mLEARn

1.0.0

Generated by Doxygen 1.8.16

Chapter 1

The Basics of mLEARn

This chapter is split into the following sections:

- Introduction
- Artificial Neural Networks
- Single-layer Networks
- Multi-layer Networks
- Training/Optimization Methods
- Model Improvement

1.1 Introduction

Neural networks theory, training and optimization methods are explained in detail in many works. This chapter gives a brief discussion/introduction to neural networks.

1.2 Artificial Neural Networks

The biological neural system [4] is made up of billions of neurons interconnected through axons, synaptic junctions and dendrites as shown in Figure 1.1. The input signal to each neuron is summed up at the cell body or the neural soma to give potential. If the potential is greater than a threshold, the neuron fires and the pulse goes down the axon, otherwise the neuron is in a quiescent state. Furthermore, the end of the axon of each neuron is connected to the dendrite of another neuron through a synaptic gap, and depending on the strength of this gap, the pulse may or may not get through. In principle, the pulse received at the dendritic ends of a neuron is proportional to the pulses of the neighboring interconnected neurons.

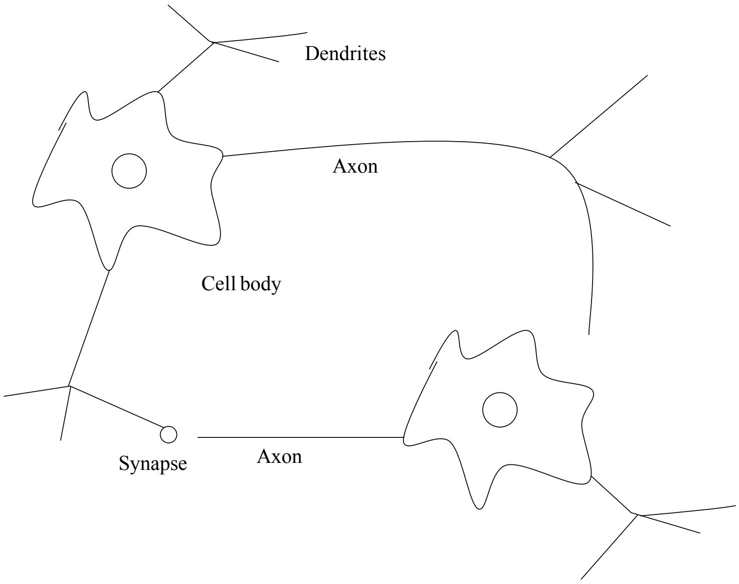


Figure 1.1 A schematic diagram of a biological neuron

Artificial neural networks (ANNs) [7] are inspired by the biological neural system. In ANNs, the strength of the synaptic gap is equivalent to the weight from one connection to the other [2]. Similarly to the biological neural system, ANNs consist of interconnected units which communicate through many weighted connections. One of the ways by which ANNs can be classified is the way the units are interconnected. The two main categories are usually `recurrent` (feedback) and `feed-forward` architectures. The main difference between `feedback` and `feed-forward` networks is that in `feedback` networks, outputs from one or more units are fed back to units in the same layer or previous layers, e.g. `Hopfield-little` model [6]. ANNs can also be classified by the number of layers in the network, namely `single-layer` and `multi-layer` networks [1]. The work in mLEA↩ Rn uses networks with the `feed-forward` architecture. Firstly, the following section presents `single-layer` networks.

1.3 Single-layer Networks

In a `single-layer` network, there is only one layer of processing units, the output layer. Figure 1.2 shows a single-layer network with a single neuron and two inputs, where w_{00}, w_{01}, w_{02} are the weights, x_1, x_2 are the inputs, b is a threshold/bias and $\hat{y}$ is the output of the neuron. The neuron fires (is activated) if the activation, a, is greater than the threshold b. The activation is the sum of weighted inputs to a neuron.

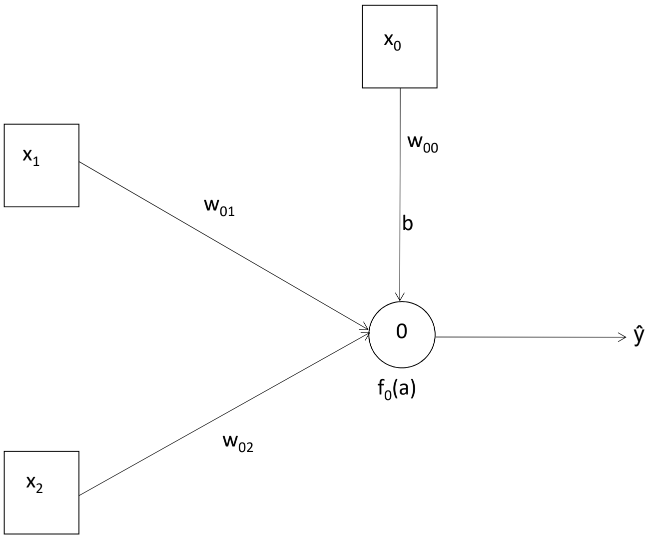


Figure 1.2 A single-layer perceptron

The output is a function of the inputs as well as the weights and is computed as follows:

$$\hat{y} = f(w,x) = f\,(a)\,,$$
$$a = \sum_{i=1}^{2} w_{0i}x_i + b.$$

The threshold, T, is usually considered as a weight that can be adapted whose input (x_0) is set equal 1. The equation can be re-written with T absorbed as an extra weight as follows:

$$a = \sum_{i=0}^{2} w_{0i}x_i.$$

For example, if the activation function f is the `Heaviside` step function, then the single neuron in Figure 1.2 outputs 1 if the activation a exceeds the threshold T and 0 otherwise [5]

A single neuron can only solve problems that are `linearly separable` [8]. To solve more complex problems, a cascade of neurons is needed. The simplest neural network is the perceptron network which consists of a cascade of processing units in the same layer. A learning algorithm is required for adapting the weights and thresholds in a perceptron network. There are two most common algorithms which are the `perceptron learning` algorithm and the `delta rule` [10], [13]. `Single-layer` networks are limited in their ability to solve complex problems. In many practical problems, networks with more layers are required and the most commonly used is the `multi-layer perceptron` (MLP).

1.4 Multi-layer Networks

An MLP contains at least two layers of processing units, namely the hidden and output layers as shown in Figure 1.3. These networks are more powerful than the `single-layer` networks and can overcome the limitations of `single-layer` perceptron. MLPs with two layers of processing units are powerful and with three layers can approximate any continuous function [1].

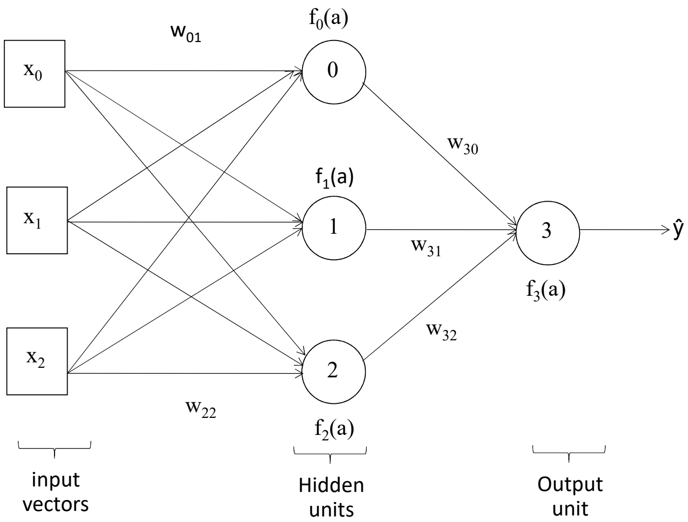


Figure 1.3 A multi-layer perceptron

The most commonly used algorithm for training MLPs is called `backpropagation` [11]. This is a `gradient descent` method of minimization of error functions. The `backpropagation` algorithm involves propagation of errors backwards from the output units through the network. The primary aim of this algorithm is to find a set of weight matrices that minimizes an error or a cost function over the total number of examples in the training set. A full discussion of this algorithm and other variants can be found in [1], [9].

1.5 Training/Optimization Methods

The purpose of training is to find weights that minimizes the error between the predicted value $\hat{y}(w, b, x)$ and the reference y for all examples in the training set. A function used to evaluate how good a set weight is the cost, error or objective function. The cost function $C(w, b)$ is a function of the weights and biases in the network; and gradient descent algorithm is used to minimize the function. The cost functions implemented in mLEARn are mean squared error (MSE), mean absolute error (MAE) and cross entropy.

$$
\begin{aligned}
C &= \frac{1}{2n}\sum_{j}^{n}\sum_{k}^{K}\left(y^k - \hat{y}^k\right)^2 \\
C &= \frac{1}{n}\sum_{j}^{n}\sum_{k}^{K}\left|y^k - \hat{y}^k\right| \\
C &= -\frac{1}{n}\sum_{j}^{n}\sum_{k}^{K} y^k \log_2 \hat{y}^k
\end{aligned}
$$

The equations for the three functions are shown above. N is the mini-batch size, K is the number of units in the output layer or the number of classes, y and $\hat{y}$ are the reference and predicted values respectively.

The manner of computation of δ (output error/delta) for the nodes in the output layer is different from those of the hidden nodes. For each node in the output layer, δ is the product of input error/delta and derivative of the activation function.

$$\delta_i = (y - \hat{y})\frac{d}{da_i}f(a_i)$$

But δ for the hidden nodes are computed by propagation of errors (δ) from higher layers. The δ for each node is a function of δ of all nodes connected to a higher layer

$$\delta_i = \sum_j \delta_j w_{ij}\frac{d}{da_i}f(a_i)$$

This is true irrespective of the activation function used. Table shows various activation functions and their derivatives.

Function	f(a)	f'(a)
linear	a	1
sigmoid	1/(1 + exp(-a))	f(a) $*$ (1 -f(a))
tanh	tanh(a)	1 - tanh(a)^2
ReLU	max(0, a)	1 if a $>$ 0 else 0
ELU	a if a $>$ 0 else alpha(exp(a) - 1)	1 if x $>$ 0 else alpha(exp(a))
softmax	exp(a)/sum(exp(a))	(a_i)(1 - a_m) if i == m else -(a_m)(a_i)

1.6 Model Improvement

This section is about techniques for improving neural networks model. It discusses regularization methods and variations of SGD, namely, Adagrad and RMSProp. The effectiveness of a neural network model is its generalization power, that is, its efficiency in predicting correctly unseen instances in the test set. There are two concepts which determine this, namely, overfitting and underfitting.

Overfitting arises when a network learns the underlying properties and relationships in a data set as well as any inherent noise. So, it tries as much as possible to reduce the error on the training set but performs badly on test data set. This is primarily caused by high degree of freedom given to network model, by either excess number of hidden layers or number of unitss in the hidden layer. Such models have low bias and high variance [1]. On the other hand, a high-biased network does not have enough units in hidden layer or enough hidden layers. Generally, it even fails to find the local minima of a cost function, so underfits. Underfitting models have high bias and low variance. A fundamental problem of machine learning is to achieve a good bias/variance trade-off. One can determine an overfitting/underfitting model by observing the performance on train and validation set. If the accuracy continues to increase on the train set but decreases on the validation set, the model is overfitting. An underfitting model does not perform well both on train and validation data.

Regularization is a method used to ‘regulate’ statistical models during training, to prevent overfitting. This is achieved by penalizing complex models. Model complexity depends on the size and number of model parameters. For example, in the context of neural networks, the parameters are the weights and the size depends on network architecture. A model can be regularized for simplicity or sparsity. Simplicity involves adding a fraction of the sum of the squares of all the weights to the cost function, known as L2. Sparsity involves adding a fraction of the sum of the absolute values of all the weights to the cost function, known as L1. There is also a third method, dropout, which is not yet implemented in mLEARn.

The weight updates with L2 and L1 regularization are:

$$w_{ij} = w_{ij} - \eta\delta_j y_i - \frac{\eta\lambda}{N} w_{ij}$$

$$w_{ij} = w_{ij} - \eta\delta_j y_i - \frac{\eta\lambda}{N} sgn\left(w_{ij}\right)$$

Where λ is a regularization parameter, sgn is the sign of function, that is, sgn(w) is $+1$ for w greater than or equal to zero, and -1 otherwise.

There are other methods used to improve or speed up convergence of stochastic gradient descent methods. Some of those methods implemented in mLEARn are momentum and variable learning rate techniques such as Adaptive gradient (Adagrad) [3] and RMSProp [12]. Stochastic gradient descent with momentum adds a fraction of a previous weight update to current update:

$$w_{ij}^t = w_{ij}^t - \Delta w_{ij}^t + \eta\beta\Delta w_{ij}^{t-1}$$

Where β is a momentum parameter (a value between 0 and 1.

SGD utilizes a common/global learning rate for all parameters. But tuning the learning rate for each parameter improves models and leads to quicker convergence. Adagrad is a variant of SGD in which each weight change has a different learning rate. The learning rate η_{ij}^t for w_{ij} is obtained by dividing the global rate by the square root of cumulative sum of square of previous weight change. Epsilon ϵ is a term added to avoid division by zero.

$$\eta_{ij}^t = \frac{\eta}{\sqrt{\sum_t \left(\Delta {w_{ij}^t}^2\right) + \epsilon}}$$

RMSProp is like Adagrad in that it employs per parameter learning rate as follows,

$$R_{ij}^t = \mu R_{ij}^{t-1} + (1-\mu)\,\Delta {w_{ij}^t}^2$$

$$\eta_{ij}^t = \frac{\eta}{\sqrt{\sum_t \left(R_{ij}^t\right) + \epsilon}}$$

where \$\eta_{ij}^t\$ is the learning rate for \$w_{ij}\$ at time \$t\$, \$ \Delta w_{ij}\$ is previous weight change, \$\epsilon\$ is used to prevent division by zero and \$\mu\$ is between \$0\$ and \$1\$. The initial value of \$R_{ij}\$ used is \$0.1\$.

Chapter 2

Architecture of mLEARn

This chapter is split into the following sections:

- Introduction
- Classes and Libraries
- Serialization - Saving and Loading Models

2.1 Introduction

There are many works on neural networks, their applications, hyper-parameter tuning and optimizers. mLEA↩ Rn is designed to be modular, making it very easy for students and researchers to test and extend architectures and optimizers. mLEARn currently implements only `feed-forward multi-layer` networks. The following sections describe implemented classes and libraries.

2.2 Classes and Libraries

The following table shows the classes implemented:

Namespace	Class
mlearn	Node< T >
mlearn	NetNode< T >
mlearn	Activation< T >
mlearn	CostFunction< T >
mlearn	MSE< T >
mlearn	CrossEntropy< T >
mlearn	MAE< T >
mlearn	Layer< T >
mlearn	Network< T >
mlearn	DataReader< T >
mlearn	MNISTReader< T >
mlearn	IrisReader< T >
mlearn	GenericReader< T >
mlearn	Optimizer< T >
mlearn	SGD< T >
mlearn	Adagrad< T >
mlearn	RMSProp< T >

The Node class is the fundamental data structure used. Different machine learning methods can extend the Node class, e.g. NetNode is an extension of the Node class for multi-layer perceptron. The Inheritance diagram for mlearn::Node$<$ T $>$ is shown in Figure 2.1.

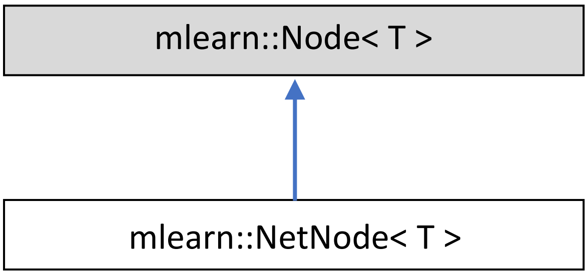


Figure 2.1 Inheritance diagram for mlearn::Node$<$ T $>$

```
// creates ublas vector of int and double from std vector
std::vector<int> sv1 {1, 2, 3, 4};
std::vector<double> sv2{2, 3, 4.0, 2};
mublas::vector<int> v1(sv1.size());
mublas::vector<double> v2(sv2.size());
std::copy(sv1.begin(), sv1.end(), v1.begin());
std::copy(sv2.begin(), sv2.end(), v2.begin());
// creates Node objects n1 and n2
Node<int> n1(v1);
Node<double> n2(v2);
// creates NetNode objects n1 and n2
NetNode<int> n3(v1);
NetNode<double> n4(v2);
```

The Activation class handles activations in the network. Currently, the functions implemented are sigmoid, tanh, ReLU, leaky ReLU, identity, softmax and ELU.

```
// create an Activation object of type sigmoid
Activation<double> act("sigmoid");
// compute on a Node object n1
act.compute(n1);
//computeDerivative
act.computeDerivative(n1);
```

The CostFunction class is responsible for objective functions. Cost functions implemented are mean squared error (MSE), mean absolute error (MAE) and cross entropy. The Inheritance diagram for mlearn::Node$<$ T $>$ is shown in Fig.

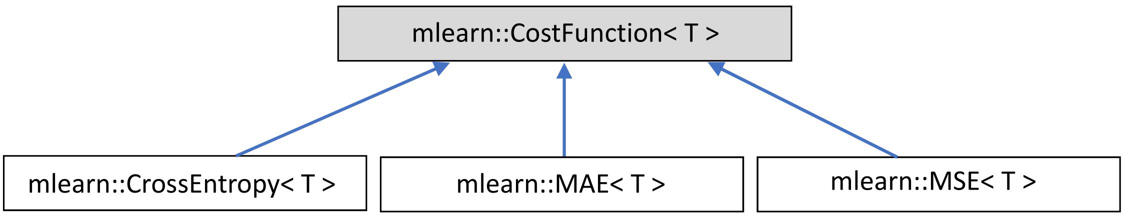


Figure 2.2 Inheritance diagram for mlearn::CostFunction$<$ T $>$

```
// create a pointer to CostFunction object of MSE
CostFunction<double>* pcost = new MSE<double>;
```

The Layer class represents a layer in a neural network. The pointers "previous" and "next" point to the previous and next layers respectively. Each layer has a forward function(forwardProp) that produces output data from input data and a backward function(backwardProp) that produces an output delta (gradient) from an input delta. A layer can be a hidden or an output layer.

```
//  Creates 2 Activation objects: hidden and output.
Activation<double> hidden("sigmoid"), output("softmax");
//  A hidden layer with input and output dimensions 2
//  Activation function used is sigmoid
Layer<double> hidden_layer(2, 2, "hidden", hidden);
//  An output layer with input and output dimensions 2 and 1 respectively.
//  Activation function used is softmax
Layer<double> output_layer(2, 1, "output", output);
```

The Network class is a classic MLP consisting of sequences of layers: one or more hidden layers and an output layer. The network can be trained using mini-batch SGD, Adagrad or RMSProp.

```
//  Creates 2 Activation objects: hidden and output.
Activation<double> hidden("sigmoid"), output("softmax");
//  A hidden layer with input and output dimensions 2.
//  Activation function used is sigmoid
Layer<double> hidden_layer(2, 2, "hidden", hidden);
//  An output layer with input and output dimensions 2 and 1 respectively.
//  Activation function used is softmax
Layer<double> output_layer(2, 1, "output", output);
//  Creates a Network object, error function is cross entropy
Network<double> model(new CrossEntropy<double>);
//  Adds the hidden and output layers to the network
model.addLayer(hidden_layer);
model.addLayer(output_layer);
//  Connects the layers together
model.connectLayers();
```

The DataReader class is the base class responsible for reading train/test dataset into features/labels. 3 different readers have been implemented, namely, MNISTReader, GenericReader and IrisReader. They extend the base class DataReader. The Inheritance diagram for mlearn::DataReader$<$ T $>$ is shown in Figure 2.3.

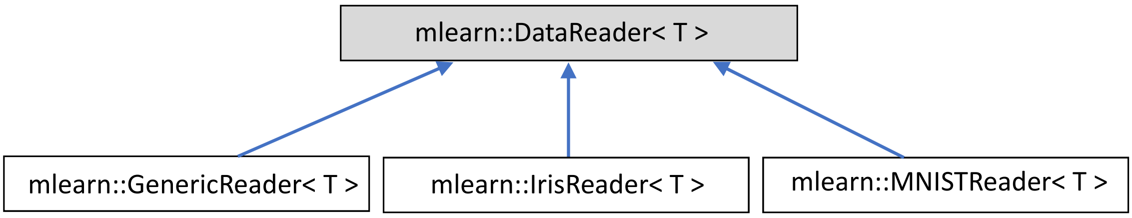


Figure 2.3 Inheritance diagram for mlearn::DataReader$<$ T $>$

```
// creates a GenericReader object "train" and read xor file in the data directory
GenericReader<double> train("data/xor_header.dat", 2, 2, ' ', true);
// creates an MNISTReader object "mnist" and read a sample file in the data directory
MNISTReader<double> mnist("data/mnist_sample.csv", ',',  false);
// call the read method
train.read();
mnist.read();
```

The Optimizer class is the base class responsible for training algorithms. 3 optimizers are currently implemented, namely, SGD, Adagrad and RMSProp. The SGD optimizer is mini-batch stochastic gradient descent. The difference between Adagrad/RMSProp and SGD is that the latter uses per parameter learning rate. The Inheritance diagram for mlearn::Optimizer$<$ T $>$ is shown in Figure 2.4.

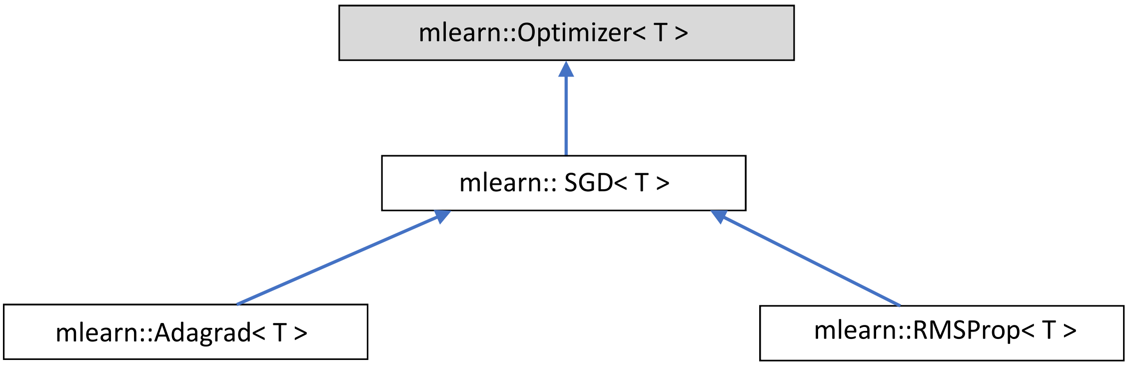


Figure 2.4 Inheritance diagram for mlearn::Optimizer< T >

```
// declares train parameters
double learning_rate = 0.05, lambda = 0.1, alpha = 0.05, beta = 0.02, cost, accuracy;
uint32_t batch_size = 50, num_epochs = 30;
// declares MNIST train and validation objects
MNISTReader<double> validation, train("data/mnist_train.csv", ',',  false);
// read data
train.read();
// split into train and validation set
train.trainTestSplit(validation, 0.1);
// create vector of indices used for shuffling
std::vector<int> indices;
// hidden activation is sigmoid and output softmax
Activation<double> hidden("sigmoid"), output("softmax");
// creates a hidden and output layers
Layer<double> hidden_layer(784, 100, "hidden", hidden);
Layer<double> output_layer(100, 10, "output", output);
// creates a Network object, objective function is MSE
Network<double> mlp (new MSE<double>);
// adds the layers to the network
mlp.addLayer(hidden_layer);
mlp.addLayer(output_layer);
// connects the layers
mlp.connectLayers();
// creates a pointer to Optimizer object, optimizer is SGD
Optimizer<double>* opt = new SGD<double>(learning_rate, batch_size, num_epochs, lambda, reg);
// calls the train method to train the model
opt->train(mlp, model_file, &train, &validation);
```

2.3 Serialization - Saving and Loading Models

Trained models can be saved and loaded back either for test or resumption of training. This feature is implemented with the boost::serialization library. The saveModel function, which saves "layers" member of the Network class, is as follows:

```
void Network<T>::saveModel(std::string model_file)
{
    std::cout « "Saving model..." « std::endl;
    std::ofstream ofs(model_file);
    if (!ofs.good())
    {
        throw std::ios::failure("Error opening file!");
    }
    boost::archive::binary_oarchive oa(ofs);
    oa & layers;
}
```

The corresponding loadModel function is as follows:

```
Network<T>& Network<T>::loadModel(std::string model_file)
{
    std::cout « "Loading model... " « std::endl;
    std::ifstream ifs(model_file);
    if (!ifs.good())
    {
        throw std::ios::failure("Error opening file!");
    }
    boost::archive::binary_iarchive ia(ifs);
```

```
    ia & layers;
    this->connectLayers();
    return *this;
}
```

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 mlearn Namespace Reference

Classes

- class Activation
- class Adagrad
- class CostFunction
- class CrossEntropy
- class DataReader
- class GenericReader
- class IrisReader
- class Layer
- class MAE
- class MNISTReader
- class MSE
- class NetNode
- class Network
- class Node
- class Optimizer
- class RMSProp
- class SGD

Functions

- template< class T >
  std::vector< Node< T > ∗ > & destroy (std::vector< Node< T > ∗ > &argv)
- template< class T >
  double mean (mublas::matrix< T > argv)
- template< class T >
  Network< T > & SGDHelper (Network< T > &model, uint32_t batch_size, uint32_t num_epochs, Optimizer< T > ∗opt, const DataReader< T > ∗train, const DataReader< T > ∗validation=nullptr, std::string id="sgd")

7.1.1 Function Documentation

7.1.1.1 destroy()

```
template<class T >
std::vector<Node<T>*>& mlearn::destroy (
            std::vector< Node< T > * > & argv )
```

Releases dynamically allocated vector of pointers to Node objects and clears the vector.

Parameters

argv	Vector of pointers to Node

Returns

Empty vector

7.1.1.2 mean()

```
template<class T >
double mlearn::mean (
            mublas::matrix< T > argv )
```

Computes the mean of values of a matrix.

Parameters

argv	The matrix

Returns

The mean

7.1.1.3 SGDHelper()

```
template<class T >
Network<T>& mlearn::SGDHelper (
            Network< T > & model,
            uint32_t batch_size,
            uint32_t num_epochs,
            Optimizer< T > * opt,
            const DataReader< T > * train,
            const DataReader< T > * validation = nullptr,
            std::string id = "sgd" )
```

The base optimizer function that implements vanilla SGD. Other optimizers call the SGDHelper function.

Parameters

model	Model object to train
batch_size	Train batch size
num_epochs	Number of train epochs
opt	Pointer to optimizer
train	Pointer to train dataset
validation	Pointer to validation dataset. Default is nullptr
id	Type of optimizer

Returns

A reference to the trained model

Chapter 8

Class Documentation

8.1 mlearn::Activation< T > Class Template Reference

```
#include <libutil.h>
```

Public Member Functions

- Activation ()
- Activation (std::string type)
- Activation (std::string type, double alpha)
- NetNode< T > & compute (NetNode< T > &input)
- NetNode< T > & computeDerivative (NetNode< T > &input)
- mublas::matrix< T > & computeDerivative (NetNode< T > &, mublas::matrix< T > &jacobian_m)
- std::string getType ()

Protected Member Functions

- template<class Archive >
  void serialize (Archive &ar, const uint64_t version)

Protected Attributes

- std::string type {"sigmoid"}
- double alpha {0.0}

Friends

- class boost::serialization::access

8.1.1 Detailed Description

template<class T>
class mlearn::Activation< T >

The Activation class handles activations in the network. Currently, functions implemented are sigmoid, tanh, ReLU, leaky ReLU, identity, softmax and ELU.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 Activation() [1/3]

```
template<class T>
mlearn::Activation< T >::Activation ( )  [inline]
```

The default constructor

8.1.2.2 Activation() [2/3]

```
template<class T>
mlearn::Activation< T >::Activation (
            std::string type )  [inline]
```

Overloaded constructor.

Parameters

type	The type of activation function.

8.1.2.3 Activation() [3/3]

```
template<class T>
mlearn::Activation< T >::Activation (
            std::string type,
            double alpha )  [inline]
```

Overloaded constructor.

Parameters

type	Type of activation function
alpha	Parameter for some functions

8.1.3 Member Function Documentation

8.1.3.1 compute()

```
template<class T>
NetNode<T>& mlearn::Activation< T >::compute (
            NetNode< T > & input )
```

Computes the activation function.

Parameters

input	Input (type NetNode) to the function

Returns

Computed value of the function

8.1.3.2 computeDerivative() [1/2]

```
template<class T>
mublas::matrix<T>& mlearn::Activation< T >::computeDerivative (
            NetNode< T > & ,
            mublas::matrix< T > & jacobian_m )
```

Overloaded function that computes the derivative of activation function. This returns a 2D Jacobian matrix.

Parameters

input	Input (type NetNode) to the function
jacobian_m	An empty 2D Jacobian matrix

Returns

Computed derivative of the function.

8.1.3.3 computeDerivative() [2/2]

```
template<class T>
NetNode<T>& mlearn::Activation< T >::computeDerivative (
            NetNode< T > & input )
```

Overloaded function that computes the derivative of activation function.

Parameters

input	Input (type NetNode) to the function

Returns

Computed derivative of the function

8.1.3.4 getType()

```
template<class T>
std::string mlearn::Activation< T >::getType ( ) [inline]
```

Accessor function that returns the type of activation function

8.1.3.5 serialize()

```
template<class T>
template<class Archive >
void mlearn::Activation< T >::serialize (
            Archive & ar,
            const uint64_t version ) [inline], [protected]
```

8.1.4 Friends And Related Function Documentation

8.1.4.1 boost::serialization::access

```
template<class T>
friend class boost::serialization::access [friend]
```

The is responsible for saving/serialization of members

8.1.5 Member Data Documentation

8.1.5.1 alpha

```
template<class T>
double mlearn::Activation< T >::alpha {0.0} [protected]
```

A parameter used in some functions such as ELU, leaky ReLU

8.1.5.2 type

```
template<class T>
std::string mlearn::Activation< T >::type {"sigmoid"} [protected]
```

The type of function, default sigmoid

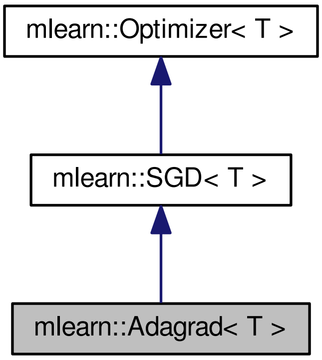
The documentation for this class was generated from the following file:

- libutil.h

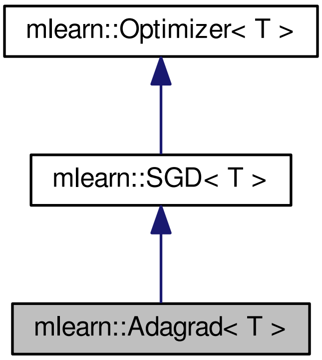
8.2 mlearn::Adagrad< T > Class Template Reference

```
#include <optimizer.h>
```

Inheritance diagram for mlearn::Adagrad< T >:



Collaboration diagram for mlearn::Adagrad< T >:



Public Member Functions

- Adagrad ()
- Adagrad (double learning_rate, uint32_t batch_size, uint32_t n_epochs, double lambda, std::string reg, double beta)
- Adagrad (double learning_rate, uint32_t batch_size, uint32_t n_epochs)
- Adagrad (double learning_rate, uint32_t batch_size, uint32_t n_epochs, double lambda, std::string reg)

- Network< T > & train (Network< T > &, std::string, const DataReader< T > ∗, const DataReader< T > ∗=nullptr, std::string="adagrad")
- virtual Network< T > & update (Network< T > &)
- double predict (Network< T > &, const DataReader< T > ∗, std::string)
- virtual ∼Adagrad ()

Additional Inherited Members

8.2.1 Detailed Description

template<class T>
class mlearn::Adagrad< T >

The Adagrad class extends the SGD class. It implements the adaptive rate SGD. The only difference as compared to SGD is how the parameters are updated. Adagrad adapts the learning rate to each parameter.

Adaptive gradient method J Duchi, E Hazan and Y Singer, Adaptive subgradient methods for online learning and stochastic optimization The Journal of Machine Learning Research, pages 2121-2159, 2011.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Adagrad() [1/4]

```
template<class T >
mlearn::Adagrad< T >::Adagrad ( )  [inline]
```

Default constructor

8.2.2.2 Adagrad() [2/4]

```
template<class T >
mlearn::Adagrad< T >::Adagrad (
            double learning_rate,
            uint32_t batch_size,
            uint32_t n_epochs,
            double lambda,
            std::string reg,
            double beta )  [inline]
```

Overloaded constructor with 6 arguments

8.2.2.3 Adagrad() [3/4]

```
template<class T >
mlearn::Adagrad< T >::Adagrad (
            double learning_rate,
            uint32_t batch_size,
            uint32_t n_epochs )  [inline]
```

Overloaded constructor with 3 arguments

8.2.2.4 Adagrad() [4/4]

```
template<class T >
mlearn::Adagrad< T >::Adagrad (
            double learning_rate,
            uint32_t batch_size,
            uint32_t n_epochs,
            double lambda,
            std::string reg )  [inline]
```

Overloaded constructor with 5 arguments

8.2.2.5 ∼Adagrad()

```
template<class T >
virtual mlearn::Adagrad< T >::~Adagrad ( )  [inline], [virtual]
```

Virtual destructor

8.2.3 Member Function Documentation

8.2.3.1 predict()

```
template<class T >
double mlearn::Adagrad< T >::predict (
            Network< T > & ,
            const DataReader< T > * ,
            std::string  )  [virtual]
```

Implement predict function

Reimplemented from mlearn::SGD< T >.

8.2.3.2 train()

```
template<class T >
Network<T>& mlearn::Adagrad< T >::train (
            Network< T > & ,
            std::string ,
            const DataReader< T > * ,
            const DataReader< T > *  = nullptr,
            std::string  = "adagrad" )  [virtual]
```

Implement train function

Reimplemented from mlearn::SGD< T >.

8.2.3.3 update()

```
template<class T >
virtual Network<T>& mlearn::Adagrad< T >::update (
            Network< T > & ) [virtual]
```

Implement update function

Reimplemented from mlearn::SGD< T >.

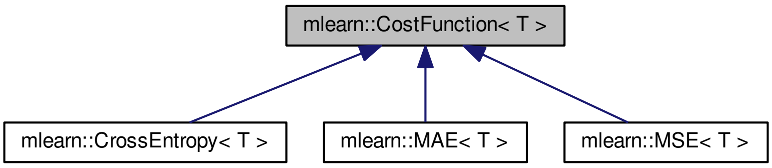
The documentation for this class was generated from the following file:

- optimizer.h

8.3 mlearn::CostFunction< T > Class Template Reference

```
#include <libutil.h>
```

Inheritance diagram for mlearn::CostFunction< T >:



Public Member Functions

- CostFunction ()
- CostFunction (std::string id)
- std::string getId () const
- virtual double cost (const NetNode< T > &prediction, const NetNode< T > &label)=0
- virtual double accuracy (const NetNode< T > &prediction, const NetNode< T > &label)=0
- virtual NetNode< T > & costDerivative (const NetNode< T > &, const NetNode< T > &, NetNode< T > &)=0
- virtual ∼CostFunction ()

Protected Attributes

- std::string id

8.3.1 Detailed Description

template<class T>
class mlearn::CostFunction< T >

The CostFunction class is responsible for objective functions. Cost functions implemented are mean squared error (MSE), mean absolute error (MAE) and cross entropy.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 CostFunction() [1/2]

```
template<class T>
mlearn::CostFunction< T >::CostFunction ( )  [inline]
```

Default constructor

8.3.2.2 CostFunction() [2/2]

```
template<class T>
mlearn::CostFunction< T >::CostFunction (
            std::string id )  [inline]
```

Constructor that takes a single parameter: id

8.3.2.3 ~CostFunction()

```
template<class T>
virtual mlearn::CostFunction< T >::~CostFunction ( )  [inline], [virtual]
```

Virtual destructor

8.3.3 Member Function Documentation

8.3.3.1 accuracy()

```
template<class T>
virtual double mlearn::CostFunction< T >::accuracy (
            const NetNode< T > & prediction,
            const NetNode< T > & label )  [pure virtual]
```

Pure virtual function

Implemented in mlearn::MAE< T >, mlearn::CrossEntropy< T >, and mlearn::MSE< T >.

8.3.3.2 cost()

```
template<class T>
virtual double mlearn::CostFunction< T >::cost (
            const NetNode< T > & prediction,
            const NetNode< T > & label )  [pure virtual]
```

Pure virtual function

Implemented in mlearn::MAE< T >, mlearn::CrossEntropy< T >, and mlearn::MSE< T >.

8.3.3.3 costDerivative()

```
template<class T>
virtual NetNode<T>& mlearn::CostFunction< T >::costDerivative (
            const NetNode< T > & ,
            const NetNode< T > & ,
            NetNode< T > & )  [pure virtual]
```

Pure virtual function

Implemented in mlearn::MAE< T >, mlearn::CrossEntropy< T >, and mlearn::MSE< T >.

8.3.3.4 getId()

```
template<class T>
std::string mlearn::CostFunction< T >::getId ( ) const  [inline]
```

Accessor function that returns the id of the cost function

8.3.4 Member Data Documentation

8.3.4.1 id

```
template<class T>
std::string mlearn::CostFunction< T >::id  [protected]
```

Type of cost function

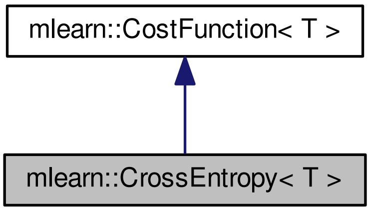
The documentation for this class was generated from the following file:

- libutil.h

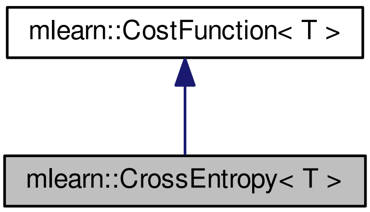
8.4 mlearn::CrossEntropy< T > Class Template Reference

```
#include <libutil.h>
```

Inheritance diagram for mlearn::CrossEntropy< T >:



Collaboration diagram for mlearn::CrossEntropy< T >:



Public Member Functions

- CrossEntropy ()
- double cost (const NetNode< T > &prediction, const NetNode< T > &label)
- double accuracy (const NetNode< T > &prediction, const NetNode< T > &label)
- NetNode< T > & costDerivative (const NetNode< T > &prediction, const NetNode< T > &label, NetNode< T > &result)

Additional Inherited Members

8.4.1 Detailed Description

template< class T >
class mlearn::CrossEntropy< T >

CrossEntropy derives from the class CostFunction. It implements the "cost", "accuracy" and "costDerivative" functions. Used for classification problems.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 CrossEntropy()

```
template<class T >
mlearn::CrossEntropy< T >::CrossEntropy ( ) [inline]
```

Default constructor

8.4.3 Member Function Documentation

8.4.3.1 accuracy()

```
template<class T >
double mlearn::CrossEntropy< T >::accuracy (
            const NetNode< T > & prediction,
            const NetNode< T > & label ) [virtual]
```

Computes the accuracy between prediction and label. This is +1 if prediction and label are same and 0 otherwise.

Parameters

prediction	Hypotheses
label	Reference

Returns

+1 if prediction and label are same and 0 otherwise

Implements mlearn::CostFunction< T >.

8.4.3.2 cost()

```
template<class T >
double mlearn::CrossEntropy< T >::cost (
            const NetNode< T > & prediction,
            const NetNode< T > & label ) [virtual]
```

Computes the loss between prediction and label. This is the negative of product of label and log of prediction.

Parameters

prediction	Hypotheses
label	Reference

Returns

Product of negative log of prediction and label

Implements mlearn::CostFunction< T >.

8.4.3.3 costDerivative()

```
template<class T >
NetNode<T>& mlearn::CrossEntropy< T >::costDerivative (
            const NetNode< T > & prediction,
            const NetNode< T > & label,
            NetNode< T > & result )  [virtual]
```

Computes the derivative. This is implemented as the difference between the prediction and label.

Parameters

prediction	Hypotheses
label	Reference
result	Result

Returns

The difference between the prediction and label

Implements mlearn::CostFunction< T >.

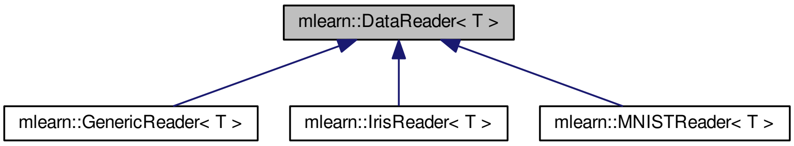
The documentation for this class was generated from the following file:

- libutil.h

8.5 mlearn::DataReader< T > Class Template Reference

```
#include <data_reader.h>
```

Inheritance diagram for mlearn::DataReader< T >:



Public Member Functions

- DataReader ()
- DataReader (const std::string file_name)
- DataReader (const std::string file_name, uint64_t feature_dim, uint64_t label_dim)
- DataReader (const std::string file_name, uint64_t feature_dim, uint64_t label_dim, bool one_hot)
- DataReader (const std::string file_name, uint64_t feature_dim, uint64_t label_dim, char sep, bool header)
- DataReader (const std::string file_name, uint64_t feature_dim, uint64_t label_dim, char sep)
- DataReader (const DataReader< T > &arg)
- const std::vector< Node< T > ∗ > & getFeatures () const
- const std::vector< Node< T > ∗ > & getLabels () const
- uint64_t getFeatureDim () const
- uint64_t getLabelDim () const
- uint64_t getRowDim () const
- virtual DataReader< T > & read ()=0
- DataReader< T > & operator= (const DataReader< T > &)
- std::vector< int > & shuffleIndex (std::vector< int > &indices) const
- DataReader< T > & trainTestSplit (DataReader< T > &test, double test_size=0.0)
- DataReader< T > & destroy ()
- virtual ∼DataReader ()

Protected Attributes

- const std::string file_name
- std::vector< Node< T > ∗ > features
- std::vector< Node< T > ∗ > labels
- uint64_t feature_dim
- uint64_t label_dim
- uint64_t row_dim
- bool one_hot {true}
- char sep {','}
- bool header {false}

8.5.1 Detailed Description

template<class T>
class mlearn::DataReader< T >

The DataReader class is the base class responsible for reading train/test dataset into features/labels. 3 different readers are implemented, namely, MNISTReader, GenericReader and IrisReader. They extend the base class DataReader.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 DataReader() [1/7]

```
template<class T>
mlearn::DataReader< T >::DataReader ( )  [inline]
```

The default constructor

8.5.2.2 DataReader() [2/7]

```
template<class T>
mlearn::DataReader< T >::DataReader (
            const std::string file_name )  [inline]
```

Overloaded constructor with 1 argument

8.5.2.3 DataReader() [3/7]

```
template<class T>
mlearn::DataReader< T >::DataReader (
            const std::string file_name,
            uint64_t feature_dim,
            uint64_t label_dim )  [inline]
```

Overloaded constructor with 3 arguments

8.5.2.4 DataReader() [4/7]

```
template<class T>
mlearn::DataReader< T >::DataReader (
            const std::string file_name,
            uint64_t feature_dim,
            uint64_t label_dim,
            bool one_hot )  [inline]
```

Overloaded constructor with 4 arguments

8.5.2.5 DataReader() [5/7]

```
template<class T>
mlearn::DataReader< T >::DataReader (
            const std::string file_name,
            uint64_t feature_dim,
            uint64_t label_dim,
            char sep,
            bool header )  [inline]
```

Overloaded constructor with 5 arguments

8.5.2.6 DataReader() [6/7]

```
template<class T>
mlearn::DataReader< T >::DataReader (
            const std::string file_name,
            uint64_t feature_dim,
            uint64_t label_dim,
            char sep ) [inline]
```

Overloaded constructor with 4 argument

8.5.2.7 DataReader() [7/7]

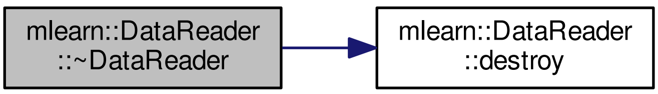
```
template<class T>
mlearn::DataReader< T >::DataReader (
            const DataReader< T > & arg ) [inline]
```

Copy constructor

8.5.2.8 ∼DataReader()

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

Virtual destructor Here is the call graph for this function:



8.5.3 Member Function Documentation

8.5.3.1 destroy()

```
template<class T>
DataReader<T>& mlearn::DataReader< T >::destroy ( )
```

Responsible for releasing/deleting dynamically allocated memory.

Returns

Reference to empty DataReader object

8.5.3.2 getFeatureDim()

```
template<class T>
uint64_t mlearn::DataReader< T >::getFeatureDim ( ) const  [inline]
```

Returns feature dimension

8.5.3.3 getFeatures()

```
template<class T>
const std::vector<Node<T>*>& mlearn::DataReader< T >::getFeatures ( ) const  [inline]
```

Returns features

8.5.3.4 getLabelDim()

```
template<class T>
uint64_t mlearn::DataReader< T >::getLabelDim ( ) const  [inline]
```

Returns label dimension

8.5.3.5 getLabels()

```
template<class T>
const std::vector<Node<T>*>& mlearn::DataReader< T >::getLabels ( ) const  [inline]
```

Returns labels

8.5.3.6 getRowDim()

```
template<class T>
uint64_t mlearn::DataReader< T >::getRowDim ( ) const  [inline]
```

Returns number of instances

8.5.3.7 operator=()

```
template<class T>
DataReader<T>& mlearn::DataReader< T >::operator= (
            const DataReader< T > &  )
```

Assignment operator

8.5.3.8 read()

```
template<class T>
virtual DataReader<T>& mlearn::DataReader< T >::read ( )  [pure virtual]
```

Reads dataset file into features and labels

Implemented in mlearn::GenericReader< T >, mlearn::IrisReader< T >, and mlearn::MNISTReader< T >.

8.5.3.9 shuffleIndex()

```
template<class T>
std::vector<int>& mlearn::DataReader< T >::shuffleIndex (
             std::vector< int > & indices ) const
```

This is for shuffling train dataset for use with stochastic gradient descent optimizers and variants.

Parameters

indices	Indices of training data

Returns

A reference to the shuffled indices

8.5.3.10 trainTestSplit()

```
template<class T>
DataReader<T>& mlearn::DataReader< T >::trainTestSplit (
            DataReader< T > & test,
            double test_size = 0.0 )
```

This splits train dataset into train/validation set.

Parameters

test	An empty DataReader object that will hold validation set

Returns

test_size Percentage of train set used for validation

8.5.4 Member Data Documentation

8.5.4.1 feature_dim

```
template<class T>
uint64_t mlearn::DataReader< T >::feature_dim  [protected]
```

Dimension of Features

8.5.4.2 features

```
template<class T>
std::vector<Node<T>*> mlearn::DataReader< T >::features  [protected]
```

Vector of Node pointers for Features

8.5.4.3 file_name

```
template<class T>
const std::string mlearn::DataReader< T >::file_name [protected]
```

Name of input file

8.5.4.4 header

```
template<class T>
bool mlearn::DataReader< T >::header {false} [protected]
```

Dataset can contain header information as the first line

8.5.4.5 label_dim

```
template<class T>
uint64_t mlearn::DataReader< T >::label_dim [protected]
```

Dimension of labels

8.5.4.6 labels

```
template<class T>
std::vector<Node<T>*> mlearn::DataReader< T >::labels [protected]
```

Vector of Node pointers for labels

8.5.4.7 one_hot

```
template<class T>
bool mlearn::DataReader< T >::one_hot {true} [protected]
```

Encoding of labels as one-hot

8.5.4.8 row_dim

```
template<class T>
uint64_t mlearn::DataReader< T >::row_dim [protected]
```

Number of instances

8.5.4.9 sep

```
template<class T>
char mlearn::DataReader< T >::sep {','} [protected]
```

Character delimiter

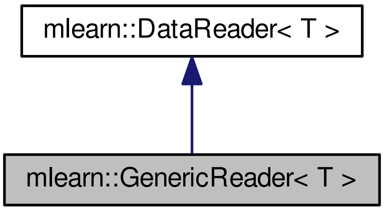
The documentation for this class was generated from the following file:

- data_reader.h

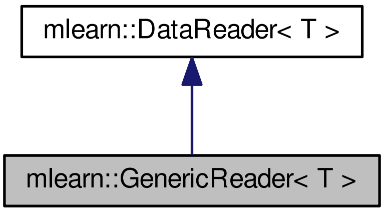
8.6 mlearn::GenericReader< T > Class Template Reference

```
#include <data_reader.h>
```

Inheritance diagram for mlearn::GenericReader< T >:



Collaboration diagram for mlearn::GenericReader< T >:



Public Member Functions

- GenericReader ()
- GenericReader (const std::string file_name, char sep, bool header)
- GenericReader (const std::string file_name, uint64_t feature_dim, uint64_t label_dim, char sep)
- GenericReader (const std::string file_name, uint64_t feature_dim, uint64_t label_dim, char sep, bool header)
- GenericReader (const std::string file_name, uint64_t feature_dim, uint64_t label_dim, bool one_hot)
- GenericReader (const GenericReader< T > &arg)
- GenericReader< T > & operator= (const GenericReader< T > &)
- GenericReader< T > & read ()
- virtual ~GenericReader ()

Additional Inherited Members

8.6.1 Detailed Description

template<class T>
class mlearn::GenericReader< T >

The GenericReader class extends the DataReader class and implements the "read" method. It is responsible for reading any text dataset into features/labels. The file can contain header information. Each row of sample should be concatenation of features and labels, where "sep" is the delimiter.

The following is an example for the "xor" dataset. The line "x1,x2,y1" is the header. x1 and x2 are the features and y1 the label. "0,0,0" is an instance containing features and labels. The delimiter is a comma.

```
x1,x2,y1
0,0,0
0,1,1
1,0,1
1,1,0
```

8.6.2 Constructor & Destructor Documentation

8.6.2.1 GenericReader() [1/6]

```
template<class T>
mlearn::GenericReader< T >::GenericReader ( )  [inline]
```

8.6.2.2 GenericReader() [2/6]

```
template<class T>
mlearn::GenericReader< T >::GenericReader (
            const std::string file_name,
            char sep,
            bool header )  [inline]
```

8.6.2.3 GenericReader() [3/6]

```
template<class T>
mlearn::GenericReader< T >::GenericReader (
            const std::string file_name,
            uint64_t feature_dim,
            uint64_t label_dim,
            char sep )  [inline]
```

8.6.2.4 GenericReader() [4/6]

```
template<class T>
mlearn::GenericReader< T >::GenericReader (
            const std::string file_name,
            uint64_t feature_dim,
            uint64_t label_dim,
            char sep,
            bool header )  [inline]
```

8.6.2.5 GenericReader() [5/6]

```
template<class T>
mlearn::GenericReader< T >::GenericReader (
            const std::string file_name,
            uint64_t feature_dim,
            uint64_t label_dim,
            bool one_hot )  [inline]
```

8.6.2.6 GenericReader() [6/6]

```
template<class T>
mlearn::GenericReader< T >::GenericReader (
            const GenericReader< T > & arg )  [inline]
```

8.6.2.7 ∼GenericReader()

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

8.6.3 Member Function Documentation

8.6.3.1 operator=()

```
template<class T>
GenericReader<T>& mlearn::GenericReader< T >::operator= (
            const GenericReader< T > &  )
```

8.6.3.2 read()

```
template<class T>
GenericReader<T>& mlearn::GenericReader< T >::read ( )  [virtual]
```

Reads dataset file into features and labels

Implements mlearn::DataReader< T >.

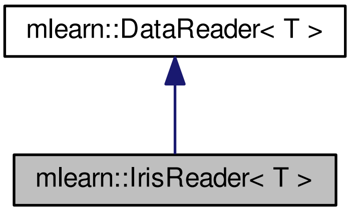
The documentation for this class was generated from the following file:

- data_reader.h

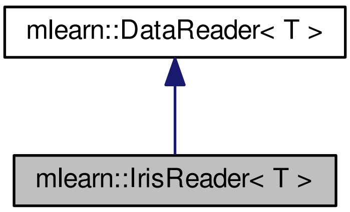
8.7 mlearn::IrisReader< T > Class Template Reference

```
#include <data_reader.h>
```

Inheritance diagram for mlearn::IrisReader< T >:



Collaboration diagram for mlearn::IrisReader< T >:



Public Member Functions

- IrisReader ()
- IrisReader (const std::string file_name)
- IrisReader (const std::string file_name, char sep, bool header)
- IrisReader (const std::string file_name, uint64_t feature_dim, uint64_t label_dim, char sep)
- IrisReader (const IrisReader< T > &arg)
- IrisReader< T > & operator= (const IrisReader< T > &)
- IrisReader< T > & read ()
- virtual ~IrisReader ()

Additional Inherited Members

8.7.1 Detailed Description

template<class T>
class mlearn::IrisReader< T >

The IrisReader class extends the DataReader class. It is responsible for reading the Iris dataset into feature/label (`https://archive.ics.uci.edu/ml/datasets/iris`) The class implements the "read" method specific to the Iris dataset.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 IrisReader() [1/5]

```
template<class T>
mlearn::IrisReader< T >::IrisReader ( )  [inline]
```

8.7.2.2 IrisReader() [2/5]

```
template<class T>
mlearn::IrisReader< T >::IrisReader (
            const std::string file_name )  [inline]
```

8.7.2.3 IrisReader() [3/5]

```
template<class T>
mlearn::IrisReader< T >::IrisReader (
            const std::string file_name,
            char sep,
            bool header )  [inline]
```

8.7.2.4 IrisReader() [4/5]

```
template<class T>
mlearn::IrisReader< T >::IrisReader (
            const std::string file_name,
            uint64_t feature_dim,
            uint64_t label_dim,
            char sep ) [inline]
```

8.7.2.5 IrisReader() [5/5]

```
template<class T>
mlearn::IrisReader< T >::IrisReader (
            const IrisReader< T > & arg ) [inline]
```

8.7.2.6 ∼IrisReader()

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

8.7.3 Member Function Documentation

8.7.3.1 operator=()

```
template<class T>
IrisReader<T>& mlearn::IrisReader< T >::operator= (
            const IrisReader< T > & )
```

8.7.3.2 read()

```
template<class T>
IrisReader<T>& mlearn::IrisReader< T >::read ( ) [virtual]
```

Reads dataset file into features and labels

Implements mlearn::DataReader< T >.

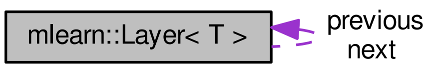
The documentation for this class was generated from the following file:

- data_reader.h

8.8 mlearn::Layer< T > Class Template Reference

```
#include <layer.h>
```

Collaboration diagram for mlearn::Layer< T >:



Public Member Functions

- Layer ()
- Layer (uint64_t input_dim, uint64_t output_dim, std::string type, Activation< T > act_function)
- Layer< T > & initialize ()
- Layer< T > & push_row (uint64_t row_id, const std::vector< T > &in_data)
- void connect (Layer< T > &layer)
- NetNode< T > & getForwardInput (NetNode< T > &out)
- NetNode< T > & forwardProp ()
- NetNode< T > & getBackwardInput (NetNode< T > &out)
- NetNode< T > & backwardProp ()
- const mublas::matrix< T > & getWeight () const
- const NetNode< T > & getBias () const
- const mublas::matrix< T > & getDeltaWeight () const
- const NetNode< T > & getDeltaBias () const
- const NetNode< T > & getInputData () const
- const NetNode< T > & getOutputData () const
- const NetNode< T > & getInputDelta () const
- const NetNode< T > & getOutputDelta () const
- void setWeight (const mublas::matrix< T > &in_weight)
- void setBias (const NetNode< T > &in_bias)
- void setDeltaWeight (const mublas::matrix< T > &in_delta_w)
- void setDeltaBias (const NetNode< T > &in_delta_b)
- void setInputData (const NetNode< T > &in_data)
- void setOutputData (const NetNode< T > &out_data)
- void setInputDelta (const NetNode< T > &in_delta)
- void setOutputDelta (const NetNode< T > &out_delta)
- Layer< T > & clearDeltas (double beta=0.0)
- Layer< T > & regularize (double rate, double lambda, std::string reg="None")
- Layer< T > & regularize (const mublas::matrix< T > &rate, double lambda, std::string reg="None")
- Layer< T > & updateParams (double rate, uint32_t batch_size, double lambda, std::string reg, double beta)
- Layer< T > & updateParams (double rate, uint32_t batch_size, double lambda, std::string reg, double beta, bool change_rate, std::string="adagrad")
- virtual ∼Layer ()

Protected Member Functions

- template<class Archive >
  void serialize (Archive &ar, const uint64_t version)

Protected Attributes

- uint64_t input_dim
- uint64_t output_dim
- Layer * previous
- Layer * next
- NetNode< T > output_data
- NetNode< T > output_delta
- NetNode< T > input_data
- NetNode< T > input_delta
- NetNode< T > bias
- NetNode< T > delta_b
- NetNode< T > activation
- mublas::matrix< T > weight
- mublas::matrix< T > delta_w
- mublas::matrix< T > regularize_w
- mublas::matrix< T > momentum_w
- mublas::matrix< T > sq_delta_w
- std::string type
- Activation< T > act_function

Friends

- class boost::serialization::access

8.8.1 Detailed Description

template<class T>
class mlearn::Layer< T >

The Layer class represents a layer in a neural network. The pointers "previous" and "next" point to the previous and next layers respectively. Each layer has a forward function(forwardProp) that produces output data from input data and a backward function(backwardProp) that produces an output delta (gradient) from an input delta. A layer can be a hidden or an output layer.

```
//  Creates 2 Activation objects: hidden and output
Activation<double> hidden("sigmoid"), output("softmax");
//  A hidden layer with input and output dimensions 2
//  Activation function used is sigmoid
Layer<double> hidden_layer(2, 2, "hidden", hidden);
//  An output layer with input and output dimensions 2 and 1 respectively.
//  Activation function used is softmax
Layer<double> output_layer(2, 1, "output", output);
```

Note

The output dimension of the last hidden layer must be same as the input dimension of the output layer.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 Layer() [1/2]

```
template<class T>
mlearn::Layer< T >::Layer ( )  [inline]
```

Default constructor

8.8.2.2 Layer() [2/2]

```
template<class T>
mlearn::Layer< T >::Layer (
            uint64_t input_dim,
            uint64_t output_dim,
            std::string type,
            Activation< T > act_function )  [inline]
```

Overloaded constructor with 4 arguments

8.8.2.3 ∼Layer()

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

Virtual destructor

8.8.3 Member Function Documentation

8.8.3.1 backwardProp()

```
template<class T>
NetNode<T>& mlearn::Layer< T >::backwardProp ( )
```

Propagates input error (delta) backward and produces an output delta.

Returns

output_delta Output delta

8.8.3.2 clearDeltas()

```
template<class T>
Layer<T>& mlearn::Layer< T >::clearDeltas (
            double beta = 0.0 )
```

After a single forward and backward pass, the parameters of each layer is updated, and the deltas reset to zero. Beta is a momentum to use to decide if a part of a previous weight update should be used. default is 0 (no momentum).

Parameters

beta	Momentum term/parameter

Returns

Reference to self

8.8.3.3 connect()

```
template<class T>
void mlearn::Layer< T >::connect (
            Layer< T > & layer )
```

Connects a layer to its neighbors.

Parameters

layer	Layer to be connected
in_data	Vector of data

Returns

Returns void

8.8.3.4 forwardProp()

```
template<class T>
NetNode<T>& mlearn::Layer< T >::forwardProp ( )
```

An input data is propagated forward through a layer and produces an output data.

Returns

output_data Output data

8.8.3.5 getBackwardInput()

```
template<class T>
NetNode<T>& mlearn::Layer< T >::getBackwardInput (
            NetNode< T > & out )
```

The last layer gets input delta from the derivative of the cost function, while other layers get their input deltas from their successors.

Returns

out Input delta

8.8.3.6 getBias()

```
template<class T>
const NetNode<T>& mlearn::Layer< T >::getBias ( ) const  [inline]
```

Returns the bias vector of a layer in a NetNode object

8.8.3.7 getDeltaBias()

```
template<class T>
const NetNode<T>& mlearn::Layer< T >::getDeltaBias ( ) const  [inline]
```

Returns the delta bias vector of a layer in a NetNode object

8.8.3.8 getDeltaWeight()

```
template<class T>
const mublas::matrix<T>& mlearn::Layer< T >::getDeltaWeight ( ) const  [inline]
```

Returns the delta weight matrix of a layer

8.8.3.9 getForwardInput()

```
template<class T>
NetNode<T>& mlearn::Layer< T >::getForwardInput (
             NetNode< T > & out )
```

The first layer gets input data from the training data, while other layers get their inputs from the output of previous layer.

Parameters

out	An empty NetNode

Returns

out Input data

8.8.3.10 getInputData()

```
template<class T>
const NetNode<T>& mlearn::Layer< T >::getInputData ( ) const  [inline]
```

Returns the input data to a layer in a NetNode object

8.8.3.11 getInputDelta()

```
template<class T>
const NetNode<T>& mlearn::Layer< T >::getInputDelta ( ) const  [inline]
```

Returns the input delta to a layer in a NetNode object

8.8.3.12 getOutputData()

```
template<class T>
const NetNode<T>& mlearn::Layer< T >::getOutputData ( ) const  [inline]
```

Returns the output data from a layer in a NetNode object

8.8.3.13 getOutputDelta()

```
template<class T>
const NetNode<T>& mlearn::Layer< T >::getOutputDelta ( ) const  [inline]
```

Returns the output delta from a layer in a NetNode object

8.8.3.14 getWeight()

```
template<class T>
const mublas::matrix<T>& mlearn::Layer< T >::getWeight ( ) const  [inline]
```

Returns the weight matrix of a layer

8.8.3.15 initialize()

```
template<class T>
Layer<T>& mlearn::Layer< T >::initialize ( )
```

Used to initialize weight in layers. Generates random values between a range. The values are uniformly sampled between sqrt(-6./(input_dim + output_dim)) $*$ 4 and sqrt(6./(input_dim + output_dim)) $*$ 4. (Y. Bengio, X. Glorot, Understanding the difficulty of training deep feedforward neuralnetworks, AISTATS 2010).

8.8.3.16 push_row()

```
template<class T>
Layer<T>& mlearn::Layer< T >::push_row (
            uint64_t row_id,
            const std::vector< T > & in_data )  [inline]
```

Uses vector to initialize rows of weight matrix.

Parameters

row_id	Matrix row index
in_data	Vector of data

Returns

Reference to self

8.8.3.17 regularize() [1/2]

```
template<class T>
Layer<T>& mlearn::Layer< T >::regularize (
            const mublas::matrix< T > & rate,
            double lambda,
            std::string reg = "None" )
```

Overloaded function. Handles regularization for Adagrad/RMSProp.

Parameters

rate	Learning rate
lambda	Regularization parameter
reg	Type of regularization (L1, L2 or None)

Returns

Reference to self

8.8.3.18 regularize() [2/2]

```
template<class T>
Layer<T>& mlearn::Layer< T >::regularize (
            double rate,
            double lambda,
            std::string reg = "None" )
```

Overloaded function. Handles regularization of layer parameters (weight). Currently, only L1 and L2 are implemented. The default is no regularization(None).

Parameters

rate	Learning rate
lambda	Regularization parameter
reg	Type of regularization (L1, L2 or None)

Returns

Reference to self

8.8.3.19 serialize()

```
template<class T>
template<class Archive >
void mlearn::Layer< T >::serialize (
            Archive & ar,
            const uint64_t version ) [inline], [protected]
```

8.8.3.20 setBias()

```
template<class T>
void mlearn::Layer< T >::setBias (
            const NetNode< T > & in_bias ) [inline]
```

Sets the bias vector of a layer

8.8.3.21 setDeltaBias()

```
template<class T>
void mlearn::Layer< T >::setDeltaBias (
            const NetNode< T > & in_delta_b ) [inline]
```

Sets the bias vector of a layer

8.8.3.22 setDeltaWeight()

```
template<class T>
void mlearn::Layer< T >::setDeltaWeight (
            const mublas::matrix< T > & in_delta_w ) [inline]
```

Sets the delta weight matrix of a layer

8.8.3.23 setInputData()

```
template<class T>
void mlearn::Layer< T >::setInputData (
            const NetNode< T > & in_data ) [inline]
```

Sets the input data to a layer

8.8.3.24 setInputDelta()

```
template<class T>
void mlearn::Layer< T >::setInputDelta (
            const NetNode< T > & in_delta )  [inline]
```

Sets the input delta to a layer

8.8.3.25 setOutputData()

```
template<class T>
void mlearn::Layer< T >::setOutputData (
            const NetNode< T > & out_data )  [inline]
```

Sets the output data from a layer

8.8.3.26 setOutputDelta()

```
template<class T>
void mlearn::Layer< T >::setOutputDelta (
            const NetNode< T > & out_delta )  [inline]
```

Sets the output delta from a layer

8.8.3.27 setWeight()

```
template<class T>
void mlearn::Layer< T >::setWeight (
            const mublas::matrix< T > & in_weight )  [inline]
```

Sets the weight matrix of a layer

8.8.3.28 updateParams() [1/2]

```
template<class T>
Layer<T>& mlearn::Layer< T >::updateParams (
            double rate,
            uint32_t batch_size,
            double lambda,
            std::string reg,
            double beta )
```

Overloaded function. After a single forward and backward pass, the parameters (weight and bias) of each layer are updated.

Parameters

rate	Learning rate
batch_size	Batch size used in training
lambda	Regularization parameter
reg	Type of regularization (L1, L2 or None)
beta	Momentum term/parameter

Returns

Reference to self

8.8.3.29 updateParams() [2/2]

```
template<class T>
Layer<T>& mlearn::Layer< T >::updateParams (
             double rate,
             uint32_t batch_size,
             double lambda,
             std::string reg,
             double beta,
             bool change_rate,
             std::string  = "adagrad" )
```

Overloaded function used for parameter updates of Adagrad/RMSProp.

Parameters

rate	Learning rate
batch_size	Batch size used in training
lambda	Regularization parameter
reg	Type of regularization (L1, L2 or None)
beta	Momentum term/parameter
change_rate	Determines if learning rates should be changed

Returns

Reference to self

8.8.4 Friends And Related Function Documentation

8.8.4.1 boost::serialization::access

```
template<class T>
friend class boost::serialization::access  [friend]
```

Responsible for saving/serialization of members

8.8.5 Member Data Documentation

8.8.5.1 act_function

```
template<class T>
Activation<T> mlearn::Layer< T >::act_function [protected]
```

Activation function of layer

8.8.5.2 activation

```
template<class T>
NetNode<T> mlearn::Layer< T >::activation [protected]
```

Contains activations

8.8.5.3 bias

```
template<class T>
NetNode<T> mlearn::Layer< T >::bias [protected]
```

Bias of layer

8.8.5.4 delta_b

```
template<class T>
NetNode<T> mlearn::Layer< T >::delta_b [protected]
```

Delta of bias

8.8.5.5 delta_w

```
template<class T>
mublas::matrix<T> mlearn::Layer< T >::delta_w [protected]
```

Delta of weight matrix

8.8.5.6 input_data

```
template<class T>
NetNode<T> mlearn::Layer< T >::input_data [protected]
```

Input data of layer

8.8.5.7 input_delta

```
template<class T>
NetNode<T> mlearn::Layer< T >::input_delta [protected]
```

Input delta of layer

8.8.5.8 input_dim

```
template<class T>
uint64_t mlearn::Layer< T >::input_dim [protected]
```

Input dimension of layer

8.8.5.9 momentum_w

```
template<class T>
mublas::matrix<T> mlearn::Layer< T >::momentum_w [protected]
```

Weight matrix used for momentum

8.8.5.10 next

```
template<class T>
Layer* mlearn::Layer< T >::next [protected]
```

Pointer to next layer

8.8.5.11 output_data

```
template<class T>
NetNode<T> mlearn::Layer< T >::output_data [protected]
```

Output data of layer

8.8.5.12 output_delta

```
template<class T>
NetNode<T> mlearn::Layer< T >::output_delta [protected]
```

Output delta of layer

8.8.5.13 output_dim

```
template<class T>
uint64_t mlearn::Layer< T >::output_dim [protected]
```

Output dimension of layer

8.8.5.14 previous

```
template<class T>
Layer* mlearn::Layer< T >::previous [protected]
```

Pointer to previous layer

8.8.5.15 regularize_w

```
template<class T>
mublas::matrix<T> mlearn::Layer< T >::regularize_w  [protected]
```

Weight matrix used for regularization

8.8.5.16 sq_delta_w

```
template<class T>
mublas::matrix<T> mlearn::Layer< T >::sq_delta_w  [protected]
```

Accumulates previous deltas. Used in Adagrad and RMSProp

8.8.5.17 type

```
template<class T>
std::string mlearn::Layer< T >::type  [protected]
```

Layer type (hidden_layer or output_layer)

8.8.5.18 weight

```
template<class T>
mublas::matrix<T> mlearn::Layer< T >::weight  [protected]
```

Weight matrix of layer

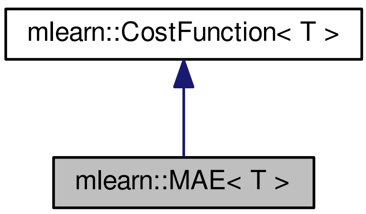
The documentation for this class was generated from the following file:

- layer.h

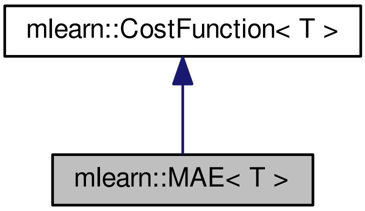
8.9 mlearn::MAE< T > Class Template Reference

```
#include <libutil.h>
```

Inheritance diagram for mlearn::MAE< T >:



Collaboration diagram for mlearn::MAE< T >:



Public Member Functions

- MAE ()
- double cost (const NetNode< T > &prediction, const NetNode< T > &label)
- double accuracy (const NetNode< T > &prediction, const NetNode< T > &label)
- NetNode< T > & costDerivative (const NetNode< T > &prediction, const NetNode< T > &label, NetNode< T > &result)

Additional Inherited Members

8.9.1 Detailed Description

template<class T>
class mlearn::MAE< T >

The MAE(mean absolute error) derives from the class CostFunction. It implements the "cost", "accuracy" and "costDerivative" functions. Used for regression problems.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 MAE()

```
template<class T >
mlearn::MAE< T >::MAE ( )  [inline]
```

Default constructor

8.9.3 Member Function Documentation

8.9.3.1 accuracy()

```
template<class T >
double mlearn::MAE< T >::accuracy (
            const NetNode< T > & prediction,
            const NetNode< T > & label )  [inline], [virtual]
```

Computes the absolute error between prediction and label. Calls the cost function.

Parameters

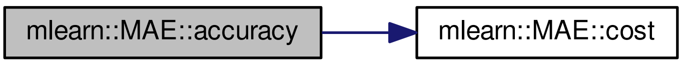
prediction	Hypotheses
label	Reference

Returns

The absolute difference between the prediction and label

Implements mlearn::CostFunction< T >.

Here is the call graph for this function:



8.9.3.2 cost()

```
template<class T >
double mlearn::MAE< T >::cost (
            const NetNode< T > & prediction,
            const NetNode< T > & label )  [virtual]
```

Computes the absolute error between the prediction and label.

Parameters

prediction	Hypotheses
label	Reference

Returns

The absolute difference between the prediction and label

Implements mlearn::CostFunction< T >.

8.9.3.3 costDerivative()

```
template<class T >
NetNode<T>& mlearn::MAE< T >::costDerivative (
            const NetNode< T > & prediction,
            const NetNode< T > & label,
            NetNode< T > & result ) [virtual]
```

Computes the derivative of MAE.

Parameters

prediction	Hypotheses
label	Reference
result	Result

Returns

Derivative of MAE

Implements mlearn::CostFunction< T >.

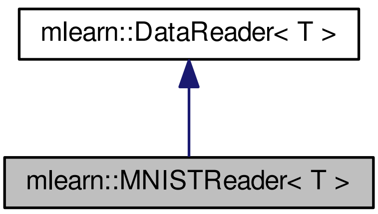
The documentation for this class was generated from the following file:

- libutil.h

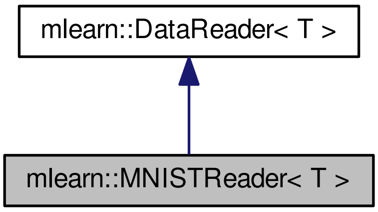
8.10 mlearn::MNISTReader< T > Class Template Reference

```
#include <data_reader.h>
```

Inheritance diagram for mlearn::MNISTReader< T >:



Collaboration diagram for mlearn::MNISTReader< T >:



Public Member Functions

- MNISTReader ()
- MNISTReader (const std::string file_name, char sep, bool header)
- MNISTReader (const std::string file_name, bool one_hot)
- MNISTReader (const MNISTReader< T > &argv)
- MNISTReader< T > & operator= (const MNISTReader< T > &argv)
- MNISTReader< T > & read ()
- virtual ~MNISTReader ()

Additional Inherited Members

8.10.1 Detailed Description

template<class T>
class mlearn::MNISTReader< T >

The MNISTReader class extends the DataReader class. It is responsible for reading the MNIST dataset into feature/label. The MNIST database of handwritten digits by Yann Lecun, Corinna Cortes http://yann.lecun.com/exdb/mnist/. https://pjreddie.com/projects/mnist-in-csv/.

```
//  Creates an MNISTReader object mnist and read a header-less file
//  "mnist_sample.csv". Entries are delimited by a comma
MNISTReader<double> mnist("data/mnist_sample.csv", ',',  false);
//  Calls the read method
mnist.read();
//  Creates a vector of integers and shuffles it.
std::vector<int> indices;
mnist.shuffleIndex(indices);
//  Creates an MNISTReader object test
MNISTReader<double> test
//  Splits "mnist" into train/validation set
//  10% of the original data is copied to test
mnist.trainTestSplit(test, 0.1);
```

8.10.2 Constructor & Destructor Documentation

8.10.2.1 MNISTReader() [1/4]

```
template<class T>
mlearn::MNISTReader< T >::MNISTReader ( ) [inline]
```

Default constructor

8.10.2.2 MNISTReader() [2/4]

```
template<class T>
mlearn::MNISTReader< T >::MNISTReader (
            const std::string file_name,
            char sep,
            bool header ) [inline]
```

Overloaded constructor with 3 arguments. The number of features is $784(28 * 28)$ and the label dimension is 10 ("one_hot" set to true).

8.10.2.3 MNISTReader() [3/4]

```
template<class T>
mlearn::MNISTReader< T >::MNISTReader (
            const std::string file_name,
            bool one_hot ) [inline]
```

Overloaded constructor with 2 arguments

8.10.2.4 MNISTReader() [4/4]

```
template<class T>
mlearn::MNISTReader< T >::MNISTReader (
            const MNISTReader< T > & argv )
```

Copy constructor.

Parameters

argv	MNISTReader object to be copied.

8.10.2.5 ∼MNISTReader()

```
template<class T>
virtual mlearn::MNISTReader< T >::~MNISTReader ( ) [inline], [virtual]
```

Virtual destructor

8.10.3 Member Function Documentation

8.10.3.1 operator=()

```
template<class T>
MNISTReader<T>& mlearn::MNISTReader< T >::operator= (
            const MNISTReader< T > & argv )
```

Overloaded assignment operator.

Parameters

argv	Reference to the second operand

Returns

Reference to self

8.10.3.2 read()

```
template<class T>
MNISTReader<T>& mlearn::MNISTReader< T >::read ( )  [virtual]
```

Reads a text file containing features and labels.

Returns

Reference to self.

Implements mlearn::DataReader< T >.

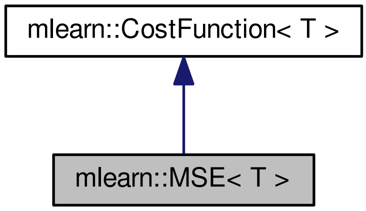
The documentation for this class was generated from the following file:

- data_reader.h

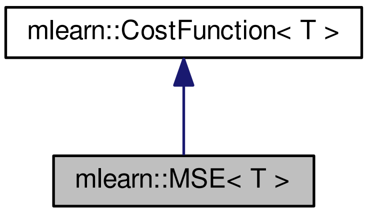
8.11 mlearn::MSE< T > Class Template Reference

```
#include <libutil.h>
```

Inheritance diagram for mlearn::MSE< T >:



Collaboration diagram for mlearn::MSE< T >:



Public Member Functions

- MSE ()
- double cost (const NetNode< T > &prediction, const NetNode< T > &label)
- double accuracy (const NetNode< T > &prediction, const NetNode< T > &label)
- NetNode< T > & costDerivative (const NetNode< T > &prediction, const NetNode< T > &label, NetNode< T > &result)

Additional Inherited Members

8.11.1 Detailed Description

template<class T>
class mlearn::MSE< T >

The MSE (mean squared error) derives from the class CostFunction. It implements "cost" "accuracy" and "cost↩ Derivative" functions. Used for regression problems.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 MSE()

```
template<class T >
mlearn::MSE< T >::MSE ( ) [inline]
```

Default constructor

8.11.3 Member Function Documentation

8.11.3.1 accuracy()

```
template<class T >
double mlearn::MSE< T >::accuracy (
            const NetNode< T > & prediction,
            const NetNode< T > & label ) [inline], [virtual]
```

Computes the error between the prediction and label. Calls the cost function.

Parameters

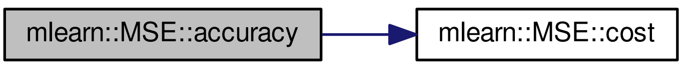
prediction	Hypotheses
label	Reference

Returns

The square of the difference between the prediction and label

Implements mlearn::CostFunction< T >.

Here is the call graph for this function:



8.11.3.2 cost()

```
template<class T >
double mlearn::MSE< T >::cost (
            const NetNode< T > & prediction,
            const NetNode< T > & label ) [virtual]
```

Computes the error between the prediction and label. This is the square of the difference between the prediction and label.

Parameters

prediction	Hypotheses
label	Reference

Returns

The square of the difference between the prediction and label

Implements mlearn::CostFunction< T >.

8.11.3.3 costDerivative()

```
template<class T >
NetNode<T>& mlearn::MSE< T >::costDerivative (
            const NetNode< T > & prediction,
            const NetNode< T > & label,
            NetNode< T > & result ) [virtual]
```

Computes the derivative of MSE. This is the difference between the prediction and label.

Parameters

prediction	Hypotheses
label	Reference
result	Result

Returns

The difference between the prediction and label

Implements mlearn::CostFunction< T >.

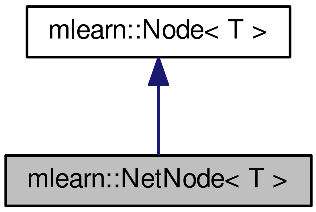
The documentation for this class was generated from the following file:

- libutil.h

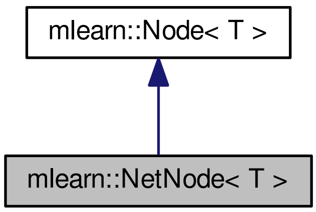
8.12 mlearn::NetNode< T > Class Template Reference

```
#include <libutil.h>
```

Inheritance diagram for mlearn::NetNode< T >:



Collaboration diagram for mlearn::NetNode< T >:



Public Member Functions

- NetNode ()
- NetNode (const mublas::vector< T > &in_data)
- NetNode (uint64_t data_size)
- NetNode (const NetNode< T > &in_node)
- NetNode (const Node< T > &in_node)
- NetNode & operator= (const NetNode &)
- NetNode< T > & sigmoid ()
- NetNode< T > sigmoidPrime ()
- NetNode< T > & hyperTan ()
- NetNode< T > & hyperTanPrime ()
- NetNode< T > & ReLU ()
- NetNode< T > & ReLUPrime (double alpha)
- NetNode< T > & softmax ()
- mublas::matrix< T > & softmaxPrime (mublas::matrix< T > &argv)

- NetNode< T > & ELU (double alpha)
- NetNode< T > & ELUPrime (double alpha)
- NetNode< T > & identity ()
- NetNode< T > & identityPrime ()
- NetNode< T > & log_e ()
- virtual ~ NetNode ()

Additional Inherited Members

8.12.1 Detailed Description

template<class T>
class mlearn::NetNode< T >

The NetNode class is a class derived from the Node class. The NetNode class extends the Node class and inherits "data" and 'data_size' members of the Node class. It adds functions specific to neural networks.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 NetNode() [1/5]

```
template<class T>
mlearn::NetNode< T >::NetNode ( )  [inline]
```

The default constructor

8.12.2.2 NetNode() [2/5]

```
template<class T>
mlearn::NetNode< T >::NetNode (
            const mublas::vector< T > & in_data )  [inline]
```

Overloaded constructor that takes one argument: in_data.

Parameters

in_data	A vector of data used to initialize the "data" member.

8.12.2.3 NetNode() [3/5]

```
template<class T>
mlearn::NetNode< T >::NetNode (
            uint64_t data_size )  [inline]
```

Calls the Node(uint64_t data_size) constructor

8.12.2.4 NetNode() [4/5]

```
template<class T>
mlearn::NetNode< T >::NetNode (
            const NetNode< T > & in_node )  [inline]
```

The copy constructor

8.12.2.5 NetNode() [5/5]

```
template<class T>
mlearn::NetNode< T >::NetNode (
            const Node< T > & in_node )  [inline]
```

Calls the Node copy constructor

8.12.2.6 ∼ NetNode()

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

A virtual destructor

8.12.3 Member Function Documentation

8.12.3.1 ELU()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::ELU (
            double alpha )
```

Computes exponential linear unit (ELU) activation.

Parameters

alpha	A parameter used in ELU

Returns

A reference to self (with ELU of "data" values)

8.12.3.2 ELUPrime()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::ELUPrime (
            double alpha )
```

Computes the derivative of ELU.

Parameters

alpha	A parameter used in ELU

Returns

A reference to self

8.12.3.3 hyperTan()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::hyperTan ( )
```

Computes the tanh (activation) of values of "data".

Precondition

None

Postcondition

Modifies "data" values.

Returns

A reference to self (with tanh of "data" values)

8.12.3.4 hyperTanPrime()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::hyperTanPrime ( )
```

Computes the derivative of tanh. The derivative of tanh is

$$1 - (f(x) * f(x)) \tag{8.1}$$

.

Returns

A reference to self (with tanh derivative of "data" values)

8.12.3.5 identity()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::identity ( )
```

The identity function. Does not modify "data"

8.12.3.6 identityPrime()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::identityPrime ( )
```

The derivative of identity function

8.12.3.7 log_e()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::log_e ( )
```

Computes log to base e of "data"

8.12.3.8 operator=()

```
template<class T>
NetNode& mlearn::NetNode< T >::operator= (
             const NetNode< T > &  )
```

Overloaded assignment operator

8.12.3.9 ReLU()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::ReLU ( )
```

Computes the Rectified linear Unit (ReLU) activation: max(0, x).

Precondition

None

Postcondition

Modifies "data" values.

Returns

A reference to self (with ReLU of "data" values)

8.12.3.10 ReLUPrime()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::ReLUPrime (
             double alpha )
```

The derivative of ReLU. This becomes a leaky ReLU if "alpha" is greater than 0.

Parameters

alpha	A parameter used for leaky ReLU

Returns

A reference to self (with the derivative of ReLU of "data" values)

8.12.3.11 sigmoid()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::sigmoid ( )
```

Computes the sigmoid (activation) of values of "data".

Precondition

None

Postcondition

Modifies "data" values.

Returns

A reference to self (with sigmoid of "data" values)

8.12.3.12 sigmoidPrime()

```
template<class T>
NetNode<T> mlearn::NetNode< T >::sigmoidPrime ( )
```

Computes the derivative of sigmoid. The derivative of sigmoid is

$$f(x) * (1 - f(x)) \tag{8.2}$$

.

Returns

A reference to self (with sigmoid derivative of "data" values)

8.12.3.13 softmax()

```
template<class T>
NetNode<T>& mlearn::NetNode< T >::softmax ( )
```

Computes softmax activation function; forces sum of probabilities to 1.

Precondition

None

Postcondition

Modifies "data" values.

Returns

A reference to self (with softmax of "data" values)

8.12.3.14 softmaxPrime()

```
template<class T>
mublas::matrix<T>& mlearn::NetNode< T >::softmaxPrime (
            mublas::matrix< T > & argv )
```

Computes derivative of softmax.

Parameters

argv	An empty 2D Jacobian matrix

Returns

A reference to argv

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

- libutil.h

8.13 mlearn::Network< T > Class Template Reference

```
#include <network.h>
```

Public Member Functions

- Network ()
- Network (CostFunction< T > *obj)
- Network< T > & addLayer (const Layer< T > &layer)
- Network< T > & connectLayers ()
- std::vector< Layer< T > > & getLayers ()
- const NetNode< T > & singleForward (const NetNode< T > &in_data)
- const NetNode< T > & singleBackward (const NetNode< T > &in_delta)
- Network< T > & updateNetwork (double learning_rate, uint32_t batch_size, double lambda, std::string reg, double beta)
- Network< T > & updateNetwork (double learning_rate, uint32_t batch_size, double lambda, std::string reg, double beta, bool change_rate, std::string id="adagrad")
- CostFunction< T > * getCostFunction ()
- void setUpdateRate (bool value)
- bool getUpdateRate ()
- void saveModel (std::string model_file)
- Network< T > & loadModel (std::string model_file)
- virtual ~Network ()

Protected Member Functions

- template< class Archive >
  void serialize (Archive &ar, const uint64_t version)

Protected Attributes

- std::vector< Layer< T > > layers
- CostFunction< T > * cost_function
- uint64_t num_layers
- bool update_rate {false}

Friends

- class boost::serialization::access

8.13.1 Detailed Description

template< class T>
class mlearn::Network< T >

The Network class is a classic multi-layer perceptron(MLP) consisting of sequences of layers: one or more hidden layers and an output layer. The network is trained using mini-batch SGD, Adagrad or RMSProp.

```
//  Creates 2 Activation objects: hidden and output.
Activation<double> hidden("sigmoid"), output("softmax");
//  A hidden layer with input and output dimensions 2.
//  Activation function used is sigmoid
Layer<double> hidden_layer(2, 2, "hidden", hidden);
//  An output layer with input and output dimensions 2 and 1 respectively.
//  Activation function used is softmax
Layer<double> output_layer(2, 1, "output", output);
//  Creates a Network object, cost function is cross entropy
Network<double> model(new CrossEntropy<double>);
//  Adds the hidden and output layers to the network
model.addLayer(hidden_layer);
model.addLayer(output_layer);
//  Connects the layers together
model.connectLayers();
```

8.13.2 Constructor & Destructor Documentation

8.13.2.1 Network() [1/2]

```
template<class T>
mlearn::Network< T >::Network ( )  [inline]
```

Default constructor, default cost function MSE

8.13.2.2 Network() [2/2]

```
template<class T>
mlearn::Network< T >::Network (
            CostFunction< T > * obj )  [inline]
```

Overloaded constructor with 1 argument

8.13.2.3 ∼Network()

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

Virtual destructor

8.13.3 Member Function Documentation

8.13.3.1 addLayer()

```
template<class T>
Network<T>& mlearn::Network< T >::addLayer (
            const Layer< T > & layer )
```

Adds a layer to the network.

Parameters

layer	The layer to be added

Returns

A reference to self

8.13.3.2 connectLayers()

```
template<class T>
Network<T>& mlearn::Network< T >::connectLayers ( )
```

Connects all layers in the network and returns a reference to self.

8.13.3.3 getCostFunction()

```
template<class T>
CostFunction<T>* mlearn::Network< T >::getCostFunction ( )  [inline]
```

Returns the cost function

8.13.3.4 getLayers()

```
template<class T>
std::vector<Layer<T> >& mlearn::Network< T >::getLayers ( )  [inline]
```

Returns the layers in a network

8.13.3.5 getUpdateRate()

```
template<class T>
bool mlearn::Network< T >::getUpdateRate ( )  [inline]
```

Gets the update_rate

8.13.3.6 loadModel()

```
template<class T>
Network<T>& mlearn::Network< T >::loadModel (
            std::string model_file )
```

Loads network/model from an archive file.

Parameters

model_file	Name of the model file

Returns

A reference to self

8.13.3.7 saveModel()

```
template<class T>
void mlearn::Network< T >::saveModel (
            std::string model_file )
```

Saves parameters of network/model.

Parameters

model_file	Name of the file to save model

8.13.3.8 serialize()

```
template<class T>
template<class Archive >
void mlearn::Network< T >::serialize (
            Archive & ar,
            const uint64_t version )  [inline], [protected]
```

8.13.3.9 setUpdateRate()

```
template<class T>
void mlearn::Network< T >::setUpdateRate (
            bool value )  [inline]
```

Sets the update_rate

8.13.3.10 singleBackward()

```
template<class T>
const NetNode<T>& mlearn::Network< T >::singleBackward (
            const NetNode< T > & in_delta )
```

Inputs delta through the network and propagates backward.

Parameters

in_delta	The input delta; a NetNode object

Returns

A reference to output delta

8.13.3.11 singleForward()

```
template<class T>
const NetNode<T>& mlearn::Network< T >::singleForward (
            const NetNode< T > & in_data )
```

Inputs a single data through the network and propagates forward.

Parameters

in_data	The input data; a NetNode object

Returns

A reference to output data

8.13.3.12 updateNetwork() [1/2]

```
template<class T>
Network<T>& mlearn::Network< T >::updateNetwork (
            double learning_rate,
            uint32_t batch_size,
            double lambda,
            std::string reg,
            double beta )
```

Overloaded function. Updates network parameters.

Parameters

learning_rate	Train learning rate
batch_size	Batch size used in training
lambda	Regularization parameter (between 0 and 1)
reg	Type of regularization (L1, L2 or None)
beta	A momentum term/parameter (between 0 and 1)

Returns

A reference to self

8.13.3.13 updateNetwork() [2/2]

```
template<class T>
Network<T>& mlearn::Network< T >::updateNetwork (
            double learning_rate,
```

```
uint32_t batch_size,
double lambda,
std::string reg,
double beta,
bool change_rate,
std::string id = "adagrad" )
```

Overloaded function. Updates network parameters. Applies to Adagrad/RMSProp

Parameters

learning_rate	Training learning rate
batch_size	Batch size used in training
lambda	Regularization parameter (between 0 and 1)
reg	Type of regularization (L1, L2 or None)
beta	A momentum term/parameter (between 0 and 1)
change_rate	Determines if learning rates should be changed
id	Id of the optimizer used: adagrad or rmsprop

Returns

A reference to self

8.13.4 Friends And Related Function Documentation

8.13.4.1 boost::serialization::access

```
template<class T>
friend class boost::serialization::access  [friend]
```

Responsible for saving/serialization of members

8.13.5 Member Data Documentation

8.13.5.1 cost_function

```
template<class T>
CostFunction<T>* mlearn::Network< T >::cost_function  [protected]
```

A pointer to CostFunction

8.13.5.2 layers

```
template<class T>
std::vector<Layer<T> > mlearn::Network< T >::layers  [protected]
```

A vector of Layer objects

8.13.5.3 num_layers

```
template<class T>
uint64_t mlearn::Network< T >::num_layers  [protected]
```

Number of layers in the network

8.13.5.4 update_rate

```
template<class T>
bool mlearn::Network< T >::update_rate {false}  [protected]
```

Used to decide if training rate should be updated/changed for Adagrad/RMSProp.

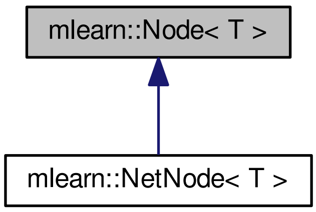
The documentation for this class was generated from the following file:

- network.h

8.14 mlearn::Node$<$ T $>$ Class Template Reference

```
#include <libutil.h>
```

Inheritance diagram for mlearn::Node$<$ T $>$:



Public Member Functions

- Node ()
- Node (const mublas::vector< T > &in_data)
- Node (uint64_t data_size)
- Node (const Node< T > &in_node)
- const mublas::vector< T > & getData () const
- Node< T > & setData (const mublas::vector< T > &in_data)
- uint64_t getDataSize () const
- Node< T > operator+ (const Node< T > &argv)
- Node< T > operator- (const Node< T > &argv)
- Node< T > operator* (const Node< T > &argv)
- Node< T > & operator= (const Node< T > &argv)
- Node< T > scalarMultiply (double rate)
- double sum () const
- double fabsSum () const
- const Node< T > & describeNode () const
- Node< T > & generateRandomData (uint64_t input_dim, uint64_t output_dim)
- Node< T > & compare (const Node< T > &argv, Node< T > &result) const
- virtual ~Node ()

Protected Member Functions

- template<class Archive >
  void serialize (Archive &ar, const uint64_t version)

Protected Attributes

- mublas::vector< T > data
- uint64_t data_size

Friends

- class boost::serialization::access

8.14.1 Detailed Description

template<class T>
class mlearn::Node< T >

The Node class is the fundamental data structure used. It contains 2 protected members: data and data_size. data is of type boost ublas vector and may hold features or labels. Different machine learning methods can extend the Node class, e.g. for multi-layer perceptron, NetNode extends the Node class.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Node() [1/4]

```
template<class T>
mlearn::Node< T >::Node ( )  [inline]
```

The default constructor

8.14.2.2 Node() [2/4]

```
template<class T>
mlearn::Node< T >::Node (
             const mublas::vector< T > & in_data )  [inline]
```

Overloaded constructor that takes one argument: in_data.

Parameters

in_data	A vector of data used to initialize the "data" member

8.14.2.3 Node() [3/4]

```
template<class T>
mlearn::Node< T >::Node (
             uint64_t data_size )  [inline]
```

Overloaded constructor that takes one argument: data_size. This initializes "data" member with a zero vector of size "data_size".

Parameters

data_size	The size/dimension of "data"

8.14.2.4 Node() [4/4]

```
template<class T>
mlearn::Node< T >::Node (
             const Node< T > & in_node )  [inline]
```

The copy constructor.

Parameters

in_node	The Node object to be copied

8.14.2.5 ∼Node()

```
template<class T>
virtual mlearn::Node< T >::~Node ( )  [inline], [virtual]
```

A virtual destructor. The Node class is a base class. Base pointers and references will often be used for derived classes. Making this virtual ensures the correct destructor is called.

8.14.3 Member Function Documentation

8.14.3.1 compare()

```
template<class T>
Node<T>& mlearn::Node< T >::compare (
            const Node< T > & argv,
            Node< T > & result ) const
```

Compares the values of "data" between self and argv. If it's less, returns 1; if it's greater, returns -1 and otherwise returns 0.

Parameters

argv	Node to be compared
result	Result Node

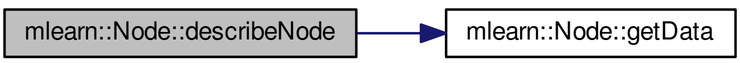
Returns

A reference to result Node

8.14.3.2 describeNode()

```
template<class T>
const Node<T>& mlearn::Node< T >::describeNode ( ) const  [inline]
```

Prints values of "data" Here is the call graph for this function:



8.14.3.3 fabsSum()

```
template<class T>
double mlearn::Node< T >::fabsSum ( ) const
```

Returns the sum of abs of "data".

8.14.3.4 generateRandomData()

```
template<class T>
Node<T>& mlearn::Node< T >::generateRandomData (
            uint64_t input_dim,
            uint64_t output_dim )
```

Generates "data" randomly between a range. Useful for constructing a Node object with randomly generated "data" value. The data follows a distribution with lower and upper bounds computed from the passed parameters.

Parameters

input_dim	Input size
output_dim	Output size

Returns

A reference to self

8.14.3.5 getData()

```
template<class T>
const mublas::vector<T>& mlearn::Node< T >::getData ( ) const [inline]
```

Accessor for the Node "data" member.

Precondition

None

Postcondition

Does not change the object

Returns

data A vector of type T

8.14.3.6 getDataSize()

```
template<class T>
uint64_t mlearn::Node< T >::getDataSize ( ) const  [inline]
```

Accessor for the Node "data_size" member.

Precondition

None

Postcondition

Does not change the object.

Returns

data_size The size/dimension of "data"

8.14.3.7 operator*()

```
template<class T>
Node<T> mlearn::Node< T >::operator* (
              const Node< T > & argv )
```

Overloaded multiplication operator.

Parameters

argv	A reference to the second operand

Returns

A new Node object

8.14.3.8 operator+()

```
template<class T>
Node<T> mlearn::Node< T >::operator+ (
              const Node< T > & argv )
```

Overloaded addition operator.

Parameters

argv	A reference to the second operand.

Returns

A new Node object.

8.14.3.9 operator-()

```
template<class T>
Node<T> mlearn::Node< T >::operator- (
            const Node< T > & argv )
```

Overloaded subtraction operator.

Parameters

argv	A reference to the second operand

Returns

A new Node object

8.14.3.10 operator=()

```
template<class T>
Node<T>& mlearn::Node< T >::operator= (
            const Node< T > & argv )
```

Overloaded assignment operator.

Parameters

argv	A reference to the second operand

Returns

A reference to self.

8.14.3.11 scalarMultiply()

```
template<class T>
Node<T> mlearn::Node< T >::scalarMultiply (
            double rate )
```

Implements scalar multiplication (constant $*$ data).

Parameters

rate	A scalar of type "double"

Returns

A new Node object with scaled "data"

8.14.3.12 serialize()

```
template<class T>
template<class Archive >
void mlearn::Node< T >::serialize (
            Archive & ar,
            const uint64_t version ) [inline], [protected]
```

8.14.3.13 setData()

```
template<class T>
Node<T>& mlearn::Node< T >::setData (
            const mublas::vector< T > & in_data ) [inline]
```

A Mutator for the Node data member.

Precondition

None

Postcondition

The "data" member variable of Node will be changed to the input value.

Parameters

in_data	A vector of input data of type T, copied to the "data" member

Returns

∗this A reference to the calling object

8.14.3.14 sum()

```
template<class T>
double mlearn::Node< T >::sum ( ) const
```

Returns the sum of "data" of the calling object (this).

8.14.4 Friends And Related Function Documentation

8.14.4.1 boost::serialization::access

```
template<class T>
friend class boost::serialization::access [friend]
```

The is responsible for saving/serialization of members

8.14.5 Member Data Documentation

8.14.5.1 data

```
template<class T>
mublas::vector<T> mlearn::Node< T >::data [protected]
```

This can hold features/labels

8.14.5.2 data_size

```
template<class T>
uint64_t mlearn::Node< T >::data_size [protected]
```

This is the size/dimension of data

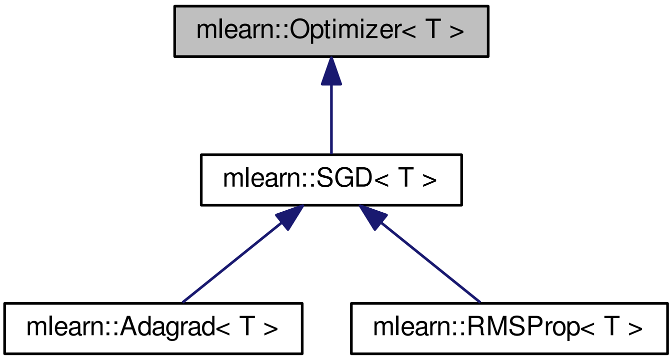
The documentation for this class was generated from the following file:

- libutil.h

8.15 mlearn::Optimizer< T > Class Template Reference

```
#include <optimizer.h>
```

Inheritance diagram for mlearn::Optimizer< T >:



Public Member Functions

- virtual Network< T > & train (Network< T > &model, std::string model_file, const DataReader< T > *train, const DataReader< T > *validation=nullptr, std::string id="sgd")
- virtual double predict (Network< T > &model, const DataReader< T > *test, std::string model_file)
- virtual Network< T > & update (Network< T > &model)
- virtual ∼Optimizer ()

8.15.1 Detailed Description

template<class T>
class mlearn::Optimizer< T >

The Optimizer class is the base class responsible for training algorithms. 3 optimizers are currently implemented, namely, SGD, Adagrad and RMSProp. The SGD optimizer is mini- batch stochastic gradient descent. The difference between Adagrad/ RMSProp and SGD is that the latter uses variable/per parameter learning rate.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 ∼Optimizer()

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

Virtual destructor

8.15.3 Member Function Documentation

8.15.3.1 predict()

```
template<class T >
virtual double mlearn::Optimizer< T >::predict (
            Network< T > & model,
            const DataReader< T > * test,
            std::string model_file )  [virtual]
```

Virtual function. This is responsible for prediction/test.

Parameters

model	Trained model object loaded from file
test	Pointer to test dataset
model_file	The name of file to save trained model

Returns

A test metric (accuracy, mse, mae)

Reimplemented in mlearn::RMSProp< T >, mlearn::Adagrad< T >, and mlearn::SGD< T >.

8.15.3.2 train()

```
template<class T >
virtual Network<T>& mlearn::Optimizer< T >::train (
            Network< T > & model,
            std::string model_file,
            const DataReader< T > * train,
            const DataReader< T > * validation = nullptr,
            std::string id = "sgd" )  [virtual]
```

Virtual function. This is responsible for model training.

Parameters

model	The Model object to train
model_file	The name of file to save trained model
train	Pointer to train dataset
validation	Pointer to validation dataset. Default is nullptr
id	The type of optimizer

Returns

A reference to the trained model

Reimplemented in mlearn::SGD< T >, mlearn::RMSProp< T >, and mlearn::Adagrad< T >.

8.15.3.3 update()

```
template<class T >
virtual Network<T>& mlearn::Optimizer< T >::update (
            Network< T > & model )  [virtual]
```

Virtual function. This is responsible for model update during training.

Parameters

model	The model object to update

Returns

A reference to updated model

Reimplemented in mlearn::RMSProp< T >, mlearn::Adagrad< T >, and mlearn::SGD< T >.

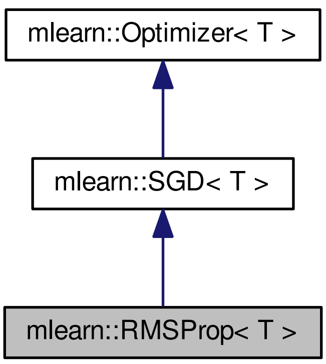
The documentation for this class was generated from the following file:

- optimizer.h

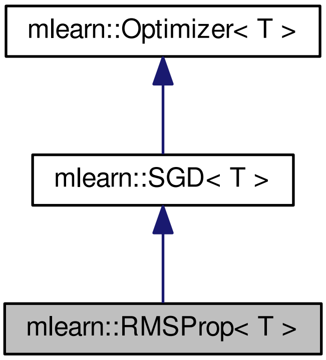
8.16 mlearn::RMSProp< T > Class Template Reference

```
#include <optimizer.h>
```

Inheritance diagram for mlearn::RMSProp< T >:



Collaboration diagram for mlearn::RMSProp< T >:



Public Member Functions

- RMSProp ()
- RMSProp (double learning_rate, uint32_t batch_size, uint32_t n_epochs, double lambda, std::string reg, double beta)
- RMSProp (double learning_rate, uint32_t batch_size, uint32_t n_epochs)
- RMSProp (double learning_rate, uint32_t batch_size, uint32_t n_epochs, double lambda, std::string reg)
- Network< T > & train (Network< T > &, std::string, const DataReader< T > *, const DataReader< T > *=nullptr, std::string="rmsprop")
- virtual Network< T > & update (Network< T > &)
- double predict (Network< T > &, const DataReader< T > *, std::string)
- virtual ~RMSProp ()

Additional Inherited Members

8.16.1 Detailed Description

template<class T>
class mlearn::RMSProp< T >

The RMSProp class extends the SGD class. It implements the root means square SGD. The only difference as compared to SGD is how the parameters are updated. RMSProp adapts the learning rate to each parameter.

Root mean square propagation (RMSProp) T Tieleman, and G E Hinton, Lecture 6.5 - rmsprop, COURSERA: Neural Networks for Machine Learning (2012)

8.16.2 Constructor & Destructor Documentation

8.16.2.1 RMSProp() [1/4]

```
template<class T >
mlearn::RMSProp< T >::RMSProp ( ) [inline]
```

Default constructor

8.16.2.2 RMSProp() [2/4]

```
template<class T >
mlearn::RMSProp< T >::RMSProp (
            double learning_rate,
            uint32_t batch_size,
            uint32_t n_epochs,
            double lambda,
            std::string reg,
            double beta ) [inline]
```

Overloaded constructor with 6 arguments

8.16.2.3 RMSProp() [3/4]

```
template<class T >
mlearn::RMSProp< T >::RMSProp (
            double learning_rate,
            uint32_t batch_size,
            uint32_t n_epochs ) [inline]
```

Overloaded constructor with 3 arguments

8.16.2.4 RMSProp() [4/4]

```
template<class T >
mlearn::RMSProp< T >::RMSProp (
            double learning_rate,
            uint32_t batch_size,
            uint32_t n_epochs,
            double lambda,
            std::string reg ) [inline]
```

Overloaded constructor with 5 arguments

8.16.2.5 ∼RMSProp()

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

Virtual destructor

8.16.3 Member Function Documentation

8.16.3.1 predict()

```
template<class T >
double mlearn::RMSProp< T >::predict (
            Network< T > & ,
            const DataReader< T > * ,
            std::string  )  [virtual]
```

Implements predict function

Reimplemented from mlearn::SGD< T >.

8.16.3.2 train()

```
template<class T >
Network<T>& mlearn::RMSProp< T >::train (
            Network< T > & ,
            std::string ,
            const DataReader< T > * ,
            const DataReader< T > *  = nullptr,
            std::string  = "rmsprop" )  [virtual]
```

Implements train function

Reimplemented from mlearn::SGD< T >.

8.16.3.3 update()

```
template<class T >
virtual Network<T>& mlearn::RMSProp< T >::update (
            Network< T > &  )  [virtual]
```

Implements update function

Reimplemented from mlearn::SGD< T >.

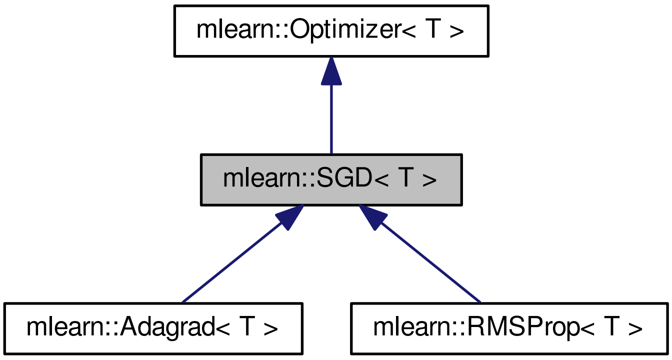
The documentation for this class was generated from the following file:

- optimizer.h

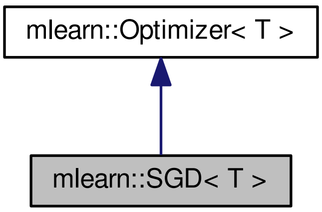
8.17 mlearn::SGD< T > Class Template Reference

```
#include <optimizer.h>
```

Inheritance diagram for mlearn::SGD< T >:



Collaboration diagram for mlearn::SGD< T >:



Public Member Functions

- SGD ()
- SGD (double learning_rate, uint32_t batch_size, uint32_t n_epochs, double lambda, std::string reg, double beta)
- SGD (double learning_rate, uint32_t batch_size, uint32_t n_epochs)
- SGD (double learning_rate, uint32_t batch_size, uint32_t n_epochs, double lambda, std::string reg)
- Network< T > & train (Network< T > &, std::string, const DataReader< T > ∗, const DataReader< T > ∗=nullptr, std::string="sgd")
- Network< T > & update (Network< T > &)
- double predict (Network< T > &, const DataReader< T > ∗, std::string)
- virtual ∼SGD ()

Protected Attributes

- double learning_rate {0.1}
- uint32_t batch_size {10}
- uint32_t num_epochs {20}
- double lambda {0.0}
- std::string reg {"None"}
- double beta {0.0}

8.17.1 Detailed Description

template<class T>
class mlearn::SGD< T >

The SGD class extends the Optimizer class. It implements the classical mini-batch stochastic gradient descent.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 SGD() [1/4]

```
template<class T >
mlearn::SGD< T >::SGD ( )  [inline]
```

Default constructor

8.17.2.2 SGD() [2/4]

```
template<class T >
mlearn::SGD< T >::SGD (
             double learning_rate,
             uint32_t batch_size,
             uint32_t n_epochs,
             double lambda,
             std::string reg,
             double beta )  [inline]
```

Overloaded constructor with 6 arguments

8.17.2.3 SGD() [3/4]

```
template<class T >
mlearn::SGD< T >::SGD (
             double learning_rate,
             uint32_t batch_size,
             uint32_t n_epochs )  [inline]
```

Overloaded constructor with 3 arguments

8.17.2.4 SGD() [4/4]

```
template<class T >
mlearn::SGD< T >::SGD (
            double learning_rate,
            uint32_t batch_size,
            uint32_t n_epochs,
            double lambda,
            std::string reg ) [inline]
```

Overloaded constructor with 5 arguments

8.17.2.5 ∼SGD()

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

Virtual destructor

8.17.3 Member Function Documentation

8.17.3.1 predict()

```
template<class T >
double mlearn::SGD< T >::predict (
            Network< T > & ,
            const DataReader< T > * ,
            std::string ) [virtual]
```

Implements predict function

Reimplemented from mlearn::Optimizer< T >.

Reimplemented in mlearn::RMSProp< T >, and mlearn::Adagrad< T >.

8.17.3.2 train()

```
template<class T >
Network<T>& mlearn::SGD< T >::train (
            Network< T > & ,
            std::string ,
            const DataReader< T > * ,
            const DataReader< T > * = nullptr,
            std::string = "sgd" ) [virtual]
```

Implements train function

Reimplemented from mlearn::Optimizer< T >.

Reimplemented in mlearn::RMSProp< T >, and mlearn::Adagrad< T >.

8.17.3.3 update()

```
template<class T >
Network<T>& mlearn::SGD< T >::update (
              Network< T > &  )  [virtual]
```

Implements update function

Reimplemented from mlearn::Optimizer< T >.

Reimplemented in mlearn::RMSProp< T >, and mlearn::Adagrad< T >.

8.17.4 Member Data Documentation

8.17.4.1 batch_size

```
template<class T >
uint32_t mlearn::SGD< T >::batch_size {10}  [protected]
```

Train batch size

8.17.4.2 beta

```
template<class T >
double mlearn::SGD< T >::beta {0.0}  [protected]
```

Momentum term/parameter

8.17.4.3 lambda

```
template<class T >
double mlearn::SGD< T >::lambda {0.0}  [protected]
```

Regularization parameter

8.17.4.4 learning_rate

```
template<class T >
double mlearn::SGD< T >::learning_rate {0.1}  [protected]
```

Train learning rate

8.17.4.5 num_epochs

```
template<class T >
uint32_t mlearn::SGD< T >::num_epochs {20}  [protected]
```

Number of train epochs

8.17.4.6 reg

```
template<class T >
std::string mlearn::SGD< T >::reg {"None"}  [protected]
```

Type of Regularization (L1 or L2)

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

- optimizer.h

Chapter 9

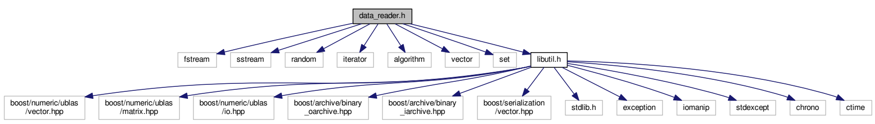
File Documentation

9.1 basics.txt File Reference

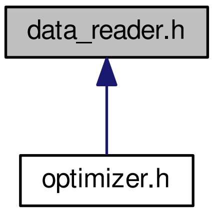
9.2 data_reader.h File Reference

```
#include <fstream>
#include <sstream>
#include <random>
#include <iterator>
#include <algorithm>
#include <vector>
#include <set>
#include "libutil.h"
```

Include dependency graph for data_reader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mlearn::DataReader< T >
- class mlearn::MNISTReader< T >
- class mlearn::IrisReader< T >
- class mlearn::GenericReader< T >

Namespaces

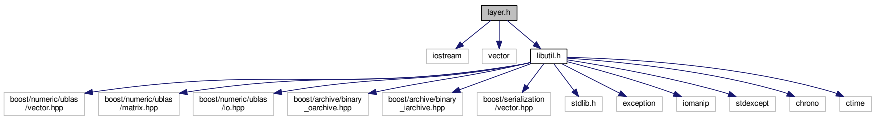
- mlearn

9.3 design.txt File Reference

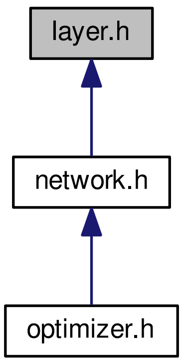
9.4 layer.h File Reference

```
#include <iostream>
#include <vector>
#include "libutil.h"
```

Include dependency graph for layer.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mlearn::Layer< T >

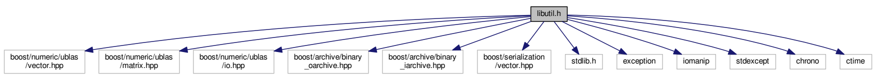
Namespaces

- mlearn

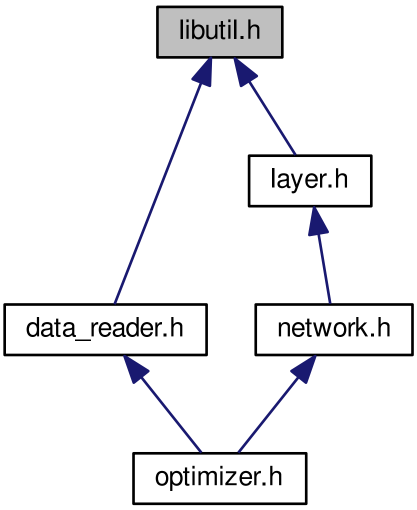
9.5 libutil.h File Reference

```
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/archive/binary_iarchive.hpp>
#include <boost/serialization/vector.hpp>
#include <stdlib.h>
#include <exception>
#include <iomanip>
#include <stdexcept>
#include <chrono>
#include <ctime>
```

Include dependency graph for libutil.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mlearn::Node< T >
- class mlearn::NetNode< T >
- class mlearn::Activation< T >
- class mlearn::CostFunction< T >
- class mlearn::MSE< T >
- class mlearn::CrossEntropy< T >
- class mlearn::MAE< T >

Namespaces

- mlearn

Macros

- #define EPSILON 1e-2
- #define ADAGRAD_EPSILON 1e-8
- #define MAX_MNIST_VALUE 255
- #define MAX_IRIS_VALUE 7.9

Functions

- template<class T >
  std::vector< Node< T > * > & mlearn::destroy (std::vector< Node< T > * > &argv)
- template<class T >
  double mlearn::mean (mublas::matrix< T > argv)

9.5.1 Detailed Description

This contains declarations of various classes in libutil.cpp, and resides in the mlearn namespace.

Author

Kalu U. Ogbureke

Version

1.0.0

Date

13.02.2019

9.5.2 Macro Definition Documentation

9.5.2.1 ADAGRAD_EPSILON

```
#define ADAGRAD_EPSILON 1e-8
```

9.5.2.2 EPSILON

```
#define EPSILON 1e-2
```

9.5.2.3 MAX_IRIS_VALUE

```
#define MAX_IRIS_VALUE 7.9
```

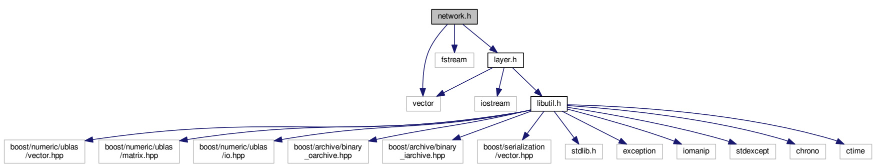
9.5.2.4 MAX_MNIST_VALUE

```
#define MAX_MNIST_VALUE 255
```

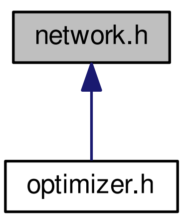
9.6 network.h File Reference

```
#include <vector>
#include <fstream>
#include "layer.h"
```

Include dependency graph for network.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mlearn::Network< T >

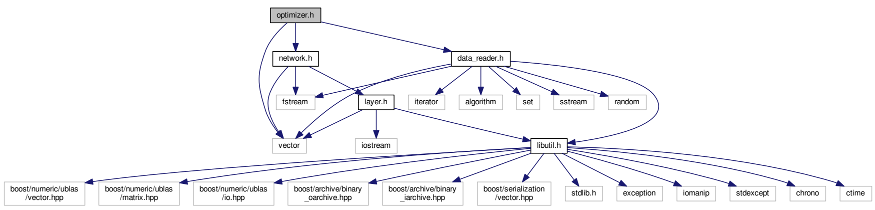
Namespaces

- mlearn

9.7 optimizer.h File Reference

```
#include <vector>
#include "network.h"
#include "data_reader.h"
```

Include dependency graph for optimizer.h:



Classes

- class mlearn::Optimizer< T >
- class mlearn::SGD< T >
- class mlearn::Adagrad< T >
- class mlearn::RMSProp< T >

Namespaces

- mlearn

Functions

- template<class T >
  Network< T > & mlearn::SGDHelper (Network< T > &model, uint32_t batch_size, uint32_t num_epochs, Optimizer< T > ∗opt, const DataReader< T > ∗train, const DataReader< T > ∗validation=nullptr, std::string id="sgd")

Bibliography

[1] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995. 2, 4, 5

[2] DARPA. Neural network study. *Lexington, MA: MIT Lincoln Laboratory*, 1988. 2

[3] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. 6

[4] Byrne et al. Role of nonlinear dynamical properties of a modelled bursting neuron in information processing and storage. *Netherlands Journal of Zoology*, 44:339–356, 1994. 1

[5] M. Hagan, H. Demuth, and M. Beale. *Neural Network Design*. PWS Publishing, 1996. 3

[6] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. volume 79, pages 2554–2558, 1982. 2

[7] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943. 2

[8] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, 1969. 3

[9] M. A. Nielsen. Neural networks and deep learning (http://neuralnetworksanddeeplearning.com/), 2018. 4

[10] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. 3

[11] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Micro-structure of Cognition*. Cambridge, MA: MIT Press, 1986. 4

[12] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. 6

[13] B. Widrow and M. E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, 4:96–104, 1960. 3

Index