ML perceptron (P1)

v0.1

Generated by Doxygen 1.13.2

Chapter 1

ML-perceptron

1.1 Student

Name: Stan Merlijn

Student nummer: 1863967

1.2 Introduction

In this repo we are going to implement and test perceptrons, perceptron layers and a perceptron networks(neural network). Theset are going to be tested by creating AND, OR, INVERT, NAND, XOR and half adder logic gates. the reader can be found `here`

1.3 Documentation

For this assignment, documentation was generated using Doxygen. The LaTeX documentation can be found `here` and if you want to run the HTML local website, you can open the `index.html`.

1.4 Installing

Enter the test dir then

Generate build files:

```
cmake -S . -B build
```

Build the project:

```
cmake --build build
```

Run the executable:

```
./build/MLPerceptronTest
```

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 Perceptron Class Reference

A simple perceptron model for binary classification.

```
#include <perceptron.hpp>
```

Public Member Functions

- Perceptron (std::vector< double > weights, double bias, double learningRate)

  Constructs a Perceptron with given weights, bias, and learning rate.
- int predict (const std::vector< int > &inputs) const

  Predicts the output for a given input vector.
- void train (const std::vector< std::vector< int > > &inputs, const std::vector< int > &targets, int epochs)

  Trains the perceptron using the given dataset. Using th learning rule to update the weights.
- void __str__ (int verbose) const

  Prints perceptron details.

4.1.1 Detailed Description

A simple perceptron model for binary classification.

Definition at line 20 of file perceptron.hpp.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Perceptron()

```
Perceptron::Perceptron (
            std::vector< double > weights,
            double bias,
            double learningRate)
```

Constructs a Perceptron with given weights, bias, and learning rate.

Parameters

weights	Initial weights.
bias	Initial bias.
learningRate	Learning rate for training.

Definition at line 13 of file perceptron.cpp.

4.1.3 Member Function Documentation

4.1.3.1 __str__()

```
void Perceptron::__str__ (
            int verbose) const
```

Prints perceptron details.

Parameters

verbose	Verbosity level.

Definition at line 49 of file perceptron.cpp.

4.1.3.2 predict()

```
int Perceptron::predict (
            const std::vector< int > & inputs) const
```

Predicts the output for a given input vector.

Parameters

inputs	Input vector.

Returns

1 if activated, otherwise 0.

Definition at line 16 of file perceptron.cpp.

4.1.3.3 train()

```
void Perceptron::train (
            const std::vector< std::vector< int > > & inputs,
            const std::vector< int > & targets,
            int epochs)
```

Trains the perceptron using the given dataset. Using th learning rule to update the weights.

Parameters

inputs	Input samples.
targets	Target outputs.
epochs	Number of training iterations.

Definition at line 27 of file perceptron.cpp.

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

- /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/header/perceptron.hpp
- /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/perceptron.cpp

4.2 PerceptronLayer Class Reference

Represents a layer of perceptrons in a neural network.

```
#include <perceptronLayer.hpp>
```

Public Member Functions

- PerceptronLayer (const std::vector< Perceptron > &neurons)

  Constructs a perceptron layer.
- std::vector< int > feedForward (const std::vector< int > &input) const

  Feeds input forward through the layer.
- void __str__ (int verbose) const

  Prints layer details.

4.2.1 Detailed Description

Represents a layer of perceptrons in a neural network.

Definition at line 20 of file perceptronLayer.hpp.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 PerceptronLayer()

```
PerceptronLayer::PerceptronLayer (
            const std::vector< Perceptron > & neurons)
```

Constructs a perceptron layer.

Parameters

neurons	List of perceptrons.

Definition at line 13 of file perceptronLayer.cpp.

4.2.3 Member Function Documentation

4.2.3.1 __str__()

```
void PerceptronLayer::__str__ (
             int verbose) const
```

Prints layer details.

Parameters

verbose	Verbosity level.

Definition at line 27 of file perceptronLayer.cpp.

4.2.3.2 feedForward()

```
std::vector< int > PerceptronLayer::feedForward (
             const std::vector< int > & input) const
```

Feeds input forward through the layer.

Parameters

input	Input vector.

Returns

Output vector after applying all perceptrons.

Definition at line 16 of file perceptronLayer.cpp.

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

- /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/header/perceptronLayer.hpp
- /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/perceptronLayer.cpp

4.3 PerceptronNetwork Class Reference

Represents a multi-layer perceptron network.

```
#include <perceptronNetwork.hpp>
```

Public Member Functions

- PerceptronNetwork (std::vector< PerceptronLayer > layers)

  Constructs a perceptron network.
- std::vector< int > feedForward (const std::vector< int > &input) const

  Feeds input forward through the network.
- void __str__ (int verbose) const

  Prints network details.

4.3.1 Detailed Description

Represents a multi-layer perceptron network.

Definition at line 20 of file perceptronNetwork.hpp.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 PerceptronNetwork()

```
PerceptronNetwork::PerceptronNetwork (
             std::vector< PerceptronLayer > layers)
```

Constructs a perceptron network.

Parameters

layers	List of perceptron layers.

Definition at line 15 of file perceptronNetwork.cpp.

4.3.3 Member Function Documentation

4.3.3.1 __str__()

```
void PerceptronNetwork::__str__ (
             int verbose) const
```

Prints network details.

Parameters

verbose	Verbosity level.

Definition at line 28 of file perceptronNetwork.cpp.

4.3.3.2 feedForward()

```
std::vector< int > PerceptronNetwork::feedForward (
             const std::vector< int > & input) const
```

Feeds input forward through the network.

Parameters

input	Input vector.

Returns

Output vector after processing through all layers.

Definition at line 18 of file perceptronNetwork.cpp.

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

- /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/header/perceptronNetwork.hpp
- /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/perceptronNetwork.cpp

Chapter 5

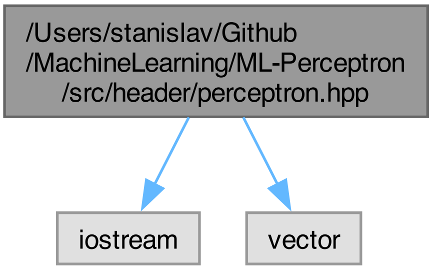
File Documentation

5.1 /Users/stanislav/Github/MachineLearning/ML-↩ Perceptron/src/header/perceptron.hpp File Reference

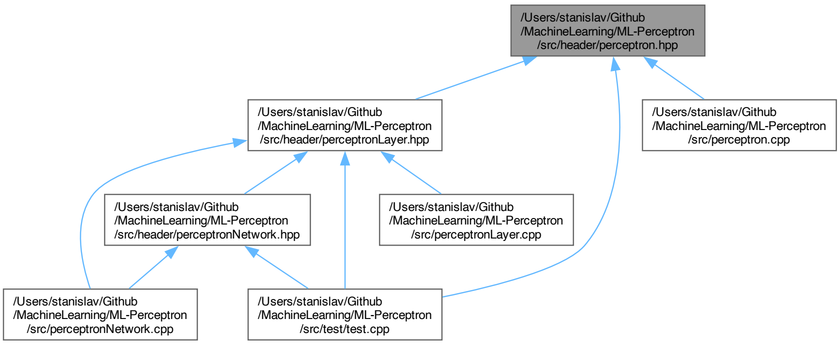
In this file the Perceptron class is defined.

```
#include <iostream>
#include <vector>
```

Include dependency graph for perceptron.hpp:



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


/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/header/perceptron.hpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/header/perceptronLayer.hpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/perceptron.cpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/header/perceptronNetwork.hpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/perceptronLayer.cpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/perceptronNetwork.cpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/test/test.cpp

Classes

- class Perceptron

  A simple perceptron model for binary classification.

5.1.1 Detailed Description

In this file the Perceptron class is defined.

Author

Stan Merlijn

Version

0.1

Date

2025-02-12

Copyright

Copyright (c) 2025

Definition in file perceptron.hpp.

5.2 perceptron.hpp

Go to the documentation of this file.

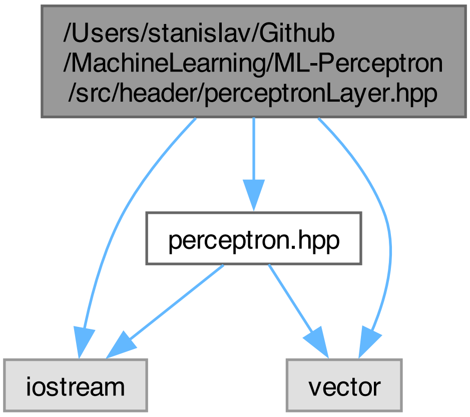
```
#pragma once
#include <iostream>
#include <vector>

class Perceptron
{
public:
    Perceptron(std::vector<double> weights, double bias, double learningRate);

    int predict(const std::vector<int>& inputs) const;

    void train(const std::vector<std::vector<int>>& inputs, const std::vector<int>& targets, int epochs);

    void __str__(int verbose) const;

private:
        std::vector<double> weights;
        double bias;
        double learningRate;
};
```

5.3 /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/header/perceptronLayer.hpp File Reference

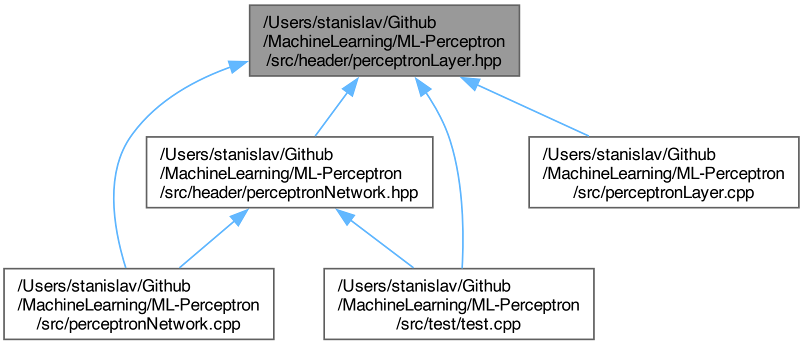
In this file the PerceptronLayer class is defined.

```
#include "perceptron.hpp"
#include <iostream>
#include <vector>
```

Include dependency graph for perceptronLayer.hpp:


/Users/stanislav/Github
/MachineLearning/ML-Perceptron
/src/header/perceptronLayer.hpp
perceptron.hpp
iostream
vector

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


/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/header/perceptronLayer.hpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/header/perceptronNetwork.hpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/perceptronLayer.cpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/perceptronNetwork.cpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/test/test.cpp

Classes

- class PerceptronLayer

  Represents a layer of perceptrons in a neural network.

5.3.1 Detailed Description

In this file the PerceptronLayer class is defined.

Author

Stan Merlijn

Version

0.1

Date

2025-02-12

Copyright

Copyright (c) 2025

Definition in file perceptronLayer.hpp.

5.4 perceptronLayer.hpp

Go to the documentation of this file.

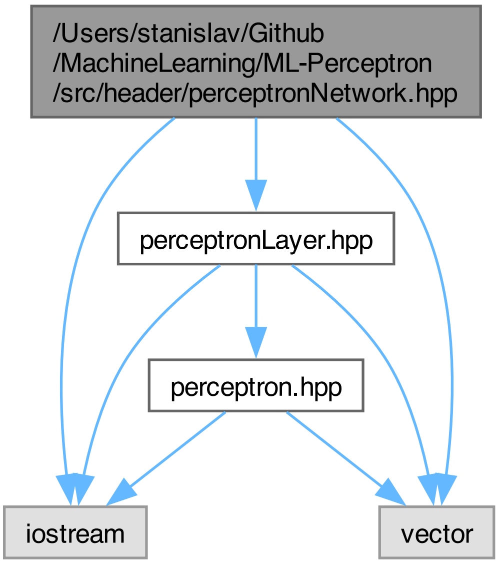
```
#pragma once
#include "perceptron.hpp"
#include <iostream>
#include <vector>

class PerceptronLayer
{
public:
    PerceptronLayer(const std::vector<Perceptron>& neurons);

    std::vector<int> feedForward(const std::vector<int>& input) const;

    void __str__(int verbose) const;

private:
    std::vector<Perceptron> neurons;
};
```

5.5 /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/header/perceptronNetwork.hpp File Reference

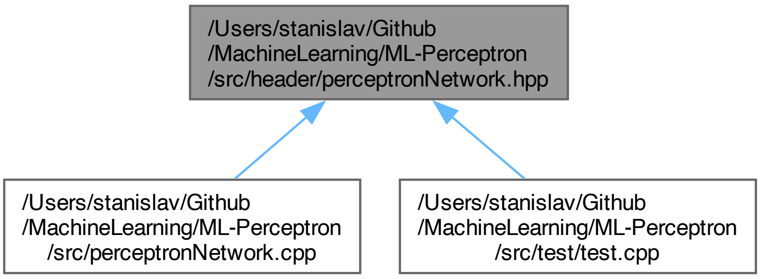
In this file the PerceptronNetwork class is defined.

```
#include "perceptronLayer.hpp"
#include <iostream>
#include <vector>
```

Include dependency graph for perceptronNetwork.hpp:



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


/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/header/perceptronNetwork.hpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/perceptronNetwork.cpp
/Users/stanislav/Github /MachineLearning/ML-Perceptron /src/test/test.cpp

Classes

- class PerceptronNetwork

  Represents a multi-layer perceptron network.

5.5.1 Detailed Description

In this file the PerceptronNetwork class is defined.

Author

Stan Merlijn

Version

0.1

Date

2025-02-12

Copyright

Copyright (c) 2025

Definition in file perceptronNetwork.hpp.

5.6 perceptronNetwork.hpp

Go to the documentation of this file.

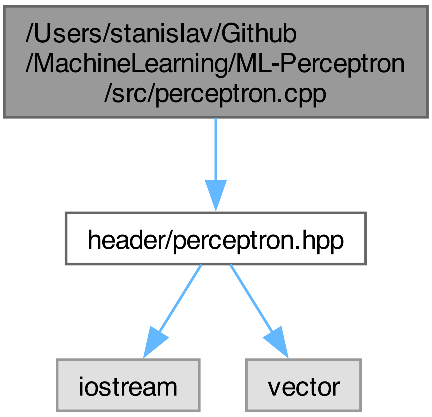
```

#pragma once
#include "perceptronLayer.hpp"
#include <iostream>
#include <vector>

class PerceptronNetwork
{
public:
    PerceptronNetwork(std::vector<PerceptronLayer> layers);

    std::vector<int> feedForward(const std::vector<int>& input) const;

    void __str__(int verbose) const;

private:
    std::vector<PerceptronLayer> layers;

};
```

5.7 /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/perceptron.cpp File Reference

Implementation of the Perceptron class.

```
#include "header/perceptron.hpp"
```

Include dependency graph for perceptron.cpp:



5.7.1 Detailed Description

Implementation of the Perceptron class.

Author

Stan Merlijn

Version

0.1

Date

2025-02-12

Copyright

Copyright (c) 2025

Definition in file perceptron.cpp.

5.8 perceptron.cpp

Go to the documentation of this file.

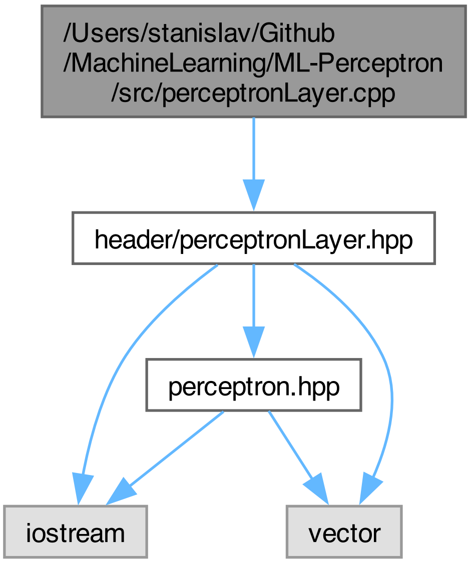
```cpp

#include "header/perceptron.hpp"

Perceptron::Perceptron(std::vector<double> weights, double bias, double learningRate)
    : weights(weights), bias(bias), learningRate(learningRate) {}

int Perceptron::predict(const std::vector<int>& inputs) const
{
    // Dot prodcut for the perceptron
    double dot_product = bias;
    for (int i = 0; i < weights.size(); i++) {
        dot_product += weights[i] * inputs[i];
    }
    // Threshold function
    return dot_product >= 0 ? 1 : 0;
}

void Perceptron::train(const std::vector<std::vector<int»& inputs, const std::vector<int>& targets, int epochs)
{
    // ensure both arrays are the same size
    if (inputs.size() != targets.size()) return;

    // Train the perceptron
    for (int epoch = 0; epoch < epochs; epoch++) {
        for (int i = 0; i < inputs.size(); i++) {
            // get the prediction
            double pred = predict(inputs[i]);
            // Calculate the error based of the target value
            double error = targets[i] - pred;
            // Update each weight based on the input value
            for (int j = 0; j < weights.size(); j++) {
                weights[j] += learningRate * error * inputs[i][j];
            }
            // Update bias
            bias += learningRate * error;
        }
    }
}

void Perceptron::__str__(int verbose) const
{
    // Printing the weights
    std::cout « "weights for perceptron:\n";
    for (auto i : weights)
        std::cout « i « " ";

    // Other info
    if (verbose >= 1) {
        std::cout « "\nbias = " « bias « "\n";
        std::cout « "Learning rate = " « learningRate « std::endl;
    }
}
```

5.9 /Users/stanislav/Github/MachineLearning/ML-↩ Perceptron/src/perceptronLayer.cpp File Reference

Implementation of the PerceptronLayer class.

```
#include "header/perceptronLayer.hpp"
```

Include dependency graph for perceptronLayer.cpp:



5.9.1 Detailed Description

Implementation of the PerceptronLayer class.

Author

Stan Merlijn

Version

0.1

Date

2025-02-12

Copyright

Copyright (c) 2025

Definition in file perceptronLayer.cpp.

5.10 perceptronLayer.cpp

Go to the documentation of this file.

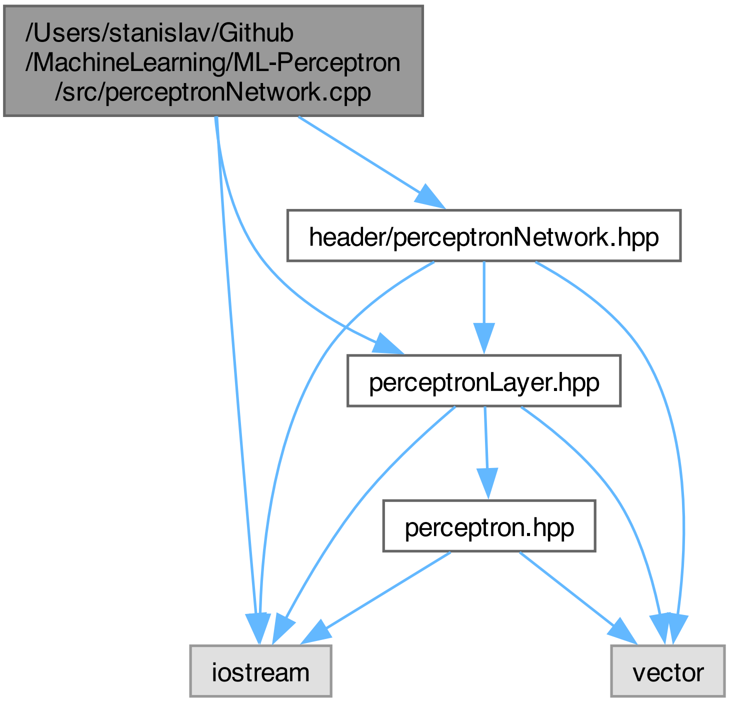
```

#include "header/perceptronLayer.hpp"

PerceptronLayer::PerceptronLayer(const std::vector<Perceptron>& neurons)
    : neurons(neurons) {}

std::vector<int> PerceptronLayer::feedForward(const std::vector<int>& input) const
{
    // Predict the output for each perceptron
    std::vector<int> outputs;
    // Propagate the input through each layer sequentially. Also called feedforward.
    for (const Perceptron& neuron : neurons) {
        outputs.push_back(neuron.predict(input));
    }
    return outputs;
}

void PerceptronLayer::__str__(int verbose) const
{
    // For each neuron in the layer print the data
    for (const Perceptron& neuron : neurons) {
        neuron.__str__(verbose);
    }
}
```

5.11 /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/perceptronNetwork.cpp File Reference

Implementation of the PerceptronNetwork class.

```
#include "header/perceptronNetwork.hpp"
#include "header/perceptronLayer.hpp"
#include <iostream>
```

Include dependency graph for perceptronNetwork.cpp:


/Users/stanislav/Github
/MachineLearning/ML-Perceptron
/src/perceptronNetwork.cpp
header/perceptronNetwork.hpp
perceptronLayer.hpp
perceptron.hpp
iostream
vector

5.11.1 Detailed Description

Implementation of the PerceptronNetwork class.

Author

Stan Merlijn

Version

0.1

Date

2025-02-12

Copyright

Copyright (c) 2025

Definition in file perceptronNetwork.cpp.

5.12 perceptronNetwork.cpp

Go to the documentation of this file.

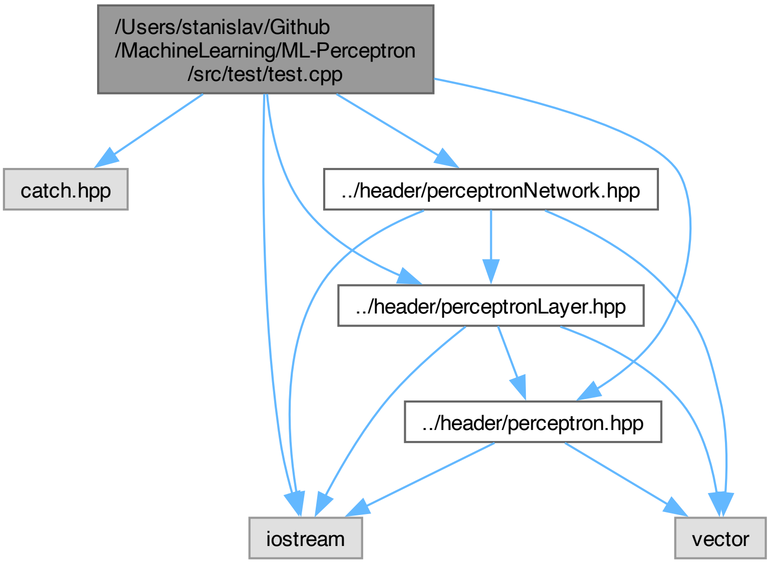
```cpp

#include "header/perceptronNetwork.hpp"
#include "header/perceptronLayer.hpp"
#include <iostream>

PerceptronNetwork::PerceptronNetwork(std::vector<PerceptronLayer> layers)
    : layers(layers) {}

std::vector<int> PerceptronNetwork::feedForward(const std::vector<int>& input) const
{
    std::vector<int> activation = input;
    // Propagate the input through each layer sequentially. Also called feedforward.
    for (const PerceptronLayer& layer : layers) {
        activation = layer.feedForward(activation);
    }
    return activation;
}

void PerceptronNetwork::__str__(int verbose) const
{
    // Print the network structure
    std::cout « "Perceptron Network Structure:" « std::endl;
    std::cout « "Number of layers: " « layers.size() « std::endl;
    // For each layer in the network print the data
    for (int i = 0; i < layers.size(); ++i)
    {
        std::cout « "Layer " « i + 1 « ": ";
        layers[i].__str__(verbose);
    }
}
```

5.13 /Users/stanislav/Github/MachineLearning/ML-Perceptron/src/test/test.cpp File Reference

In this file the test cases for the Perceptron, PerceptronLayer and PerceptronNetwork classes are defined.

```cpp
#include "catch.hpp"
#include "../header/perceptron.hpp"
#include "../header/perceptronLayer.hpp"
#include "../header/perceptronNetwork.hpp"
#include <iostream>
```

Include dependency graph for test.cpp:


/Users/stanislav/Github
/MachineLearning/ML-Perceptron
/src/test/test.cpp
catch.hpp
../header/perceptronNetwork.hpp
../header/perceptronLayer.hpp
../header/perceptron.hpp
iostream
vector

Macros

- #define CATCH_CONFIG_MAIN
- #define EPOCHS 100

Functions

- TEST_CASE ("Perceptron for INVERT Gate", "[perceptron]")

  Perceptron for INVERT Gate: Tests the perceptron's ability to learn the INVERT gate.
- TEST_CASE ("Perceptron for AND Gate", "[perceptron]")

  Perceptron for AND Gate: Tests the perceptron's ability to learn the AND gate.
- TEST_CASE ("Perceptron for OR Gate", "[perceptron]")

  Perceptron for OR Gate: Tests the perceptron's ability to learn the OR gate.
- TEST_CASE ("Perceptron for NOR Gate (3 inputs)", "[perceptron]")

  Perceptron for NOR Gate (3 inputs): Tests the perceptron's ability to learn the NOR gate with 3 inputs. The NOR gate is a digital logic gate that implements logical NOR
- TEST_CASE ("Perceptron for 3-input Majority Gate", "[perceptron]")

  Perceptron for 3-input Majority Gate: Tests the perceptron's ability to learn the 3-input Majority gate.
- TEST_CASE ("PerceptronLayer for AND and OR Gates", "[perceptronLayer]")

  PerceptronLayer for AND and OR Gates: Tests the PerceptronLayer's ability to learn the AND and OR gates. It contains two perceptrons: one for the AND gate and one for the OR gate.
- TEST_CASE ("PerceptronNetwork for the XOR gate with 2 inputs", "[perceptronNetwork]")

  PerceptronNetwork for the XOR gate with 2 inputs. This network contains two layers: inputLayer for the AND gate and one for the OR gate. outputLayer for the AND gate.
- TEST_CASE ("PerceptronNetwork for half adder", "[perceptronNetwork]")

  PerceptronNetwork for a half adder. This network contains two layers: hiddenLayer for the OR and AND gates. outputLayer for the XOR gate(sum) and the carry.

Variables

- std::vector< std::vector< int > > inputs = {{0, 0}, {0, 1}, {1, 0}, {1, 1}}

5.13.1 Detailed Description

In this file the test cases for the Perceptron, PerceptronLayer and PerceptronNetwork classes are defined.

Unit tests for the Perceptron, PerceptronLayer and PerceptronNetwork classes.

Author

Stan Merlijn

Version

0.1

Date

2025-02-12

Copyright

Copyright (c) 2025

This file contains a series of test cases to verify the functionality of the Perceptron and PerceptronLayer classes. The tests include training and prediction for various logic gates.

Test Cases:

- Perceptron for INVERT Gate: Tests the perceptron's ability to learn the INVERT gate.
- Perceptron for AND Gate: Tests the perceptron's ability to learn the AND gate.
- Perceptron for OR Gate: Tests the perceptron's ability to learn the OR gate.
- Perceptron for NOR Gate (3 inputs): Tests the perceptron's ability to learn the NOR gate with 3 inputs.
- Perceptron for 3-input Majority Gate: Tests the perceptron's ability to learn the 3-input Majority gate.
- PerceptronLayer for AND and OR Gates: Tests the PerceptronLayer's ability to learn the AND and OR gates.
- PerceptronNetwork for the XOR gate with 2 inputs.
- PerceptronNetwork for a half adder.

Note

The tests use the Catch2 framework for unit testing.

Definition in file test.cpp.

5.13.2 Macro Definition Documentation

5.13.2.1 CATCH_CONFIG_MAIN

```
#define CATCH_CONFIG_MAIN
```

Definition at line 11 of file test.cpp.

5.13.2.2 EPOCHS

```
#define EPOCHS 100
```

Definition at line 12 of file test.cpp.

5.13.3 Function Documentation

5.13.3.1 TEST_CASE() [1/8]

```
TEST_CASE (
            "Perceptron for 3-input Majority Gate" ,
            "" [perceptron])
```

Perceptron for 3-input Majority Gate: Tests the perceptron's ability to learn the 3-input Majority gate.

Definition at line 124 of file test.cpp.

5.13.3.2 TEST_CASE() [2/8]

```
TEST_CASE (
            "Perceptron for AND Gate" ,
            "" [perceptron])
```

Perceptron for AND Gate: Tests the perceptron's ability to learn the AND gate.

Definition at line 66 of file test.cpp.

5.13.3.3 TEST_CASE() [3/8]

```
TEST_CASE (
            "Perceptron for INVERT Gate" ,
            "" [perceptron])
```

Perceptron for INVERT Gate: Tests the perceptron's ability to learn the INVERT gate.

Definition at line 49 of file test.cpp.

5.13.3.4 TEST_CASE() [4/8]

```
TEST_CASE (
             "Perceptron for NOR Gate (3 inputs)" ,
             "" [perceptron])
```

Perceptron for NOR Gate (3 inputs): Tests the perceptron's ability to learn the NOR gate with 3 inputs. The NOR gate is a digital logic gate that implements logical NOR

0, 0, 0, 0, 0, 0, 0, 1

Definition at line 98 of file test.cpp.

5.13.3.5 TEST_CASE() [5/8]

```
TEST_CASE (
             "Perceptron for OR Gate" ,
             "" [perceptron])
```

Perceptron for OR Gate: Tests the perceptron's ability to learn the OR gate.

Definition at line 81 of file test.cpp.

5.13.3.6 TEST_CASE() [6/8]

```
TEST_CASE (
             "PerceptronLayer for AND and OR Gates" ,
             "" [perceptronLayer])
```

PerceptronLayer for AND and OR Gates: Tests the PerceptronLayer's ability to learn the AND and OR gates. It contains two perceptrons: one for the AND gate and one for the OR gate.

Definition at line 152 of file test.cpp.

5.13.3.7 TEST_CASE() [7/8]

```
TEST_CASE (
             "PerceptronNetwork for half adder" ,
             "" [perceptronNetwork])
```

PerceptronNetwork for a half adder. This network contains two layers: hiddenLayer for the OR and AND gates. outputLayer for the XOR gate(sum) and the carry.

Definition at line 216 of file test.cpp.

5.13.3.8 TEST_CASE() [8/8]

```
TEST_CASE (
             "PerceptronNetwork for the XOR gate with 2 inputs" ,
             "" [perceptronNetwork])
```

PerceptronNetwork for the XOR gate with 2 inputs. This network contains two layers: inputLayer for the AND gate and one for the OR gate. outputLayer for the AND gate.

Definition at line 181 of file test.cpp.

5.13.4 Variable Documentation

5.13.4.1 inputs

```
std::vector<std::vector<int> > inputs = {{0, 0}, {0, 1}, {1, 0}, {1, 1}}
```

Definition at line 42 of file test.cpp.

5.14 test.cpp

Go to the documentation of this file.

```
00001
00011 #define CATCH_CONFIG_MAIN
00012 #define EPOCHS 100
00013
00014 #include "catch.hpp"
00015 #include "../header/perceptron.hpp"
00016 #include "../header/perceptronLayer.hpp"
00017 #include "../header/perceptronNetwork.hpp"
00018
00019 #include <iostream>
00020
00040
00041 // Define the input vectors for the logic gates
00042 std::vector<std::vector<int» inputs = {{0, 0}, {0, 1}, {1, 0}, {1, 1}};
00043
00044
00049 TEST_CASE("Perceptron for INVERT Gate", "[perceptron]")
00050 {
00051     Perceptron invert_gate({0.1, 0.1}, 1, 0.1);
00052
00053     // Training data: for input 0 we expect output 1, and for input 1 we expect output 0.
00054     // The second element in the input vector is always 0.
00055     std::vector<std::vector<int» inputsInverter = {{0, 0}, {1, 0}};
00056     std::vector<int> targets = {1, 0};
00057     invert_gate.train(inputsInverter, targets, EPOCHS);
00058
00059     REQUIRE(invert_gate.predict({1, 0}) == 0);
00060     REQUIRE(invert_gate.predict({0, 1}) == 1);
00061 }
00062
00066 TEST_CASE("Perceptron for AND Gate", "[perceptron]")
00067 {
00068     Perceptron p_and({0.1, 0.1}, 1, 0.1);
00069     std::vector<int> targets = {0,0,0,1};
00070     p_and.train(inputs, targets, EPOCHS);
00071
00072     REQUIRE(p_and.predict({0, 0}) == 0);
00073     REQUIRE(p_and.predict({0, 1}) == 0);
00074     REQUIRE(p_and.predict({1, 0}) == 0);
00075     REQUIRE(p_and.predict({1, 1}) == 1);
00076 }
00077
00081 TEST_CASE("Perceptron for OR Gate", "[perceptron]")
00082 {
00083     Perceptron p_or({0.1, 0.1}, 1, 0.1);
00084     std::vector<int> targets = {0,1,1,1};
00085     p_or.train(inputs, targets, EPOCHS);
00086
00087     REQUIRE(p_or.predict({0, 0}) == 0);
00088     REQUIRE(p_or.predict({0, 1}) == 1);
00089     REQUIRE(p_or.predict({1, 0}) == 1);
00090     REQUIRE(p_or.predict({1, 1}) == 1);
00091 }
00092
00098 TEST_CASE("Perceptron for NOR Gate (3 inputs)", "[perceptron]") {
00099     // Instantiate the perceptron with three weights.
00100     Perceptron norGate({-0.1, -0.1, -0.1}, 1, 0.1);
00101
00102     // Training data for a NOR gate with 3 inputs:
00103     // Only (0,0,0) should yield 1; all others yield 0.
00104     std::vector<std::vector<int» inputsNOR = {
00105         {0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {1, 0, 0},
00106         {1, 1, 0}, {1, 0, 1}, {0, 1, 1}, {1, 1, 1}
00107     };
00108     std::vector<int> targets = {1, 0, 0, 0, 0, 0, 0, 0};
```

```
    norGate.train(inputsNOR, targets, EPOCHS);

    REQUIRE(norGate.predict({0, 0, 0}) == 1);
    REQUIRE(norGate.predict({0, 0, 1}) == 0);
    REQUIRE(norGate.predict({0, 1, 0}) == 0);
    REQUIRE(norGate.predict({1, 0, 0}) == 0);
    REQUIRE(norGate.predict({0, 1, 1}) == 0);
    REQUIRE(norGate.predict({1, 0, 1}) == 0);
    REQUIRE(norGate.predict({1, 1, 0}) == 0);
    REQUIRE(norGate.predict({1, 1, 1}) == 0);
}

TEST_CASE("Perceptron for 3-input Majority Gate", "[perceptron]") {
    // Instantiate the perceptron with three inputs. Here we choose small positive initial weights
    // and a negative bias. Adjust these parameters if necessary to speed up convergence.
    Perceptron majorityGate({0.1, 0.1, 0.1}, -0.2, 0.1);

    // Training data for a majority gate:
    // Output 1 if at least two inputs are 1, else output 0.
    std::vector<std::vector<int» inputsMajority = {
        {0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {1, 0, 0},
        {0, 1, 1}, {1, 0, 1}, {1, 1, 0}, {1, 1, 1}
    };
    std::vector<int> y = {0, 0, 0, 0, 1, 1, 1, 1};
    majorityGate.train(inputsMajority, y, EPOCHS);

    REQUIRE(majorityGate.predict({0, 0, 0}) == 0);
    REQUIRE(majorityGate.predict({0, 0, 1}) == 0);
    REQUIRE(majorityGate.predict({0, 1, 0}) == 0);
    REQUIRE(majorityGate.predict({1, 0, 0}) == 0);
    REQUIRE(majorityGate.predict({0, 1, 1}) == 1);
    REQUIRE(majorityGate.predict({1, 0, 1}) == 1);
    REQUIRE(majorityGate.predict({1, 1, 0}) == 1);
    REQUIRE(majorityGate.predict({1, 1, 1}) == 1);
}

TEST_CASE("PerceptronLayer for AND and OR Gates", "[perceptronLayer]") {
    // Training data common to both gates:
    Perceptron p_or({0.1, 0.1}, 1, 0.1);
    Perceptron p_and({0.1, 0.1}, 1, 0.1);

    // Training the OR and AND gates.
    p_or.train(inputs, {0, 1, 1, 1}, EPOCHS);
    p_and.train(inputs, {0, 0, 0, 1}, EPOCHS);

    // Create a layer with two neurons (2 inputs) for the AND gate and a learning rate of 0.1.
    // Train the layer with the AND gate targets and OR gate targets.
    PerceptronLayer andLayer({p_and, p_or});

    // Define expected outputs for the AND gate and OR gate.
    std::vector<int> out00 = {0, 0};
    std::vector<int> out01 = {0, 1};
    std::vector<int> out11 = {1, 1};

    REQUIRE(andLayer.feedForward({0, 0}) == out00);
    REQUIRE(andLayer.feedForward({0, 1}) == out01);
    REQUIRE(andLayer.feedForward({1, 0}) == out01);
    REQUIRE(andLayer.feedForward({1, 1}) == out11);
}

TEST_CASE("PerceptronNetwork for the XOR gate with 2 inputs", "[perceptronNetwork]") {
    // Create a network with two layers: one for the AND gate and one for the OR gate.
    // OR and NAND gates for the input layer
    Perceptron p_or({0.1, 0.1}, 1, 0.1);
    Perceptron p_nand({0.1, 0.1}, 1, 0.1);
    Perceptron p_and({0.1, 0.1}, 1, 0.1);

    // Training The gates
    p_or.train(inputs,  {0, 1, 1, 1} , EPOCHS);
    p_nand.train(inputs, {1, 1, 1, 0}, EPOCHS);
    p_and.train(inputs,  {0, 0, 0, 1} , EPOCHS);

    PerceptronLayer inputLayer({p_or, p_nand});
    PerceptronLayer outputLayer({p_and});

    PerceptronNetwork xor_network({inputLayer, outputLayer});

    // Define expected outputs for the XOR gate.
    std::vector<int> out00 = {0};
    std::vector<int> out01 = {1};
    std::vector<int> out10 = {1};
    std::vector<int> out11 = {0};

    // Verify network’s predictions for the XOR gate.
    REQUIRE(xor_network.feedForward({0, 0}) == out00);
    REQUIRE(xor_network.feedForward({0, 1}) == out01);
    REQUIRE(xor_network.feedForward({1, 0}) == out10);
    REQUIRE(xor_network.feedForward({1, 1}) == out11);
}

TEST_CASE("PerceptronNetwork for half adder", "[perceptronNetwork]")
{
    // Hidden layer: compute OR and AND
    Perceptron n_or({0.1, 0.1}, 0.1, 0.1);
    Perceptron n_and({0.1, 0.1}, 0.1, 0.1);

    n_or.train(inputs, {0, 1, 1, 1}, EPOCHS);
    n_and.train(inputs, {0, 0, 0, 1}, EPOCHS);

    PerceptronLayer hiddenLayer({n_or, n_and});

    // Output layer: compute XOR (for sum) and carry
    // XOR = OR - AND; Perceptron with weights {1, -1} and bias -0.5
    // x1 = 0, x2 = 0: 1 * 0 + -1 * 0 - 0.5 = -0.5
    // x1 = 1, x2 = 0: 1 * 1 + -1 * 0 - 0.5 = 0.5
    // x1 = 1, x2 = 1: 1 * 1 + -1 * 1 - 0.5 = -0.5

    Perceptron n_xor({0.1, 0.1}, 0.1, 0.1);
    Perceptron n_carry({0.1, 0.1}, 0.1, 0.1);

    n_xor.train({{0, 0}, {1, 0}, {1, 1}}, {0, 1, 0}, EPOCHS);
    n_carry.train(inputs, {0, 0, 0, 1}, EPOCHS);

    PerceptronLayer outputLayer({n_xor, n_carry});

    PerceptronNetwork halfAdder({hiddenLayer, outputLayer});

    // Test cases for half adder: {Sum, Carry}
    REQUIRE(halfAdder.feedForward({0, 0}) == std::vector<int>{0, 0});
    REQUIRE(halfAdder.feedForward({0, 1}) == std::vector<int>{1, 0});
    REQUIRE(halfAdder.feedForward({1, 0}) == std::vector<int>{1, 0});
    REQUIRE(halfAdder.feedForward({1, 1}) == std::vector<int>{0, 1});
}
```

Index