CNet

Generated by Doxygen 1.8.13

Contents

6 File Documentation 33

Chapter 1

Todo List

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

Make this effecient and non-copying

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

Make this effecient 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:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

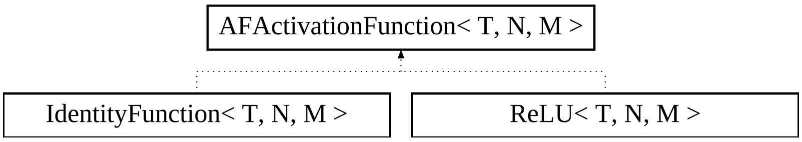
Chapter 5

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]
```

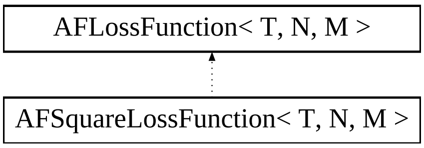
The documentation for this class was generated from the following file:

- 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

copyFrom	

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

copyFrom	

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

other	- The matrix to add to `this`.
out	- 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 identitcal.

Parameters

dst	
src	

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 identitcal.

Parameters

dst	
src	

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

dst	
src	

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

N	- The size of the arrays, must be equal

Parameters

dst	
src	

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

col	

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

row	
col	

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

row	

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

row	
col	

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

other	The other matrix to multiply this against
out	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

other	The other matrix to multiply this against
out	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

other	The other array to inner product with.
out	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

other	The other vector to inner product with.
out	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

factor	
out	

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

row	
col	
newValue	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

other	- The matrix to subtract from `this`.
out	- 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

out	- Matrix that has this.numCols rows and this.numRows 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 | = [a b c d e f]
| e f |

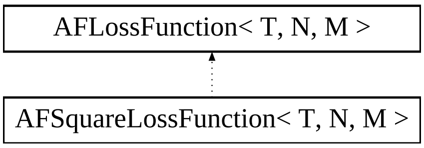
The documentation for this class was generated from the following file:

- 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(inputVal, actualVal, expectedVal) w.r.t. actualVals.

Parameters

actualVals	
expectedVals	
output	

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]
```

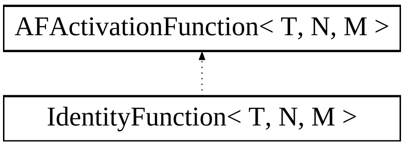
The documentation for this class was generated from the following file:

- 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

lenIn	The input length of this layer
lenOut	The output size of this layer
activationFn	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

LEN_OUT_NEXT	The next layer's output length

Parameters

nextDeltas	The next layer's d(Err)/d(sums);

Parameters

nextWeights	
newDeltas	

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 backpropogation algorithm and writes output to `this->deltas`.

Template Parameters

LEN_OUT_NEXT	

Parameters

nextDeltas	
nextWeights	

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(Err)/d(outputVals)`, which is the derivative of the loss function w.r.t to `actualVals`. It then calculates d(Err)/d(sums)‘.

Parameters

actualVals	
expectedVals	
newDeltas	

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(Err)/d(outputVals)`, which is the derivative of the loss function w.r.t to `actualVals`. It then calculates d(Err)/d(sums)'.

Parameters

actualVals	
expectedVals	
newDeltas	

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

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

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

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

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 ‘outputValues = g(weights ∗ 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, `deltas[i] = d(Error)/d(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>`

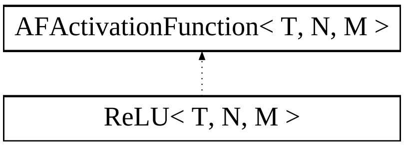
The documentation for this class was generated from the following file:

- 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

T	The type of data in the vectors being multiplies. Probably a `double`.

Parameters

vec1	- Left vector
vec2	- 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

T	The type of data in the vectors being multiplies. Probably a `double`.

Parameters

vec1	- Left vector
vec2	- 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