through the reversed list of Operations and executes backwardsPass() on each. !this expects that each following Operation holds the previous Operation as input!

3.9.2.4 computeForward()

```
void Graph::computeForward ( )
```

iterates through the list of Operations and executes forwardPass() on each. !this expects that each following Operation holds the previous Operation as input!

3.9.2.5 readParameters()

```
bool Graph::readParameters (
            std::string dir )
```

Reads all Parameters from path dir

Parameters

dir	path/to/Parameter/files/Networkname

Returns

success

3.9.2.6 updateParameters()

```
void Graph::updateParameters (
            HyperParameters params = HyperParameters() )
```

This function may first be called after execution of the Backward Pass It uses the gradients of computation w.r.t Parameter to update its forward pass value.

Parameters

params	a HyperParameters object, specifying things like learningRate, BatchSize and which Optimizer should be used

3.9.2.7 writeParameters()

```
bool Graph::writeParameters (
            std::string dir )
```

Writes all Parameters to path dir

Parameters

dir	path/to/Parameter/files/Networkname

Returns

success

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

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/include/Graph.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/src/Graph.cpp

3.10 HyperParameters Struct Reference

```
#include <commonDatatypes.hpp>
```

Public Member Functions

- **HyperParameters** (int epochs=10, int batchSize=8, float learningRate=0.01, Optimizer optimizer=Optimizer↩ ::Adam, float beta1=0.9, float beta2=0.999)
- std::string **toString** ()

Public Attributes

- float **_learningRate**
- float **_beta1**
- float **_beta2**
- int **_batchSize**
- int **_epochs**
- Optimizer **_optimizer**

3.10.1 Detailed Description

This struct holds all kind of possible Hyper parameters, that may be used to update an Parameter during training

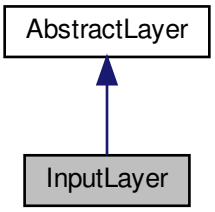
The documentation for this struct was generated from the following file:

- /home/pbo/CLionProjects/libdl/library/Utils/include/commonDatatypes.hpp

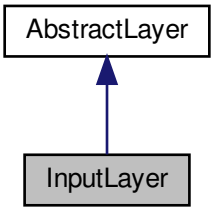
3.11 InputLayer Class Reference

`#include <InputLayer.hpp>`

Inheritance diagram for InputLayer:



Collaboration diagram for InputLayer:



Public Member Functions

- InputLayer (std::shared_ptr< Graph > computeGraph, int batchSize, int dataPoints, int channel)
- void updateX (Matrix &newMiniBatch)

  default destructor

3.11.1 Detailed Description

Creates an InputLayer this is the only Layer that does not take an input Layer as it is the starting Point

3.11.2 Constructor & Destructor Documentation

3.11.2.1 InputLayer()

```
InputLayer::InputLayer (
            std::shared_ptr< Graph > computeGraph,
            int batchSize,
            int dataPoints,
            int channel )
```

Parameters

computeGraph	
batchSize	
dataPoints	amount of data one sample has
channel	channels of the input

3.11.3 Member Function Documentation

3.11.3.1 updateX()

```
void InputLayer::updateX (
            Matrix & newMiniBatch )
```

default destructor

During Training or Prediction this method should be used to set a batch for calculation

Parameters

newMiniBatch	the batch to be predicted, to be trained on

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

- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/include/InputLayer.hpp
- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/src/InputLayer.cpp

3.12 LegoDataLoader Class Reference

Static Public Member Functions

- static void **getData** (int samples, std::vector< std::pair< std::string, int >> shuffledFiles, DataSet &data)
- static std::vector< std::pair< std::string, int > > **shuffleData** (const std::string &dataDir)

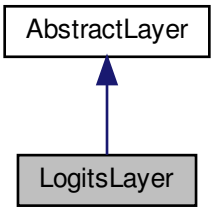
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/Utils/include/LegoDataLoader.hpp
- /home/pbo/CLionProjects/libdl/library/Utils/src/LegoDataLoader.cpp

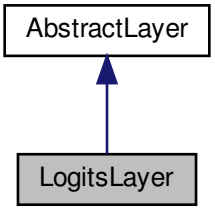
3.13 LogitsLayer Class Reference

`#include <LogitsLayer.hpp>`

Inheritance diagram for LogitsLayer:



Collaboration diagram for LogitsLayer:



Public Member Functions

- LogitsLayer (std::shared_ptr< AbstractLayer > input, std::shared_ptr< Graph > computeGraph, int output↩ Classes)

3.13.1 Detailed Description

Creates a Softmax Classification Layer

3.13.2 Constructor & Destructor Documentation

3.13.2.1 LogitsLayer()

```
LogitsLayer::LogitsLayer (
            std::shared_ptr< AbstractLayer > input,
            std::shared_ptr< Graph > computeGraph,
            int outputClasses )
```

Creates a Logits Layer using the softmax classifier

Parameters

input	
computeGraph	
outputClasses	the number of labels

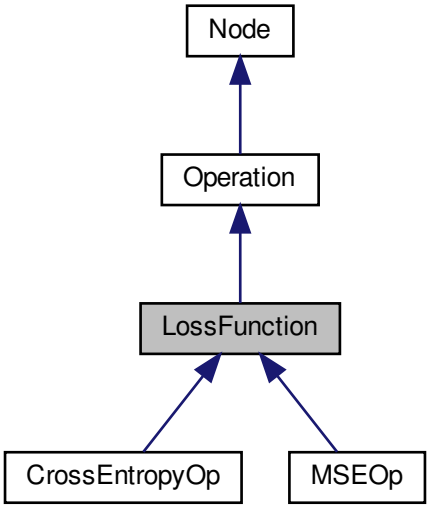
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/include/LogitsLayer.hpp
- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/src/LogitsLayer.cpp

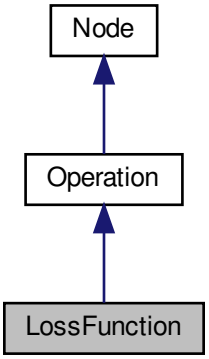
3.14 LossFunction Class Reference

```
#include <LossFunction.hpp>
```

Inheritance diagram for LossFunction:



Collaboration diagram for LossFunction:



Public Member Functions

- LossFunction (std::shared_ptr< Node > X, std::shared_ptr< Placeholder > labels)
- const std::shared_ptr< Placeholder > & getLabels () const

  the ground truth to the input samples
- const Matrix **getPrediction** () const
- const float getLoss () const

  returns the forward pass of the input Node aka the prediction

3.14.1 Detailed Description

A LossFunction operation object represents operation to calculate the loss of a sample computation compared to its ground truth

3.14.2 Constructor & Destructor Documentation

3.14.2.1 LossFunction()

```
LossFunction::LossFunction (
            std::shared_ptr< Node > X,
            std::shared_ptr< Placeholder > labels )
```

- creates the operation Object using the the obligatory Input Node.
- stores the labels for this operation
- sets the output Channel of the operation Object to a random Value, as at this point channels do not matter anymore

Parameters

X	the input Node
labels	the ground truth to the input samples

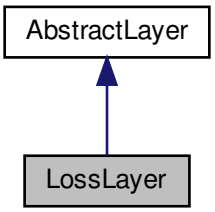
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/LossFunction.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/LossFunction.cpp

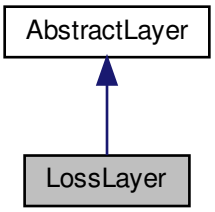
3.15 LossLayer Class Reference

```
#include <LossLayer.hpp>
```

Inheritance diagram for LossLayer:



Collaboration diagram for LossLayer:



Public Member Functions

- LossLayer (std::shared_ptr< AbstractLayer > input, std::shared_ptr< Graph > computeGraph, LossType losstype)
- void updateLabels (Matrix &newLabels)

  default destructor
- const Matrix getPrediction () const
- float getLoss ()

3.15.1 Detailed Description

Creates an Layer for the loss calculation

3.15.2 Constructor & Destructor Documentation

3.15.2.1 LossLayer()

```
LossLayer::LossLayer (
            std::shared_ptr< AbstractLayer > input,
            std::shared_ptr< Graph > computeGraph,
            LossType losstype )
```

Creates the selected loss function and sets it as outputNode

Parameters

input	
computeGraph	
losstype	

3.15.3 Member Function Documentation

3.15.3.1 getLoss()

```
float LossLayer::getLoss ( )
```

throws a runtime_error if no training or prediction has yet been done

Returns

the current Loss

3.15.3.2 getPrediction()

```
const Matrix LossLayer::getPrediction ( ) const
```

throws a runtime_error if no computation has yet been done

Returns

the current Prediciton

3.15.3.3 updateLabels()

```
void LossLayer::updateLabels (
            Matrix & newLabels )
```

default destructor

During Training this method should be used to set the labels for the loss calculation

Parameters

the	Labels that should be used for training

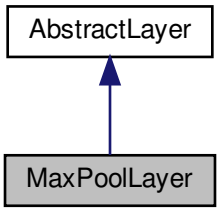
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/include/LossLayer.hpp
- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/src/LossLayer.cpp

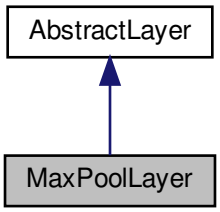
3.16 MaxPoolLayer Class Reference

`#include <MaxPoolLayer.hpp>`

Inheritance diagram for MaxPoolLayer:



Collaboration diagram for MaxPoolLayer:



Public Member Functions

- MaxPoolLayer (const std::shared_ptr< AbstractLayer > &input, std::shared_ptr< Graph > computeGraph, int kernelDim, int stride)

3.16.1 Detailed Description

Creates a max pooling layer

3.16.2 Constructor & Destructor Documentation

3.16.2.1 MaxPoolLayer()

```
MaxPoolLayer::MaxPoolLayer (
            const std::shared_ptr< AbstractLayer > & input,
            std::shared_ptr< Graph > computeGraph,
            int kernelDim,
            int stride )
```

Parameters

input	
computeGraph	
kernelDim	
stride	

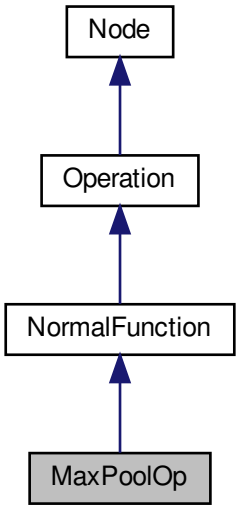
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/include/MaxPoolLayer.hpp
- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/LayerFactory/src/MaxPoolLayer.cpp

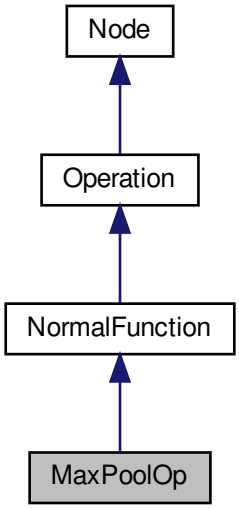
3.17 MaxPoolOp Class Reference

```
#include <MaxPoolOp.hpp>
```

Inheritance diagram for MaxPoolOp:



Collaboration diagram for MaxPoolOp:



Public Member Functions

- MaxPoolOp (std::shared_ptr< Node > X, int windowDim, int stride=1)

- void forwardPass () override

  default destructor
- void backwardPass () override
- const Eigen::MatrixXf & getMaxIndexMatrix () const

  stores the indices of the max Values
- void **setMaxIndexMatrix** (const Eigen::MatrixXf &maxIndexMatrix)

3.17.1 Detailed Description

Implements a MaxPooling operation

3.17.2 Constructor & Destructor Documentation

3.17.2.1 MaxPoolOp()

```
MaxPoolOp::MaxPoolOp (
            std::shared_ptr< Node > X,
            int windowDim,
            int stride = 1 )
```

- creates a Normal Function and sets the outputChannel to the number of channels of the input Node

Parameters

X	input Noode
windowDim	dimensions of the square window that is sliding over the samples taking the max value
stride	stride

3.17.3 Member Function Documentation

3.17.3.1 backwardPass()

```
void MaxPoolOp::backwardPass ( )  [override], [virtual]
```

calculates the gradient w.r.t the input isung the index Matrix

Reimplemented from Operation.

3.17.3.2 forwardPass()

```
void MaxPoolOp::forwardPass ( )  [override], [virtual]
```

default destructor

applies Maxpooling to input X usig the window dimension stores additionaly an index matrix of dim(X) containing at the index of the Max Values a one

Reimplemented from Operation.

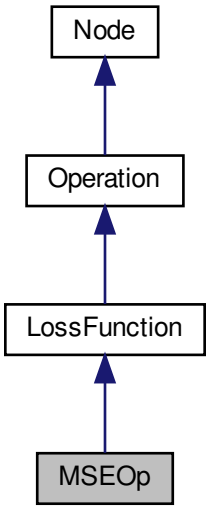
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/MaxPoolOp.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/MaxPoolOp.cpp

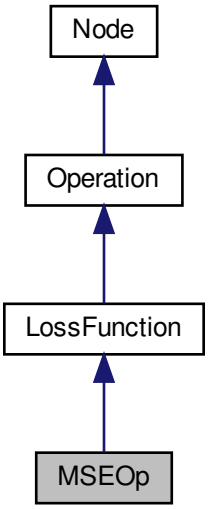
3.18 MSEOp Class Reference

```
#include <MSEOp.hpp>
```

Inheritance diagram for MSEOp:



Collaboration diagram for MSEOp:



Public Member Functions

- MSEOp (std::shared_ptr< Node > X, std::shared_ptr< Placeholder > labels)
- void forwardPass () override

  default destructor
- void backwardPass () override

3.18.1 Detailed Description

Implements the Mean Square Error

3.18.2 Constructor & Destructor Documentation

3.18.2.1 MSEOp()

```
MSEOp::MSEOp (
            std::shared_ptr< Node > X,
            std::shared_ptr< Placeholder > labels )
```

creates an LossFunction object stores the labels which are need to calculate the loss

Parameters

X	softmax classified samples
labels	groundTruth for initial samples

3.18.3 Member Function Documentation

3.18.3.1 backwardPass()

```
void MSEOp::backwardPass ( )  [override], [virtual]
```

calculates the gradient w.r.t to X

Reimplemented from Operation.

3.18.3.2 forwardPass()

```
void MSEOp::forwardPass ( )  [override], [virtual]
```

default destructor

calculates the Mean Square Error of Input X with corresponding labels and sets the output as forward pass value

Reimplemented from Operation.

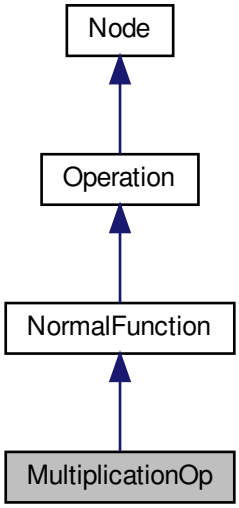
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/MSEOp.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/MSEOp.cpp

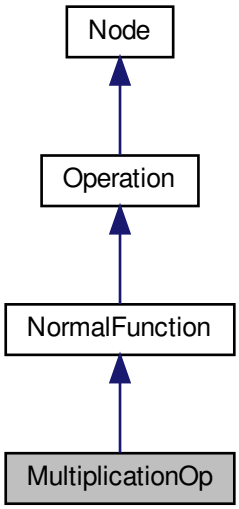
3.19 MultiplicationOp Class Reference

`#include <MultiplicationOp.hpp>`

Inheritance diagram for MultiplicationOp:



Collaboration diagram for MultiplicationOp:



Public Member Functions

- MultiplicationOp (std::shared_ptr< Node > X, std::shared_ptr< Parameter > weights)
- void forwardPass () override
- void backwardPass () override

3.19.1 Detailed Description

Implements the Matrix multiplication that simulates Neurons

3.19.2 Constructor & Destructor Documentation

3.19.2.1 MultiplicationOp()

```
MultiplicationOp::MultiplicationOp (
            std::shared_ptr< Node > X,
            std::shared_ptr< Parameter > weights )
```

creates NormalFunction object and passes the input outputChannesl through stores a node Containing the weights for the multiplication

Parameters

X	input Batch
weights	the amount of cols of the forward pass corresponds to the number of Neurons, the amount of rows must equal the size of one sample of X

3.19.3 Member Function Documentation

3.19.3.1 backwardPass()

```
void MultiplicationOp::backwardPass ( )  [override], [virtual]
```

calculates the gradients w.r.t to X and weights

Reimplemented from Operation.

3.19.3.2 forwardPass()

```
void MultiplicationOp::forwardPass ( )  [override], [virtual]
```

executes the Matrix multiplication and stores the output as forward pass value

Reimplemented from Operation.

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

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/MultiplicationOp.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/MultiplicationOp.cpp

3.20 NeuralNetwork Class Reference

```
#include <NeuralNetwork.hpp>
```

Public Member Functions

- NeuralNetwork (std::shared_ptr< Graph > computeGraph, std::shared_ptr< InputLayer > inputLayer, std::shared_ptr< LossLayer > lossLayer)
- float train (DataSet &data, HyperParameters params, float trainingLossThreshold=1)
- TrainingEvaluation trainAndValidate (DataSet &data, HyperParameters params, float trainingLossThreshold=1)
- void trainBatch (Matrix &miniBatch, Matrix &labels, HyperParameters &params)
- Matrix predictBatch (Matrix &miniBatch, Matrix &labels)
- bool writeParameters (std::string dir, std::string networkName)
- bool readParameters (std::string dir, std::string networkName)
- float getLoss ()

Static Public Member Functions

- static void testAccuracy (Matrix &results, Matrix &labels, float &correct, float &total)
- static std::vector< Matrix > extractBatchList (std::vector< Matrix > &dataset, int batchSize)

3.20.1 Detailed Description

This class implements the DeepLearning logic to run a training or prediction on a set of Layers/operations, specified by a inputLayer, a lossLayer and a computational graph

3.20.2 Constructor & Destructor Documentation

3.20.2.1 NeuralNetwork()

```
NeuralNetwork::NeuralNetwork (
            std::shared_ptr< Graph > computeGraph,
            std::shared_ptr< InputLayer > inputLayer,
            std::shared_ptr< LossLayer > lossLayer )
```

simple construction and initialization of the member variables with provided Parameters

Parameters

computeGraph	
inputLayer	
lossLayer	

3.20.3 Member Function Documentation

3.20.3.1 extractBatchList()

```
std::vector< Matrix > NeuralNetwork::extractBatchList (
            std::vector< Matrix > & dataset,
            int batchSize ) [static]
```

Parameters

dataset	a list of samples
batchSize	

Returns

a list of batches

3.20.3.2 getLoss()

```
float NeuralNetwork::getLoss ( )
```

Returns

the current Loss

3.20.3.3 predictBatch()

```
Matrix NeuralNetwork::predictBatch (
            Matrix & miniBatch,
            Matrix & labels )
```

this executes a simple prediction and returns the predicted labels

Parameters

miniBatch	
labels	this should be removed in Future, but is still needed as the Loss operation needs an label input

Returns

the Predictions: one row per sample, one collum for each label. if e.g. sample zero has label one label.↩ row(0)={0,1,0,0,....,0}

3.20.3.4 readParameters()

```
bool NeuralNetwork::readParameters (
            std::string dir,
            std::string networkName )
```

Initializes the Network with prestored Parameters stored in dir

Parameters

dir	
networkName	

Returns

if succeeded

3.20.3.5 testAccuracy()

```
void NeuralNetwork::testAccuracy (
            Matrix & results,
            Matrix & labels,
            float & correct,
            float & total )  [static]
```

Adds to the variable correct the number of correct predictions Adds to the variable total the number of samples that where predicted accuracy can then be calculated with correct/total

Parameters

results	
labels	
correct	
total	

3.20.3.6 train()

```
float NeuralNetwork::train (
            DataSet & data,
            HyperParameters params,
            float trainingLossThreshold = 1 )
```

Trains the Network on the trainingData stored in data

Parameters

data	
params	
trainingLossThreshold	

Returns

the final Loss

3.20.3.7 trainAndValidate()

```
TrainingEvaluation NeuralNetwork::trainAndValidate (
            DataSet & data,
            HyperParameters params,
            float trainingLossThreshold = 1 )
```

Trains the Network on the trainingData stored in data After each epoch the loss and accuracy is evaluated for both trainingSet and Validation Set the results are stored in an TrainingEvaluation object

Parameters

data	
params	
trainingLossThreshold	this can be used to tell at which cost the training should stop

Returns

Training results

3.20.3.8 trainBatch()

```
void NeuralNetwork::trainBatch (
            Matrix & miniBatch,
```

```
            Matrix & labels,
            HyperParameters & params )
```

Trains the network on a specific batch and the corresponding Labels

Parameters

miniBatch	one row per sample
labels	one row per sample, one collum for each label. if e.g. sample zero has label one label.row(0)={0,1,0,0,....,0}
params	the HyperParameter object contains everything need to update the Weights, Biases and Filters

3.20.3.9 writeParameters()

```
bool NeuralNetwork::writeParameters (
            std::string dir,
            std::string networkName )
```

Writes the parameter of the current Network

Parameters

dir	
networkName	

Returns

if succeeded

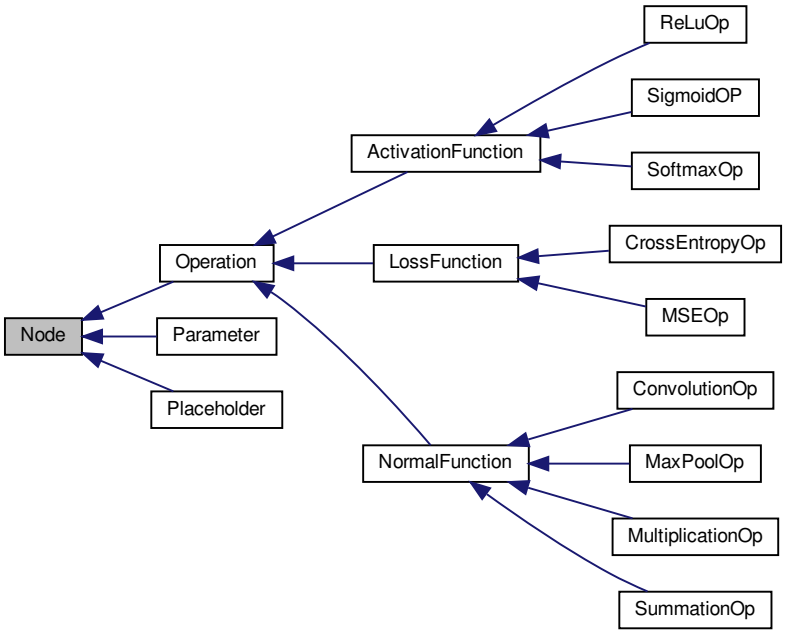
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/include/NeuralNetwork.hpp
- /home/pbo/CLionProjects/libdl/library/NeuralNetwork/src/NeuralNetwork.cpp

3.21 Node Class Reference

```
#include <Node.hpp>
```

Inheritance diagram for Node:



Public Member Functions

- const Matrix & **getForward** () const
- void **setForward** (const Matrix &forward)
- const Matrix & **getPreviousGradients** () const
- void **setPreviousGradients** (const Matrix &previousGradients)
- void **setOutputChannels** (int outputChannels)
- int **getOutputChannels** () const

3.21.1 Detailed Description

This class represents the Base Class of all Nodes of the Computational Graph.

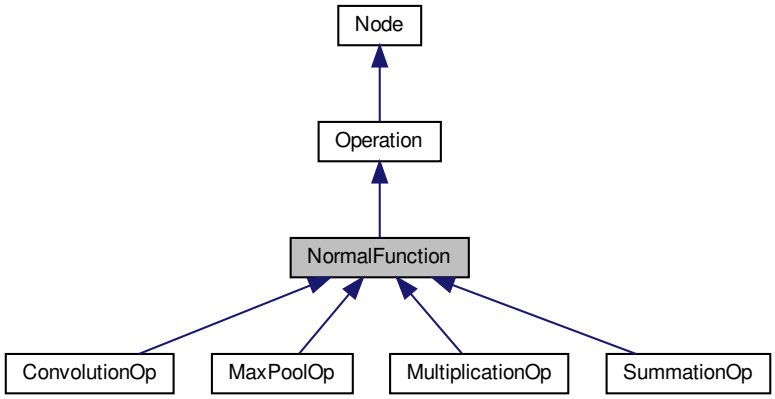
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/include/Node.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/src/Node.cpp

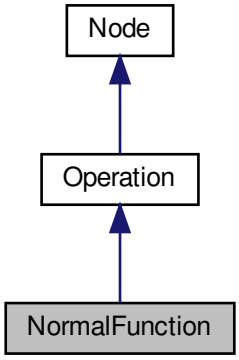
3.22 NormalFunction Class Reference

`#include <NormalFunction.hpp>`

Inheritance diagram for NormalFunction:



Collaboration diagram for NormalFunction:



Public Member Functions

- NormalFunction (std::shared_ptr< Node > X, std::shared_ptr< Parameter > parameter, int outputChannels)
- NormalFunction (std::shared_ptr< Node > X, int outputChannels)
- const std::shared_ptr< Parameter > & getParameter () const

  arameters such as weights, biases, filters

3.22.1 Detailed Description

A NormalFunction operation represents typical operations that take a Weight or Bias Right now this also counts for the MaxPool Layer, however in future it should get its own Base class as MaxPool does not hold Weights or Biases

3.22.2 Constructor & Destructor Documentation

3.22.2.1 NormalFunction() [1/2]

```
NormalFunction::NormalFunction (
            std::shared_ptr< Node > X,
            std::shared_ptr< Parameter > parameter,
            int outputChannels )
```

- creates the operation Object using the the obligatory Input Node and the outputChannels.
- stores the parameters for this operation

Parameters

X	input Node
parameter	parameters such as weights, biases, filters
outputChannels	the expected outputChannels for this operation

3.22.2.2 NormalFunction() [2/2]

```
NormalFunction::NormalFunction (
            std::shared_ptr< Node > X,
            int outputChannels )
```

- creates the operation Object using the the obligatory Input Node and the outputChannels.

Parameters

X	input Node
outputChannels	the expected outputChannels for this operation

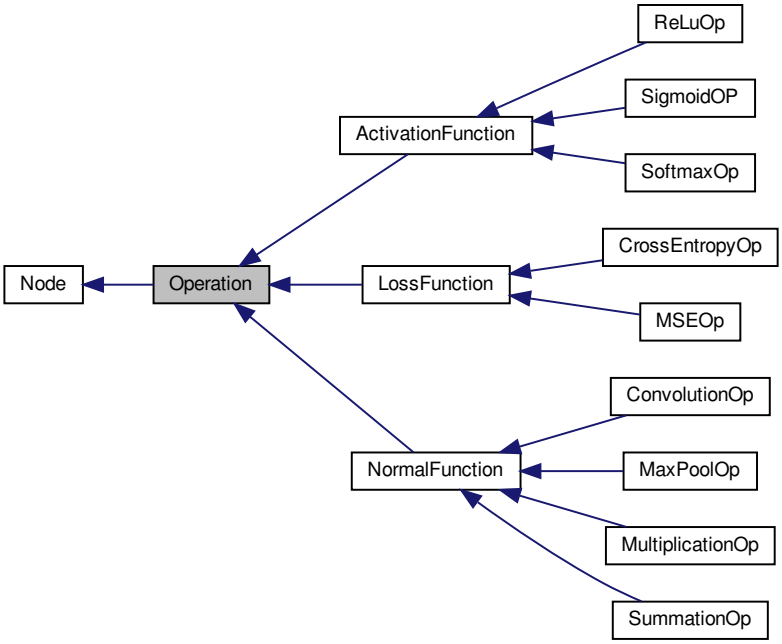
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/NormalFunction.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/NormalFunction.cpp

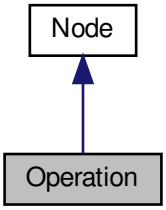
3.23 Operation Class Reference

`#include <Operation.hpp>`

Inheritance diagram for Operation:



Collaboration diagram for Operation:



Public Member Functions

- Operation (std::shared_ptr< Node > input, int outputChannel)
- ~Operation () override=default
- virtual void forwardPass ()

- virtual void backwardPass ()
- int forwardPassWithMeasurement ()
- int backwardPassWithMeasurement ()
- int getForwardTime () const
- int getBackwardsTime () const
- const std::shared_ptr< Node > & **getInput** () const
- int **getInputChannels** () const

3.23.1 Detailed Description

This class defines the interface each Operation of the Computational Graph has to implement

3.23.2 Constructor & Destructor Documentation

3.23.2.1 Operation()

```
Operation::Operation (
            std::shared_ptr< Node > input,
            int outputChannel )
```

stores the obligatory Input Node for a operation and sets the _outputChannel for this Operation. Further it stores the _outputChannel of the input Node as _inputChannel

Parameters

input	the inputNode which might be an Placeholder or an Operation
outputChannel	the #outputChannel of this Operation

3.23.2.2 ~Operation()

```
Operation::~Operation ( )  [override], [default]
```

Default destructor

3.23.3 Member Function Documentation

3.23.3.1 backwardPass()

```
virtual void Operation::backwardPass ( )  [inline], [virtual]
```

calculates the gradients for the operation inputs w.r.t. the last Node in the Computational Grpah. In order to do so it uses the member variable _previousGradients of Class Node, which contains the Gradients of the previous operation.

Each implementation must set the member variable _previousGradients of the operations Inputs by calling [All↩ InputNodes]->setPreviousGradients(Matrix calculatedGradients)

Reimplemented in MaxPoolOp, SigmoidOP, ConvolutionOp, SoftmaxOp, CrossEntropyOp, MSEOp, Multiplication↩ Op, ReLuOp, and SummationOp.

3.23.3.2 backwardPassWithMeasurement()

```
int Operation::backwardPassWithMeasurement ( )
```

executes the backwardPass and stores the computation time

Returns

the computation Time

3.23.3.3 forwardPass()

```
virtual void Operation::forwardPass ( )  [inline], [virtual]
```

calculates the outputValue of this Operation using the Input Nodes.

Each implementation must set the member variable _forward by calling setForward(Matrix calculatedOutputValue)

Reimplemented in MaxPoolOp, ConvolutionOp, SoftmaxOp, CrossEntropyOp, MSEOp, MultiplicationOp, ReLuOp, SigmoidOP, and SummationOp.

3.23.3.4 forwardPassWithMeasurement()

```
int Operation::forwardPassWithMeasurement ( )
```

executes the forwardPass and stores the computation time

Returns

the computation Time

3.23.3.5 getBackwardsTime()

```
int Operation::getBackwardsTime ( ) const
```

this does only work if backwardPassWithMeasurement() is called

Returns

computation Time of backwardPass

3.23.3.6 getForwardTime()

```
int Operation::getForwardTime ( ) const
```

this does only work if forwardPassWithMeasurement() is called

Returns

computation Time of forwardPass

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

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/include/Operation.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/src/Operation.cpp

3.24 OperationsFactory Class Reference

```
#include <OperationsFactory.hpp>
```

Static Public Member Functions

- static std::shared_ptr< ConvolutionOp > **createConvolutionOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input, Matrix filterMatrix, int channels, int stride)
- static std::shared_ptr< SummationOp > **createSummationOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input, Matrix biasMatrix, int channel)
- static std::shared_ptr< MultiplicationOp > **createMultiplicationOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input, Matrix weightMatrix)
- static std::shared_ptr< MaxPoolOp > **createMaxpoolOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input, int kernelDim, int stride)
- static std::shared_ptr< MSEOp > **createMSEOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input, Matrix labelMatrix)
- static std::shared_ptr< CrossEntropyOp > **createCrossEntropyOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input, Matrix labelMatrix)
- static std::shared_ptr< ReLuOp > **createReLuOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input)
- static std::shared_ptr< SigmoidOP > **createSigmoidOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input)
- static std::shared_ptr< SoftmaxOp > **createSoftmaxOp** (std::shared_ptr< Graph > graph, const std::shared_ptr< Node > input, int amountClasses)

3.24.1 Detailed Description

This class implements the Factory Pattern in a functional way. Each creation function assures that an Operation is not only created but also added to the compute graph if the input Node is not already in the Graph this will fail with an runtime_error cause by the Graph class

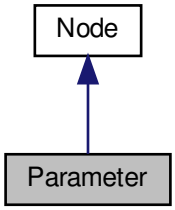
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/OperationsFactory.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/OperationsFactory.cpp

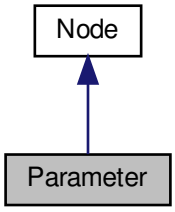
3.25 Parameter Class Reference

`#include <Parameter.hpp>`

Inheritance diagram for Parameter:



Collaboration diagram for Parameter:



Public Member Functions

- Parameter (Matrix &filter, int channel)
- Parameter (Matrix &input)
- ~Parameter () override=default
- void updateParameter (const HyperParameters &hyperParameters)

3.25.1 Detailed Description

This Node holds the Parameters that define the function of the Computational Graph. In terms of a Neural Networks, this class holds the functionality for Filters, Weights and Biases, which may be updated during Backpropagation

3.25.2 Constructor & Destructor Documentation

3.25.2.1 Parameter() [1/2]

```
Parameter::Parameter (
            Matrix & filter,
            int channel )
```

Constructs a Filter and sets the Output to the filterMatrix. Further it initializes the momentum and RMSprob for possible future Updates using the ADAM optimizer

Parameters

filter	each Row corresponds to one Kernel, the size of one Kernel equals std::pow(KernelDim,2)∗channel
channel	the Amount of Channels each Kernel has

3.25.2.2 Parameter() [2/2]

```
Parameter::Parameter (
            Matrix & input )  [explicit]
```

Constructs a Weight or Bias and sets the Output to the inputMatrix. Further it initializes the momentum and RM↩ Sprob for possible future Updates using the ADAM optimizer

Parameters

input	

3.25.2.3 ∼Parameter()

```
Parameter::~Parameter ( )  [override], [default]
```

Default Destructor

3.25.3 Member Function Documentation

3.25.3.1 updateParameter()

```
void Parameter::updateParameter (
            const HyperParameters & hyperParameters )
```

updates the output of the Variable according to the Optimizer and Hyperparameters that are passed.

Parameters

hyperParameters	Object containing all hyperParameters for Neural Networks

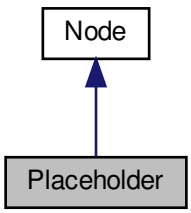
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/include/Parameter.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/src/Parameter.cpp

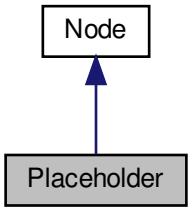
3.26 Placeholder Class Reference

```
#include <Placeholder.hpp>
```

Inheritance diagram for Placeholder:



Collaboration diagram for Placeholder:



Public Member Functions

- Placeholder (Matrix &batch, int channel)
- Placeholder (Matrix &labels)
- ~Placeholder () override=default

3.26.1 Detailed Description

This Node represents a Placeholder for a fixed Input into the Computational Graph In terms of a Neural Networks, this class holds the functionality for inputing Samples or Labels to corresponding Samples

3.26.2 Constructor & Destructor Documentation

3.26.2.1 Placeholder() [1/2]

```
Placeholder::Placeholder (
            Matrix & batch,
            int channel )
```

constructs a mini-batch a sets the output to batch. Further the number of output channels is set to the number of channels of each samples

Parameters

batch	a whole mini-batch, each row contains one sample
channel	number of channels of each sample

3.26.2.2 Placeholder() [2/2]

```
Placeholder::Placeholder (
            Matrix & labels )  [explicit]
```

constructs a Placeholder that contains the labels to a previously created mini-batch.

Parameters

labels	each row corresponds to the label of one sample, while the Dimension of each row is the #labels, this way the corresponding label is identified by the collumn that holds a "1"

3.26.2.3 ~Placeholder()

```
Placeholder::~Placeholder ( )  [override], [default]
```

Default Destructor

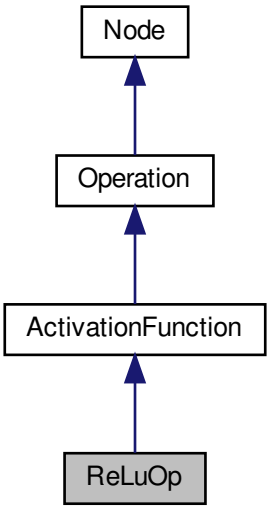
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/include/Placeholder.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/src/Placeholder.cpp

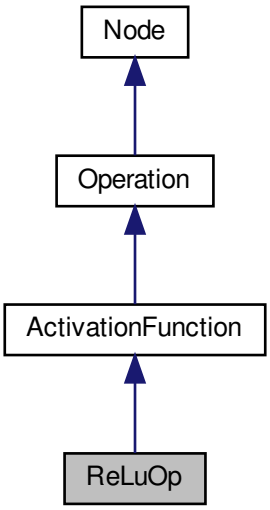
3.27 ReLuOp Class Reference

```
#include <ReLuOp.hpp>
```

Inheritance diagram for ReLuOp:



Collaboration diagram for ReLuOp:



Public Member Functions

- ReLuOp (std::shared_ptr< Node > X)
- void forwardPass () override

  default destructor
- void backwardPass () override

Static Public Member Functions

- static float deriveReLu (float element)

3.27.1 Detailed Description

Implements the ReLu activation function

3.27.2 Constructor & Destructor Documentation

3.27.2.1 ReLuOp()

```
ReLuOp::ReLuOp (
            std::shared_ptr< Node > X )  [explicit]
```

creates an ActivationFunction object using the input node X

Parameters

X	

3.27.3 Member Function Documentation

3.27.3.1 backwardPass()

```
void ReLuOp::backwardPass ( )  [override], [virtual]
```

calculates the sigmoid gradients w.r.t to input X using function deriveReLu

Reimplemented from Operation.

3.27.3.2 deriveReLu()

```
float ReLuOp::deriveReLu (
            float element )  [static]
```

calculates the derivative of the ReLu function for a given element

Parameters

element	

Returns

dReLu(element)

3.27.3.3 forwardPass()

```
void ReLuOp::forwardPass ( )  [override], [virtual]
```

default destructor

calculates ReLu for the whole input X and set the output value as forward pass value

Reimplemented from Operation.

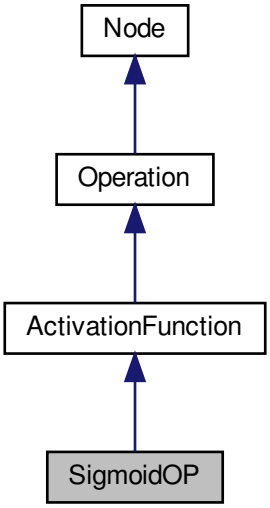
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/ReLuOp.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/ReLuOp.cpp

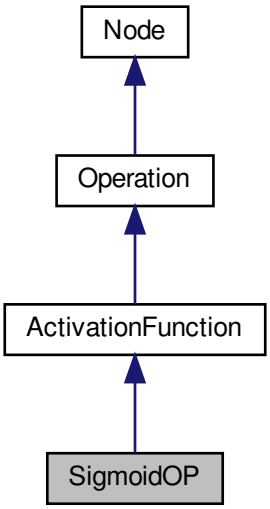
3.28 SigmoidOP Class Reference

`#include <SigmoidOP.hpp>`

Inheritance diagram for SigmoidOP:



Collaboration diagram for SigmoidOP:



Public Member Functions

- SigmoidOP (std::shared_ptr< Node > X)
- void forwardPass () override

  default destructor
- void backwardPass () override

Static Public Member Functions

- static float sigmoid (float a)

3.28.1 Detailed Description

Implements the Sigmoid activation function

3.28.2 Constructor & Destructor Documentation

3.28.2.1 SigmoidOP()

```
SigmoidOP::SigmoidOP (
            std::shared_ptr< Node > X )  [explicit]
```

creates an ActivationFunction object using the input node X

Parameters

X	

3.28.3 Member Function Documentation

3.28.3.1 backwardPass()

```
void SigmoidOP::backwardPass ( )  [override], [virtual]
```

calculates the sigmoid gradients w.r.t to input X

Reimplemented from Operation.

3.28.3.2 forwardPass()

```
void SigmoidOP::forwardPass ( )  [override], [virtual]
```

default destructor

calculates sigmoid for the whole input X and set the output value as forward pass value

Reimplemented from Operation.

3.28.3.3 sigmoid()

```
float SigmoidOP::sigmoid (
            float a )  [static]
```

calculates the sigmoid function

Parameters

a	

Returns

sigmoid(a)

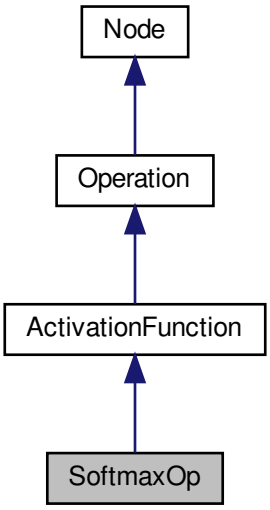
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/SigmoidOP.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/SigmoidOP.cpp

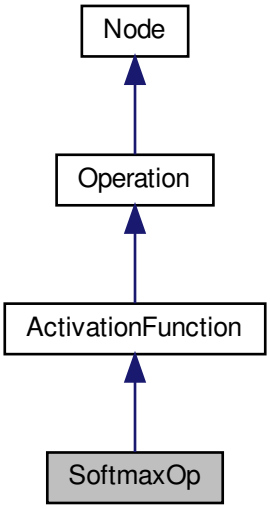
3.29 SoftmaxOp Class Reference

```
#include <SoftmaxOp.hpp>
```

Inheritance diagram for SoftmaxOp:



Collaboration diagram for SoftmaxOp:



Public Member Functions

- SoftmaxOp (std::shared_ptr< Node > X, int numberClasses)

- void forwardPass () override

  default destructor
- void backwardPass () override

3.29.1 Detailed Description

Implements the Softmax Classifier Right Now Softmax does only work in combination with CrossEntropyOp, as the gradient of both together is calculated there and then just passed backwards

3.29.2 Constructor & Destructor Documentation

3.29.2.1 SoftmaxOp()

```
SoftmaxOp::SoftmaxOp (
            std::shared_ptr< Node > X,
            int numberClasses )
```

creates an ActivationFunction object stores the number of classes that need to be predicted

Parameters

X	input to be transformed
numberClasses	#predictionClasses

3.29.3 Member Function Documentation

3.29.3.1 backwardPass()

```
void SoftmaxOp::backwardPass ( )  [override], [virtual]
```

passes the softmax-crossEntropy-classifier gradient calculated in CrossEntropyLoss further backwards to the input Node X see also: https://deepnotes.io/softmax-crossentropy

Reimplemented from Operation.

3.29.3.2 forwardPass()

```
void SoftmaxOp::forwardPass ( )  [override], [virtual]
```

default destructor

performs softmax classification

Reimplemented from Operation.

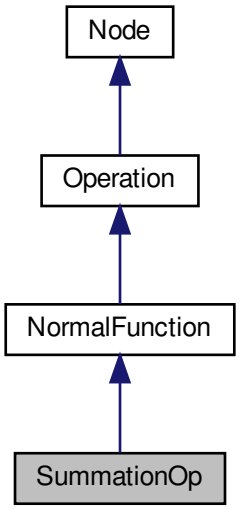
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/SoftmaxOp.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/SoftmaxOp.cpp

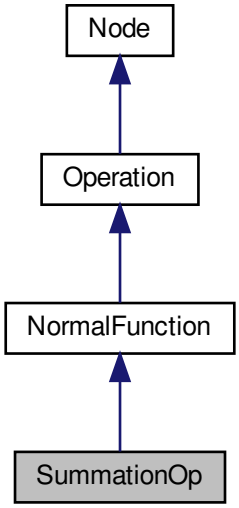
3.30 SummationOp Class Reference

```
#include <SummationOp.hpp>
```

Inheritance diagram for SummationOp:



Collaboration diagram for SummationOp:



Public Member Functions

- SummationOp (std::shared_ptr< Node > X, std::shared_ptr< Parameter > biases)
- void forwardPass () override

  default destructor
- void backwardPass () override

3.30.1 Detailed Description

Implements the Matrix summation that is mainly used for adding a Bias to a previous calculated forward pass of a Neuron/MultiplicationOP

3.30.2 Constructor & Destructor Documentation

3.30.2.1 SummationOp()

```
SummationOp::SummationOp (
            std::shared_ptr< Node > X,
            std::shared_ptr< Parameter > biases )
```

creates NormalFunction object and passes the input outputChannesl through stores a node Containing the biases for the summation

Parameters

X	input Batch
biases	the shape of biases should completly equal X

3.30.3 Member Function Documentation

3.30.3.1 backwardPass()

```
void SummationOp::backwardPass ( )  [override], [virtual]
```

calculates the gradients w.r.t to X and biases

Reimplemented from Operation.

3.30.3.2 forwardPass()

```
void SummationOp::forwardPass ( )  [override], [virtual]
```

default destructor

executes the matrix summation and stores the output as forward pass value

Reimplemented from Operation.

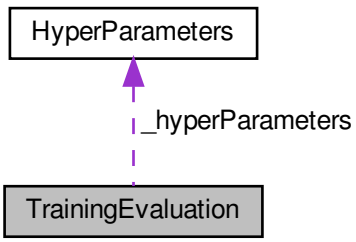
The documentation for this class was generated from the following files:

- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/include/SummationOp.hpp
- /home/pbo/CLionProjects/libdl/library/ComputationalGraph/Operations/src/SummationOp.cpp

3.31 TrainingEvaluation Struct Reference

```
#include <commonDatatypes.hpp>
```

Collaboration diagram for TrainingEvaluation:



Public Member Functions

- **TrainingEvaluation** (HyperParameters hyperParameters)
- **TrainingEvaluation** (std::vector< float > trainingLoss, std::vector< float > trainingAccuracy, HyperParameters hyperParameters)
- **TrainingEvaluation** (std::vector< float > trainingLoss, std::vector< float > trainingAccuracy, std::vector< float > validationLoss, std::vector< float > validationAccuracy, HyperParameters hyperParameters)

Public Attributes

- std::vector< float > **_trainingLoss**
- std::vector< float > **_validationLoss**
- std::vector< float > **_trainingAccuracy**
- std::vector< float > **_validationAccuracy**
- HyperParameters **_hyperParameters**

3.31.1 Detailed Description

This struct holds the results of a training, storing in each field of a vecotr, the value for one epoch

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

- /home/pbo/CLionProjects/libdl/library/Utils/include/commonDatatypes.hpp

Index