

CNet

Generated by Doxygen 1.8.13

Contents

1	Todo List	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	AFActivationFunction< T, N, M > Class Template Reference	9
5.1.1	Member Function Documentation	9
5.1.1.1	derivative()	9
5.1.1.2	evaluate()	10
5.2	AFLossFunction< T, N, M > Class Template Reference	10
5.2.1	Member Function Documentation	10
5.2.1.1	derivative()	10
5.2.1.2	evaluate()	11
5.3	AFMatrix< T, ROWS, COLS > Class Template Reference	11
5.3.1	Constructor & Destructor Documentation	12
5.3.1.1	AFMatrix() [1/3]	12
5.3.1.2	AFMatrix() [2/3]	12
5.3.1.3	AFMatrix() [3/3]	12

5.3.1.4	<code>~AFMatrix()</code>	13
5.3.2	Member Function Documentation	13
5.3.2.1	<code>add()</code>	13
5.3.2.2	<code>copyValues()</code> [1/4]	13
5.3.2.3	<code>copyValues()</code> [2/4]	14
5.3.2.4	<code>copyValues()</code> [3/4]	14
5.3.2.5	<code>copyValues()</code> [4/4]	14
5.3.2.6	<code>equals()</code>	15
5.3.2.7	<code>getCol()</code>	15
5.3.2.8	<code>getIndex()</code>	15
5.3.2.9	<code>getRow()</code>	16
5.3.2.10	<code>getValue()</code>	16
5.3.2.11	<code>innerProduct()</code> [1/4]	16
5.3.2.12	<code>innerProduct()</code> [2/4]	18
5.3.2.13	<code>innerProduct()</code> [3/4]	18
5.3.2.14	<code>innerProduct()</code> [4/4]	18
5.3.2.15	<code>scale()</code>	19
5.3.2.16	<code>setValue()</code>	19
5.3.2.17	<code>subtract()</code>	20
5.3.2.18	<code>toArray()</code>	20
5.3.2.19	<code>transpose()</code> [1/2]	20
5.3.2.20	<code>transpose()</code> [2/2]	20
5.3.3	Member Data Documentation	21
5.3.3.1	<code>numCols</code>	21
5.3.3.2	<code>numRows</code>	21
5.3.3.3	<code>vals</code>	21
5.4	AFSquareLossFunction< T, N, M > Class Template Reference	22
5.4.1	Member Function Documentation	22
5.4.1.1	<code>derivative()</code>	22
5.4.1.2	<code>evaluate()</code>	22

5.5	IdentityFunction< T, N, M > Class Template Reference	23
5.5.1	Member Function Documentation	23
5.5.1.1	derivative()	23
5.5.1.2	evaluate()	23
5.6	Layer< LEN_IN, LEN_OUT > Class Template Reference	23
5.6.1	Detailed Description	24
5.6.2	Constructor & Destructor Documentation	25
5.6.2.1	Layer()	25
5.6.2.2	~Layer()	25
5.6.3	Member Function Documentation	25
5.6.3.1	backpropagate() [1/2]	25
5.6.3.2	backpropagate() [2/2]	26
5.6.3.3	backpropagateBase() [1/2]	26
5.6.3.4	backpropagateBase() [2/2]	27
5.6.3.5	forwardPass() [1/2]	27
5.6.3.6	forwardPass() [2/2]	27
5.6.3.7	randomizeWeights()	28
5.6.3.8	updateWeights()	28
5.6.4	Member Data Documentation	28
5.6.4.1	activationFunction	28
5.6.4.2	deltas	28
5.6.4.3	inputVals	28
5.6.4.4	lenIn	29
5.6.4.5	lenOut	29
5.6.4.6	outputVals	29
5.6.4.7	sums	29
5.6.4.8	weightGradient	29
5.6.4.9	weights	30
5.7	Net< T > Class Template Reference	30
5.8	ReLU< T, N, M > Class Template Reference	30
5.8.1	Member Function Documentation	30
5.8.1.1	derivative()	30
5.8.1.2	evaluate()	31

6 File Documentation	33
6.1 C:/Users/Aryan/CLionProjects/CNet/src/AFFunctions.h File Reference	33
6.2 C:/Users/Aryan/CLionProjects/CNet/src/AFMatrix.cpp File Reference	33
6.3 C:/Users/Aryan/CLionProjects/CNet/src/AFMatrix.h File Reference	33
6.3.1 Macro Definition Documentation	34
6.3.1.1 ACCEPTABLE_DOUBLE_DIFF	34
6.3.2 Function Documentation	34
6.3.2.1 doubleVectorEqual()	34
6.3.2.2 vectorInnerProduct()	34
6.3.2.3 vectorInnerProductBounded()	35
6.4 C:/Users/Aryan/CLionProjects/CNet/src/Layer.cpp File Reference	35
6.5 C:/Users/Aryan/CLionProjects/CNet/src/Layer.h File Reference	35
6.6 C:/Users/Aryan/CLionProjects/CNet/src/main.cpp File Reference	36
6.6.1 Function Documentation	36
6.6.1.1 main()	36
6.7 C:/Users/Aryan/CLionProjects/CNet/src/Net.cpp File Reference	36
6.8 C:/Users/Aryan/CLionProjects/CNet/src/Net.h File Reference	36
Index	37

Chapter 1

Todo List

Member `AFMatrix< T, ROWS, COLS >::AFMatrix` (`AFMatrix< T, ROWS, COLS > *copyFrom`)

Make this efficient and non-copying

Member `AFMatrix< T, ROWS, COLS >::AFMatrix` (`array< T, ROWS *COLS > *copyFromArray`)

Make this efficient and non-copying

Member `AFMatrix< T, ROWS, COLS >::copyValues` (`AFMatrix< T, ROWS, COLS > *dst, array< T, ROWS *COLS > *src`)

Clarify how things work if matrices can be row-major or column-major.

Member `AFMatrix< T, ROWS, COLS >::vals`

Is vals dynamically allocated or what?? | a b |

| c d | = [a b c d e f]

| e f |

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

AFActivationFunction< T, N, M >	9
IdentityFunction< T, N, M >	23
ReLU< T, N, M >	30
AFActivationFunction< double, LEN_IN, LEN_OUT >	9
AFLossFunction< T, N, M >	10
AFSquareLossFunction< T, N, M >	22
AFMatrix< T, ROWS, COLS >	11
AFMatrix< double, LEN_OUT, LEN_IN >	11
Layer< LEN_IN, LEN_OUT >	23
Net< T >	30

Chapter 3

Class Index

3.1 Class List

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

AFActivationFunction< T, N, M >	9
AFLossFunction< T, N, M >	10
AFMatrix< T, ROWS, COLS >	11
AFSquareLossFunction< T, N, M >	22
IdentityFunction< T, N, M >	23
Layer< LEN_IN, LEN_OUT >	23
Net< T >	30
ReLU< T, N, M >	30

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

C:/Users/Aryan/CLionProjects/CNet/src/ AFFunctions.h	33
C:/Users/Aryan/CLionProjects/CNet/src/ AFMatrix.cpp	33
C:/Users/Aryan/CLionProjects/CNet/src/ AFMatrix.h	33
C:/Users/Aryan/CLionProjects/CNet/src/ Layer.cpp	35
C:/Users/Aryan/CLionProjects/CNet/src/ Layer.h	35
C:/Users/Aryan/CLionProjects/CNet/src/ main.cpp	36
C:/Users/Aryan/CLionProjects/CNet/src/ Net.cpp	36
C:/Users/Aryan/CLionProjects/CNet/src/ Net.h	36

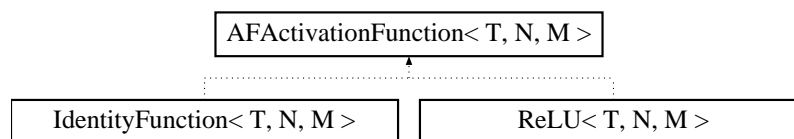
Chapter 5

Class Documentation

5.1 AFActivationFunction< T, N, M > Class Template Reference

```
#include <AFFunctions.h>
```

Inheritance diagram for AFActivationFunction< T, N, M >:



Public Member Functions

- void [evaluate](#) (array< T, N > *input, array< T, N > *output)
- void [derivative](#) (array< T, M > *input, array< T, M > *output)

5.1.1 Member Function Documentation

5.1.1.1 derivative()

```
template<typename T, size_t N, size_t M>
void AFActivationFunction< T, N, M >::derivative (
    array< T, M > * input,
    array< T, M > * output ) [inline]
```

5.1.1.2 evaluate()

```
template<typename T, size_t N, size_t M>
void AFActivationFunction< T, N, M >::evaluate (
    array< T, N > * input,
    array< T, N > * output ) [inline]
```

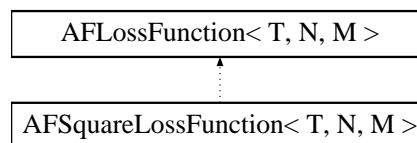
The documentation for this class was generated from the following file:

- C:/Users/Aryan/CLionProjects/CNet/src/AFFunctions.h

5.2 AFLossFunction< T, N, M > Class Template Reference

```
#include <AFFunctions.h>
```

Inheritance diagram for AFLossFunction< T, N, M >:



Public Member Functions

- void [evaluate](#) (array< T, N > *actualVals, array< T, N > *expectedVals, array< T, N > *output)
- void [derivative](#) (array< T, N > *actualVals, array< T, N > *expectedVals, array< T, N > *output)

5.2.1 Member Function Documentation

5.2.1.1 derivative()

```
template<typename T , size_t N, size_t M>
void AFLossFunction< T, N, M >::derivative (
    array< T, N > * actualVals,
    array< T, N > * expectedVals,
    array< T, N > * output ) [inline]
```


5.2.1.2 evaluate()

```
template<typename T , size_t N, size_t M>
void AFLossFunction< T, N, M >::evaluate (
    array< T, N > * actualVals,
    array< T, N > * expectedVals,
    array< T, N > * output ) [inline]
```

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

- C:/Users/Aryan/CLionProjects/CNet/src/AFFunctions.h

5.3 AFMatrix< T, ROWS, COLS > Class Template Reference

```
#include <AFMatrix.h>
```

Public Member Functions

- [AFMatrix](#) ()
- [AFMatrix](#) ([AFMatrix](#)< T, ROWS, COLS > *copyFrom)
- [AFMatrix](#) (array< T, ROWS *COLS > *copyFromArray)
- [~AFMatrix](#) ()
- int [getIndex](#) (int row, int col)
- T [getValue](#) (int row, int col)
- array< T, ROWS > * [getCol](#) (int col)
- array< T, COLS > * [getRow](#) (int row)
- void [setValue](#) (int row, int col, T newValue)
- template<size_t OTHER_COLS>
 void [innerProduct](#) ([AFMatrix](#)< T, COLS, OTHER_COLS > *other, [AFMatrix](#)< T, ROWS, OTHER_COLS > *out)
- template<size_t OTHER_COLS>
 void [innerProduct](#) ([AFMatrix](#)< T, COLS, OTHER_COLS > *other, [AFMatrix](#)< T, ROWS, OTHER_COLS > *out, size_t outStartRow, size_t outStartCol)
- template<size_t COLSOUT>
 void [innerProduct](#) (array< T, COLS > *other, [AFMatrix](#)< T, ROWS, COLSOUT > *out, int outCol)
- void [innerProduct](#) (array< T, COLS > *other, array< T, ROWS > *out)
- void [transpose](#) ([AFMatrix](#)< T, COLS, ROWS > *out)
- [AFMatrix](#) * [transpose](#) ()
- void [scale](#) (double factor, [AFMatrix](#) *out)
- void [add](#) ([AFMatrix](#)< T, ROWS, COLS > *other, [AFMatrix](#)< T, ROWS, COLS > *out)
- void [subtract](#) ([AFMatrix](#)< T, ROWS, COLS > *other, [AFMatrix](#)< T, ROWS, COLS > *out)
- array< T, ROWS *COLS > * [toArray](#) ()
- template<size_t ROWSDST, size_t COLSDST>
 void [copyValues](#) ([AFMatrix](#)< T, ROWSDST, COLSDST > *dst, [AFMatrix](#)< T, ROWS, COLS > *src)
- template<size_t ROWSDST, size_t COLSDST>
 void [copyValues](#) ([AFMatrix](#)< T, ROWSDST, COLSDST > *dst, [AFMatrix](#)< T, ROWS, COLS > *src, size_t srcRowStart, size_t srcColStart)
- void [copyValues](#) ([AFMatrix](#)< T, ROWS, COLS > *dst, array< T, ROWS *COLS > *src)
- template<typename T1 , size_t N>
 void [copyValues](#) (array< T1, N > *dst, array< T1, N > *src)
- bool [equals](#) ([AFMatrix](#)< T, ROWS, COLS > *otherMat)

Public Attributes

- int [numRows](#)
- int [numCols](#)
- array< T, ROWS *COLS > * [vals](#)

5.3.1 Constructor & Destructor Documentation

5.3.1.1 AFMatrix() [1/3]

```
template<class T, size_t ROWS, size_t COLS>
AFMatrix< T, ROWS, COLS >::AFMatrix ( ) [inline]
```

5.3.1.2 AFMatrix() [2/3]

```
template<class T, size_t ROWS, size_t COLS>
AFMatrix< T, ROWS, COLS >::AFMatrix (
    AFMatrix< T, ROWS, COLS > * copyFrom ) [inline]
```

Creates a new copy and copies data.

Todo Make this effecient and non-copying

Parameters

<i>copyFrom</i>	
-----------------	--

5.3.1.3 AFMatrix() [3/3]

```
template<class T, size_t ROWS, size_t COLS>
AFMatrix< T, ROWS, COLS >::AFMatrix (
    array< T, ROWS *COLS > * copyFromArray ) [inline]
```

Creates a new copy and copies data from *copyFrom*.

Todo Make this effecient and non-copying

Parameters

<i>copyFrom</i>	
-----------------	--

5.3.1.4 ~AFMatrix()

```
template<class T, size_t ROWS, size_t COLS>
AFMatrix< T, ROWS, COLS >::~~AFMatrix ( ) [inline]
```

5.3.2 Member Function Documentation

5.3.2.1 add()

```
template<class T, size_t ROWS, size_t COLS>
void AFMatrix< T, ROWS, COLS >::add (
    AFMatrix< T, ROWS, COLS > * other,
    AFMatrix< T, ROWS, COLS > * out ) [inline]
```

Adds two matrices and writes result into out

Parameters

<i>other</i>	- The matrix to add to this.
<i>out</i>	- The matrix to write the result to

Warning

requires

5.3.2.2 copyValues() [1/4]

```
template<class T, size_t ROWS, size_t COLS>
template<size_t ROWSDST, size_t COLSDST>
void AFMatrix< T, ROWS, COLS >::copyValues (
    AFMatrix< T, ROWSDST, COLSDST > * dst,
    AFMatrix< T, ROWS, COLS > * src ) [inline]
```

Copies values from src to dst. The two matrices will be exactly identical.

Parameters

<i>dst</i>	
<i>src</i>	

5.3.2.3 copyValues() [2/4]

```
template<class T, size_t ROWS, size_t COLS>
template<size_t ROWSDST, size_t COLSDST>
void AFMatrix< T, ROWS, COLS >::copyValues (
    AFMatrix< T, ROWSDST, COLSDST > * dst,
    AFMatrix< T, ROWS, COLS > * src,
    size_t srcRowStart,
    size_t srcColStart ) [inline]
```

Copies values from `src` to `dst`. The two matrices will be exactly identical.

Parameters

<i>dst</i>	
<i>src</i>	

5.3.2.4 copyValues() [3/4]

```
template<class T, size_t ROWS, size_t COLS>
void AFMatrix< T, ROWS, COLS >::copyValues (
    AFMatrix< T, ROWS, COLS > * dst,
    array< T, ROWS * COLS > * src ) [inline]
```

Copies data from the `src` array to `dst->vals`.

Todo Clarify how things work if matrices can be row-major or column-major.

Parameters

<i>dst</i>	
<i>src</i>	

5.3.2.5 copyValues() [4/4]

```
template<class T, size_t ROWS, size_t COLS>
template<typename T1 , size_t N>
void AFMatrix< T, ROWS, COLS >::copyValues (
    array< T1, N > * dst,
    array< T1, N > * src ) [inline]
```

Copies values from `src` to `dst`. The arrays will be identical afterwards.

Template Parameters

<i>N</i>	- The size of the arrays, must be equal
----------	---

Parameters

<i>dst</i>	
<i>src</i>	

5.3.2.6 equals()

```
template<class T, size_t ROWS, size_t COLS>
bool AFMatrix< T, ROWS, COLS >::equals (
    AFMatrix< T, ROWS, COLS > * otherMat ) [inline]
```

5.3.2.7 getCol()

```
template<class T, size_t ROWS, size_t COLS>
array<T, ROWS>* AFMatrix< T, ROWS, COLS >::getCol (
    int col ) [inline]
```

Parameters

<i>col</i>	
------------	--

Returns

An `std::array` filled with the column values of this matrix

Warning

Delete the new array to free memory

5.3.2.8 getIndex()

```
template<class T, size_t ROWS, size_t COLS>
int AFMatrix< T, ROWS, COLS >::getIndex (
    int row,
    int col ) [inline]
```

Parameters

<i>row</i>	
<i>col</i>	

Returns

the index `i` such that `this->vals[i] = (row, col)`.

5.3.2.9 `getRow()`

```
template<class T, size_t ROWS, size_t COLS>
array<T, COLS>* AFMatrix< T, ROWS, COLS >::getRow (
    int row ) [inline]
```

Parameters

<i>row</i>	
------------	--

Returns

An `std::array` filled with the row values of this matrix

Warning

Delete the new array to free memory

5.3.2.10 `getValue()`

```
template<class T, size_t ROWS, size_t COLS>
T AFMatrix< T, ROWS, COLS >::getValue (
    int row,
    int col ) [inline]
```

Parameters

<i>row</i>	
<i>col</i>	

Returns

the index `i` such that `this->vals[i] = (row, col)`.

5.3.2.11 `innerProduct()` [1/4]

```
template<class T, size_t ROWS, size_t COLS>
template<size_t OTHER_COLS>
```

```
void AFMatrix< T, ROWS, COLS >::innerProduct (
    AFMatrix< T, COLS, OTHER_COLS > * other,
    AFMatrix< T, ROWS, OTHER_COLS > * out ) [inline]
```

Parameters

<i>other</i>	The other matrix to multiply this against
<i>out</i>	The matrix to write the output values

5.3.2.12 innerProduct() [2/4]

```
template<class T, size_t ROWS, size_t COLS>
template<size_t OTHER_COLS>
void AFMatrix< T, ROWS, COLS >::innerProduct (
    AFMatrix< T, COLS, OTHER_COLS > * other,
    AFMatrix< T, ROWS, OTHER_COLS > * out,
    size_t outStartRow,
    size_t outStartCol ) [inline]
```

Parameters

<i>other</i>	The other matrix to multiply this against
<i>out</i>	The matrix to write the output values

5.3.2.13 innerProduct() [3/4]

```
template<class T, size_t ROWS, size_t COLS>
template<size_t COLSOUT>
void AFMatrix< T, ROWS, COLS >::innerProduct (
    array< T, COLS > * other,
    AFMatrix< T, ROWS, COLSOUT > * out,
    int outCol ) [inline]
```

Multiplies a matrix on the left against a array on the right. The array on the right is treated as a column vector.

Precondition

```
this.numCols = other.size()
```

Parameters

<i>other</i>	The other array to inner product with.
<i>out</i>	Output matrix to write values to. It has this.numRows rows and 1 column.

5.3.2.14 innerProduct() [4/4]

```
template<class T, size_t ROWS, size_t COLS>
```



```
void AFMatrix< T, ROWS, COLS >::innerProduct (
    array< T, COLS > * other,
    array< T, ROWS > * out ) [inline]
```

Multiplies a matrix on the left against a vector on the right.

Precondition

```
this.numRows = other.size()
```

Parameters

<i>other</i>	The other vector to inner product with.
<i>out</i>	Output matrix to write values to. It has this.numRows rows and 1 column.

5.3.2.15 scale()

```
template<class T, size_t ROWS, size_t COLS>
void AFMatrix< T, ROWS, COLS >::scale (
    double factor,
    AFMatrix< T, ROWS, COLS > * out ) [inline]
```

Multiplies all entries of a matrix by factor

Parameters

<i>factor</i>	
<i>out</i>	

5.3.2.16 setValue()

```
template<class T, size_t ROWS, size_t COLS>
void AFMatrix< T, ROWS, COLS >::setValue (
    int row,
    int col,
    T newValue ) [inline]
```

Parameters

<i>row</i>	
<i>col</i>	
<i>newValue</i>	The new value to put in this row/col

5.3.2.17 subtract()

```
template<class T, size_t ROWS, size_t COLS>
void AFMatrix< T, ROWS, COLS >::subtract (
    AFMatrix< T, ROWS, COLS > * other,
    AFMatrix< T, ROWS, COLS > * out ) [inline]
```

Subtracts two matrices and writes result into out

Parameters

<i>other</i>	- The matrix to subtract from this.
<i>out</i>	- The matrix to write the result to

5.3.2.18 toArray()

```
template<class T, size_t ROWS, size_t COLS>
array<T, ROWS*COLS>* AFMatrix< T, ROWS, COLS >::toArray ( ) [inline]
```

Dynamically makes a new vector that is `this.numRows * this.numCols` elements long and copies `this.vals` into it.

Returns

The new dynamically allocated vector (we call `reserve()` on it though).

5.3.2.19 transpose() [1/2]

```
template<class T, size_t ROWS, size_t COLS>
void AFMatrix< T, ROWS, COLS >::transpose (
    AFMatrix< T, COLS, ROWS > * out ) [inline]
```

Transposes this matrix and writes result to out

Parameters

<i>out</i>	- Matrix that has <code>this.numCols</code> rows and <code>this.numRows</code> cols. The result will be written to out.
------------	---

5.3.2.20 transpose() [2/2]

```
template<class T, size_t ROWS, size_t COLS>
AFMatrix* AFMatrix< T, ROWS, COLS >::transpose ( ) [inline]
```

Returns

A new, dynamically allocated matrix that is the transpose of this

Warning

Remember to delete the returned Matrix when done

5.3.3 Member Data Documentation**5.3.3.1 numCols**

```
template<class T, size_t ROWS, size_t COLS>
int AFMatrix< T, ROWS, COLS >::numCols
```

5.3.3.2 numRows

```
template<class T, size_t ROWS, size_t COLS>
int AFMatrix< T, ROWS, COLS >::numRows
```

5.3.3.3 vals

```
template<class T, size_t ROWS, size_t COLS>
array<T, ROWS*COLS>* AFMatrix< T, ROWS, COLS >::vals
```

This is the values of the matrix stored in one long array regardless of the matrix's actual shape. The values are stored in row-by-row

Todo Is vals dynamically allocated or what??

a	b
c	d
e	f

= [a b c d e f]

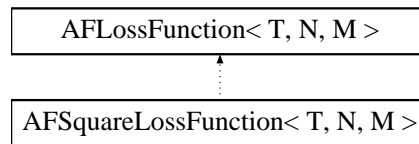
The documentation for this class was generated from the following file:

- C:/Users/Aryan/CLionProjects/CNet/src/AFMatrix.h

5.4 AFSquareLossFunction< T, N, M > Class Template Reference

```
#include <AFFunctions.h>
```

Inheritance diagram for AFSquareLossFunction< T, N, M >:



Public Member Functions

- T [evaluate](#) (array< T, N > *actualVals, array< T, N > *expectedVals)
- void [derivative](#) (array< T, N > *actualVals, array< T, N > *expectedVals, array< T, N > *output)

5.4.1 Member Function Documentation

5.4.1.1 derivative()

```
template<typename T , size_t N, size_t M>
void AFSquareLossFunction< T, N, M >::derivative (
    array< T, N > * actualVals,
    array< T, N > * expectedVals,
    array< T, N > * output ) [inline]
```

Finds derivative of squared loss $L(\text{inputVal}, \text{actualVal}, \text{expectedVal})$ w.r.t. actualVals.

Parameters

<i>actualVals</i>	
<i>expectedVals</i>	
<i>output</i>	

5.4.1.2 evaluate()

```
template<typename T , size_t N, size_t M>
T AFSquareLossFunction< T, N, M >::evaluate (
    array< T, N > * actualVals,
    array< T, N > * expectedVals ) [inline]
```

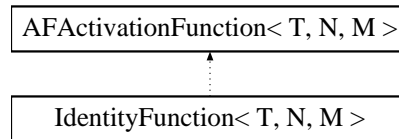
The documentation for this class was generated from the following file:

- C:/Users/Aryan/CLionProjects/CNet/src/[AFFunctions.h](#)

5.5 IdentityFunction< T, N, M > Class Template Reference

```
#include <AFFunctions.h>
```

Inheritance diagram for IdentityFunction< T, N, M >:



Public Member Functions

- void [evaluate](#) (array< T, N > *input, array< T, N > *output)
- void [derivative](#) (array< T, M > *input, array< T, M > *output)

5.5.1 Member Function Documentation

5.5.1.1 derivative()

```
template<typename T , size_t N, size_t M>
void IdentityFunction< T, N, M >::derivative (
    array< T, M > * input,
    array< T, M > * output ) [inline]
```

5.5.1.2 evaluate()

```
template<typename T , size_t N, size_t M>
void IdentityFunction< T, N, M >::evaluate (
    array< T, N > * input,
    array< T, N > * output ) [inline]
```

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

- C:/Users/Aryan/CLionProjects/CNet/src/[AFFunctions.h](#)

5.6 Layer< LEN_IN, LEN_OUT > Class Template Reference

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

Public Member Functions

- [Layer](#) (int [lenIn](#), int [lenOut](#), [AFActivationFunction](#)< double, LEN_IN, LEN_OUT > *activationFn)
- [~Layer](#) ()
- void [randomizeWeights](#) ()
- void [forwardPass](#) (array< double, LEN_IN > *inputVals, array< double, LEN_OUT > *outputVals)
- void [forwardPass](#) (array< double, LEN_IN > *inputVals)
- template<size_t LEN_OUT_NEXT>
void [backpropagate](#) (array< double, LEN_IN > *nextDeltas, [AFMatrix](#)< double, LEN_OUT_NEXT, LEN_IN > *nextWeights, array< double, LEN_OUT > *newDeltas)
- template<size_t LEN_OUT_NEXT>
void [backpropagate](#) (array< double, LEN_IN > *nextDeltas, [AFMatrix](#)< double, LEN_OUT_NEXT, LEN_IN > *nextWeights)
- void [backpropagateBase](#) (array< double, LEN_OUT > *actualVals, array< double, LEN_OUT > *expectedVals, [AFLossFunction](#) *lossFn, array< double, LEN_OUT > *newDeltas)
- void [backpropagateBase](#) (array< double, LEN_OUT > *actualVals, array< double, LEN_OUT > *expectedVals, [AFLossFunction](#) *lossFn)
- void [updateWeights](#) ()

Public Attributes

- int [lenIn](#)
The size of the vector that this layer takes as input.
- int [lenOut](#)
The size of the vector that this layer outputs.
- array< double, LEN_IN > * [inputVals](#)
The values that this layer receives from the previous layer.
- array< double, LEN_OUT > * [sums](#)
The sums after the weights are multiplied by input value'.
- array< double, LEN_OUT > * [deltas](#)
The intermediate gradients of the loss, $deltas[i] = d(Error)/d(sum_i)$
- array< double, LEN_OUT > * [outputVals](#)
The values after the sums are put through the activation function.
- [AFMatrix](#)< double, LEN_OUT, LEN_IN > * [weights](#)
The weights which are multiplied against the input values. This has lenOut rows and lenIn cols.
- [AFMatrix](#)< double, LEN_OUT, LEN_IN > * [weightGradient](#)
The weight gradients. $weightGradient[i,j] = d(Error)/d(weights[i,j])$. Same shape as weights.
- [AFActivationFunction](#)< double, LEN_IN, LEN_OUT > * [activationFunction](#)
*The activation function g such that $outputValues = g(weights * InputVals)$. Note that g takes a vector.*

5.6.1 Detailed Description

```
template<size_t LEN_IN, size_t LEN_OUT>
class Layer< LEN_IN, LEN_OUT >
```

Author

Aryan Falahatpisheh

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Layer()

```
template<size_t LEN_IN, size_t LEN_OUT>
Layer< LEN_IN, LEN_OUT >::Layer (
    int lenIn,
    int lenOut,
    AFActivationFunction< double, LEN_IN, LEN_OUT > * activationFn ) [inline]
```

Parameters

<i>lenIn</i>	The input length of this layer
<i>lenOut</i>	The output size of this layer
<i>activationFn</i>	Pass an AFActivationFunction by value so this layer knows how to calculate output values.

5.6.2.2 ~Layer()

```
template<size_t LEN_IN, size_t LEN_OUT>
Layer< LEN_IN, LEN_OUT >::~~Layer ( ) [inline]
```

5.6.3 Member Function Documentation

5.6.3.1 backpropagate() [1/2]

```
template<size_t LEN_IN, size_t LEN_OUT>
template<size_t LEN_OUT_NEXT>
void Layer< LEN_IN, LEN_OUT >::backpropagate (
    array< double, LEN_IN > * nextDeltas,
    AFMatrix< double, LEN_OUT_NEXT, LEN_IN > * nextWeights,
    array< double, LEN_OUT > * newDeltas ) [inline]
```

Performs backpropogation algorithm. Writes this layer's new d(Err)/d(Sums) into *newDeltas*.

Template Parameters

<i>LEN_OUT_NEXT</i>	The next layer's output length
---------------------	--------------------------------

Parameters

<i>nextDeltas</i>	The next layer's d(Err)/d(sums);
-------------------	----------------------------------

Parameters

<i>nextWeights</i>	
<i>newDeltas</i>	

5.6.3.2 `backpropagate()` [2/2]

```
template<size_t LEN_IN, size_t LEN_OUT>
template<size_t LEN_OUT_NEXT>
void Layer< LEN_IN, LEN_OUT >::backpropagate (
    array< double, LEN_IN > * nextDeltas,
    AFMatrix< double, LEN_OUT_NEXT, LEN_IN > * nextWeights ) [inline]
```

Performs backpropagation algorithm and writes output to `this->deltas`.

Template Parameters

<i>LEN_OUT_NEXT</i>	
---------------------	--

Parameters

<i>nextDeltas</i>	
<i>nextWeights</i>	

5.6.3.3 `backpropagateBase()` [1/2]

```
template<size_t LEN_IN, size_t LEN_OUT>
void Layer< LEN_IN, LEN_OUT >::backpropagateBase (
    array< double, LEN_OUT > * actualVals,
    array< double, LEN_OUT > * expectedVals,
    AFLossFunction * lossFn,
    array< double, LEN_OUT > * newDeltas ) [inline]
```

The backprop algorithm for the last layer. First calculates $d(\text{Err})/d(\text{outputVals})$, which is the derivative of the loss function w.r.t to `actualVals`. It then calculates $d(\text{Err})/d(\text{sums})$.

Parameters

<i>actualVals</i>	
<i>expectedVals</i>	
<i>newDeltas</i>	

5.6.3.4 backpropagateBase() [2/2]

```
template<size_t LEN_IN, size_t LEN_OUT>
void Layer< LEN_IN, LEN_OUT >::backpropagateBase (
    array< double, LEN_OUT > * actualVals,
    array< double, LEN_OUT > * expectedVals,
    AFLossFunction * lossFn ) [inline]
```

The backprop algorithm for the last layer. First calculates $d(\text{Err})/d(\text{outputVals})$, which is the derivative of the loss function w.r.t to `actualVals`. It then calculates $d(\text{Err})/d(\text{sums})$.

Parameters

<i>actualVals</i>	
<i>expectedVals</i>	
<i>newDeltas</i>	

5.6.3.5 forwardPass() [1/2]

```
template<size_t LEN_IN, size_t LEN_OUT>
void Layer< LEN_IN, LEN_OUT >::forwardPass (
    array< double, LEN_IN > * inputVals,
    array< double, LEN_OUT > * outputVals ) [inline]
```

Will perform the forward pass on this [Layer](#). Will take in `inputVals`, calculate weighted sums, and then pass that result to this layer's activation function. The output will be written to `outputVals`.

Parameters

<i>inputVals</i>	
<i>outputVals</i>	- <code>outputVals[i] = this->weights.row(i).innerProduct(inputVals).</code>

5.6.3.6 forwardPass() [2/2]

```
template<size_t LEN_IN, size_t LEN_OUT>
void Layer< LEN_IN, LEN_OUT >::forwardPass (
    array< double, LEN_IN > * inputVals ) [inline]
```

Will perform the forward pass on this [Layer](#). Will take in `inputVals`, calculate weighted sums, and then pass that result to this layer's activation function. The output will be written to `outputVals`.

Parameters

<i>inputVals</i>	
<i>outputVals</i>	- <code>outputVals[i] = this->weights.row(i).innerProduct(inputVals).</code>

5.6.3.7 randomizeWeights()

```
template<size_t LEN_IN, size_t LEN_OUT>
void Layer< LEN_IN, LEN_OUT >::randomizeWeights ( ) [inline]
```

5.6.3.8 updateWeights()

```
template<size_t LEN_IN, size_t LEN_OUT>
void Layer< LEN_IN, LEN_OUT >::updateWeights ( ) [inline]
```

5.6.4 Member Data Documentation

5.6.4.1 activationFunction

```
template<size_t LEN_IN, size_t LEN_OUT>
AFActivationFunction<double, LEN_IN, LEN_OUT>* Layer< LEN_IN, LEN_OUT >::activationFunction
```

The activation function g such that $\text{outputValues} = g(\text{weights} * \text{InputVals})$. Note that g takes a vector.

5.6.4.2 deltas

```
template<size_t LEN_IN, size_t LEN_OUT>
array<double, LEN_OUT>* Layer< LEN_IN, LEN_OUT >::deltas
```

The intermediate gradients of the loss, $\text{deltas}[i] = d(\text{Error})/d(\text{sum}_i)$

5.6.4.3 inputVals

```
template<size_t LEN_IN, size_t LEN_OUT>
array<double, LEN_IN>* Layer< LEN_IN, LEN_OUT >::inputVals
```

The values that this layer receives from the previous layer.

5.6.4.4 lenIn

```
template<size_t LEN_IN, size_t LEN_OUT>
int Layer< LEN_IN, LEN_OUT >::lenIn
```

The size of the vector that this layer takes as input.

Hello!

5.6.4.5 lenOut

```
template<size_t LEN_IN, size_t LEN_OUT>
int Layer< LEN_IN, LEN_OUT >::lenOut
```

The size of the vector that this layer outputs.

5.6.4.6 outputVals

```
template<size_t LEN_IN, size_t LEN_OUT>
array<double, LEN_OUT>* Layer< LEN_IN, LEN_OUT >::outputVals
```

The values after the sums are put through the activation function.

5.6.4.7 sums

```
template<size_t LEN_IN, size_t LEN_OUT>
array<double, LEN_OUT>* Layer< LEN_IN, LEN_OUT >::sums
```

The sums after the weights are multiplied by input value'.

5.6.4.8 weightGradient

```
template<size_t LEN_IN, size_t LEN_OUT>
AFMatrix<double, LEN_OUT, LEN_IN>* Layer< LEN_IN, LEN_OUT >::weightGradient
```

The weight gradients. `weightGradient[i,j] = d(Error)/d(weights[i,j])`. Same shape as weights.

5.6.4.9 weights

```
template<size_t LEN_IN, size_t LEN_OUT>
AFMatrix<double, LEN_OUT, LEN_IN>* Layer< LEN_IN, LEN_OUT >::weights
```

The weights which are multiplied against the input values. This has `lenOut` rows and `lenIn` cols.

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

- C:/Users/Aryan/CLionProjects/CNet/src/[Layer.h](#)

5.7 Net< T > Class Template Reference

```
#include <Net.h>
```

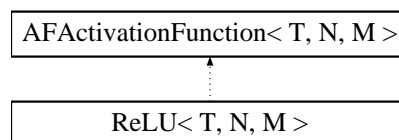
The documentation for this class was generated from the following file:

- C:/Users/Aryan/CLionProjects/CNet/src/[Net.h](#)

5.8 ReLU< T, N, M > Class Template Reference

```
#include <AFFunctions.h>
```

Inheritance diagram for ReLU< T, N, M >:



Public Member Functions

- void [evaluate](#) (array< T, N > *input, array< T, N > *output)
- void [derivative](#) (array< T, M > *input, array< T, M > *output)

5.8.1 Member Function Documentation

5.8.1.1 derivative()

```
template<typename T , size_t N, size_t M>
void ReLU< T, N, M >::derivative (
    array< T, M > * input,
    array< T, M > * output ) [inline]
```

5.8.1.2 evaluate()

```
template<typename T , size_t N, size_t M>
void ReLU< T, N, M >::evaluate (
    array< T, N > * input,
    array< T, N > * output ) [inline]
```

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

- C:/Users/Aryan/CLionProjects/CNet/src/[AFFunctions.h](#)

Chapter 6

File Documentation

6.1 C:/Users/Aryan/CLionProjects/CNet/src/AFFunctions.h File Reference

```
#include <math.h>
#include <array>
```

Classes

- class [AFActivationFunction< T, N, M >](#)
- class [ReLU< T, N, M >](#)
- class [IdentityFunction< T, N, M >](#)
- class [AFLossFunction< T, N, M >](#)
- class [AFSquareLossFunction< T, N, M >](#)

6.2 C:/Users/Aryan/CLionProjects/CNet/src/AFMatrix.cpp File Reference

```
#include "AFMatrix.h"
```

6.3 C:/Users/Aryan/CLionProjects/CNet/src/AFMatrix.h File Reference

```
#include <array>
```

Classes

- class [AFMatrix< T, ROWS, COLS >](#)

Macros

- [#define ACCEPTABLE_DOUBLE_DIFF 0.000000001](#)

Functions

- `template<typename T, size_t N>`
`T vectorInnerProductBounded (array< T, N > *vec1, array< T, N > *vec2, size_t start1, size_t start2, size_t len)`
- `template<typename T, size_t N>`
`T vectorInnerProduct (array< T, N > *vec1, array< T, N > *vec2)`
- `template<size_t N>`
`bool doubleVectorEqual (array< double, N > *vec1, array< double, N > *vec2)`

6.3.1 Macro Definition Documentation

6.3.1.1 ACCEPTABLE_DOUBLE_DIFF

```
#define ACCEPTABLE_DOUBLE_DIFF 0.000000001
```

6.3.2 Function Documentation

6.3.2.1 doubleVectorEqual()

```
template<size_t N>
bool doubleVectorEqual (
    array< double, N > * vec1,
    array< double, N > * vec2 )
```

6.3.2.2 vectorInnerProduct()

```
template<typename T, size_t N>
T vectorInnerProduct (
    array< T, N > * vec1,
    array< T, N > * vec2 )
```

Template Parameters

<i>T</i>	The type of data in the vectors being multiplies. Probably a <code>double</code> .
----------	--

Parameters

<i>vec1</i>	- Left vector
<i>vec2</i>	- Right vector

Returns

The dot product (inner product) of two vectors.

Precondition

vec1 and vec2 have the same length.

6.3.2.3 vectorInnerProductBounded()

```
template<typename T , size_t N>
T vectorInnerProductBounded (
    array< T, N > * vec1,
    array< T, N > * vec2,
    size_t start1,
    size_t start2,
    size_t len )
```

Template Parameters

<i>T</i>	The type of data in the vectors being multiplies. Probably a double.
----------	--

Parameters

<i>vec1</i>	- Left vector
<i>vec2</i>	- Right vector

Returns

The dot product (inner product) of two vectors.

Precondition

vec1 and vec2 have the same length.

6.4 C:/Users/Aryan/CLionProjects/CNet/src/Layer.cpp File Reference

```
#include "Layer.h"
```

6.5 C:/Users/Aryan/CLionProjects/CNet/src/Layer.h File Reference

```
#include "AFFunctions.h"
#include "AFMatrix.h"
#include <iostream>
```

Classes

- class [Layer](#)< LEN_IN, LEN_OUT >

6.6 C:/Users/Aryan/CLionProjects/CNet/src/main.cpp File Reference

```
#include <iostream>
```

Functions

- int [main](#) ()

6.6.1 Function Documentation

6.6.1.1 main()

```
int main ( )
```

6.7 C:/Users/Aryan/CLionProjects/CNet/src/Net.cpp File Reference

```
#include "Net.h"
```

6.8 C:/Users/Aryan/CLionProjects/CNet/src/Net.h File Reference

Classes

- class [Net](#)< T >

Index

- ~AFMatrix
 - AFMatrix, [13](#)
- ~Layer
 - Layer, [25](#)
- ACCEPTABLE_DOUBLE_DIFF
 - AFMatrix.h, [34](#)
- AFActivationFunction
 - derivative, [9](#)
 - evaluate, [9](#)
- AFActivationFunction< T, N, M >, [9](#)
- AFLossFunction
 - derivative, [10](#)
 - evaluate, [10](#)
- AFLossFunction< T, N, M >, [10](#)
- AFMatrix
 - ~AFMatrix, [13](#)
 - AFMatrix, [12](#)
 - add, [13](#)
 - copyValues, [13](#), [14](#)
 - equals, [15](#)
 - getCol, [15](#)
 - getIndex, [15](#)
 - getRow, [16](#)
 - getValue, [16](#)
 - innerProduct, [16](#), [18](#)
 - numCols, [21](#)
 - numRows, [21](#)
 - scale, [19](#)
 - setValue, [19](#)
 - subtract, [19](#)
 - toArray, [20](#)
 - transpose, [20](#)
 - vals, [21](#)
- AFMatrix< T, ROWS, COLS >, [11](#)
- AFMatrix.h
 - ACCEPTABLE_DOUBLE_DIFF, [34](#)
 - doubleVectorEqual, [34](#)
 - vectorInnerProduct, [34](#)
 - vectorInnerProductBounded, [35](#)
- AFSquareLossFunction
 - derivative, [22](#)
 - evaluate, [22](#)
- AFSquareLossFunction< T, N, M >, [22](#)
- activationFunction
 - Layer, [28](#)
- add
 - AFMatrix, [13](#)
- backpropagate
 - Layer, [25](#), [26](#)
- backpropagateBase
 - Layer, [26](#)
- C:/Users/Aryan/CLionProjects/CNet/src/AFFunctions.h, [33](#)
- C:/Users/Aryan/CLionProjects/CNet/src/AFMatrix.cpp, [33](#)
- C:/Users/Aryan/CLionProjects/CNet/src/AFMatrix.h, [33](#)
- C:/Users/Aryan/CLionProjects/CNet/src/Layer.cpp, [35](#)
- C:/Users/Aryan/CLionProjects/CNet/src/Layer.h, [35](#)
- C:/Users/Aryan/CLionProjects/CNet/src/Net.cpp, [36](#)
- C:/Users/Aryan/CLionProjects/CNet/src/Net.h, [36](#)
- C:/Users/Aryan/CLionProjects/CNet/src/main.cpp, [36](#)
- copyValues
 - AFMatrix, [13](#), [14](#)
- deltas
 - Layer, [28](#)
- derivative
 - AFActivationFunction, [9](#)
 - AFLossFunction, [10](#)
 - AFSquareLossFunction, [22](#)
 - IdentityFunction, [23](#)
 - ReLU, [30](#)
- doubleVectorEqual
 - AFMatrix.h, [34](#)
- equals
 - AFMatrix, [15](#)
- evaluate
 - AFActivationFunction, [9](#)
 - AFLossFunction, [10](#)
 - AFSquareLossFunction, [22](#)
 - IdentityFunction, [23](#)
 - ReLU, [30](#)
- forwardPass
 - Layer, [27](#)
- getCol
 - AFMatrix, [15](#)
- getIndex
 - AFMatrix, [15](#)
- getRow
 - AFMatrix, [16](#)
- getValue
 - AFMatrix, [16](#)
- IdentityFunction
 - derivative, [23](#)

- evaluate, [23](#)
- IdentityFunction< T, N, M >, [23](#)
- innerProduct
 - AFMatrix, [16](#), [18](#)
- inputVals
 - Layer, [28](#)
- Layer
 - ~Layer, [25](#)
 - activationFunction, [28](#)
 - backpropagate, [25](#), [26](#)
 - backpropagateBase, [26](#)
 - deltas, [28](#)
 - forwardPass, [27](#)
 - inputVals, [28](#)
 - Layer, [25](#)
 - lenIn, [28](#)
 - lenOut, [29](#)
 - outputVals, [29](#)
 - randomizeWeights, [28](#)
 - sums, [29](#)
 - updateWeights, [28](#)
 - weightGradient, [29](#)
 - weights, [29](#)
- Layer< LEN_IN, LEN_OUT >, [23](#)
- lenIn
 - Layer, [28](#)
- lenOut
 - Layer, [29](#)
- main
 - main.cpp, [36](#)
- main.cpp
 - main, [36](#)
- Net< T >, [30](#)
- numCols
 - AFMatrix, [21](#)
- numRows
 - AFMatrix, [21](#)
- outputVals
 - Layer, [29](#)
- randomizeWeights
 - Layer, [28](#)
- ReLU< T, N, M >, [30](#)
- ReLU
 - derivative, [30](#)
 - evaluate, [30](#)
- scale
 - AFMatrix, [19](#)
- setValue
 - AFMatrix, [19](#)
- subtract
 - AFMatrix, [19](#)
- sums
 - Layer, [29](#)
- toArray
 - AFMatrix, [20](#)
- transpose
 - AFMatrix, [20](#)
- updateWeights
 - Layer, [28](#)
- vals
 - AFMatrix, [21](#)
- vectorInnerProduct
 - AFMatrix.h, [34](#)
- vectorInnerProductBounded
 - AFMatrix.h, [35](#)
- weightGradient
 - Layer, [29](#)
- weights
 - Layer, [29](#)