Neural Network Visualization

1.0.0

Generated by Doxygen 1.12.0

Chapter 1

Class Index

1.1 Class List

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

ActivationFunction
Class for an activation function . . . . ??

CostFunction
Class for a cost function . . . . ??

CSVReader . . . . ??

EvaluationFunctions . . . . ??

Layer
Class for a layer in a neural network . . . . ??

Network
Class for a neural network . . . . ??

VisualizeClassification
This is a class used to visualize the process of neural networks learning to classify points . . . ??

Chapter 2

File Index

2.1 File List

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

include/ActivationFunction.h . . . . ??
include/CostFunction.h . . . . ??
include/CSVReader.h . . . . ??
include/EvaluationFunctions.h . . . . ??
include/Layer.h . . . . ??
include/Network.h . . . . ??
include/VisualizeClassification.h . . . . ??

Chapter 3

Class Documentation

3.1 ActivationFunction Class Reference

Class for an activation function.

```
#include <ActivationFunction.h>
```

Public Member Functions

- **ActivationFunction** ()

  Default constructor for ActivationFunction.
- ActivationFunction (float(∗activationFunction)(float x), float(∗derivative)(float x))

  Constructor for ActivationFunction.

Public Attributes

- float(∗ activation)(float x)

  Returns the activation of a certain value.
- float(∗ derivative)(float x)

  Returns the derivative of the activation function for a certain value.

3.1.1 Detailed Description

Class for an activation function.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ActivationFunction()

```
ActivationFunction::ActivationFunction (
            float(* activation )(float x),
            float(* derivative )(float x))
```

Constructor for ActivationFunction.

Parameters

activation	The activation function
derivative	The derivative of the activation function

3.1.3 Member Data Documentation

3.1.3.1 activation

```
float(* ActivationFunction::activation) (float x)
```

Returns the activation of a certain value.

Parameters

x	The value to be activated

Returns

: The activated value

3.1.3.2 derivative

```
float(* ActivationFunction::derivative) (float x)
```

Returns the derivative of the activation function for a certain value.

Parameters

x	The value to be activated

Returns

: The derivative of the activated value

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

- include/ActivationFunction.h
- src/ActivationFunction.cpp

3.2 CostFunction Class Reference

Class for a cost function.

```
#include <CostFunction.h>
```

Public Member Functions

- **CostFunction** ()

  Default constructor for CostFunction.
- CostFunction (float(∗cost)(float output, float expectedOutput), float(∗derivative)(float output, float expected↩ Output))

  Constructor for CostFunction.

Public Attributes

- float(∗ cost)(float output, float expectedOutput)

  Returns the cost of a certain output and expected output.
- float(∗ derivative)(float output, float expectedOutput)

  Returns the derivative of the cost function for a certain output and expected output.

3.2.1 Detailed Description

Class for a cost function.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 CostFunction()

```
CostFunction::CostFunction (
              float(∗ cost )(float output, float expectedOutput),
              float(∗ derivative )(float output, float expectedOutput))
```

Constructor for CostFunction.

Parameters

cost	The cost function
derivative	The derivative of the cost function

3.2.3 Member Data Documentation

3.2.3.1 cost

```
float(∗ CostFunction::cost) (float output, float expectedOutput)
```

Returns the cost of a certain output and expected output.

Parameters

output	The output of the network
expectedOutput	The expected output of the network

3.2.3.2 derivative

```
float(∗ CostFunction::derivative) (float output, float expectedOutput)
```

Returns the derivative of the cost function for a certain output and expected output.

Parameters

output	The output of the network
expectedOutput	The expected output of the network

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

- include/CostFunction.h
- src/CostFunction.cpp

3.3 CSVReader Class Reference

Static Public Member Functions

- static std::vector< std::vector< float > > readfloatRows (std::string fileName, int startColumn, int endColumn, int startRow, int endRow)

  Reads a CSV file and returns a vector of vectors of floats.
- static std::vector< float > readfloatColumn (std::string fileName, int columnIndex, int startRow, int endRow)

  Reads a CSV file and returns a vector of ints.
- static std::vector< std::vector< float > > normalize (std::vector< std::vector< float > > data, float max)

  Normalizes a vector of vectors of floats.
- static std::vector< std::vector< float > > vectorizeOutputs (std::vector< float > outputs, int numClasses)

  Vectorizes a vector of ints.

3.3.1 Member Function Documentation

3.3.1.1 normalize()

```
vector< vector< float > > CSVReader::normalize (
            std::vector< std::vector< float > > data,
            float max)  [static]
```

Normalizes a vector of vectors of floats.

Parameters

data	The data to normalize
max	The maximum value to normalize to

Returns

: The normalized data

3.3.1.2 readfloatColumn()

```
vector< float > CSVReader::readfloatColumn (
            std::string fileName,
            int columnIndex,
            int startRow,
            int endRow)  [static]
```

Reads a CSV file and returns a vector of ints.

Parameters

fileName	The name of the file to read
columnIndex	The column to read from

Returns

: A vector of ints

3.3.1.3 readfloatRows()

```
vector< vector< float > > CSVReader::readfloatRows (
            std::string fileName,
            int startColumn,
            int endColumn,
            int startRow,
            int endRow)  [static]
```

Reads a CSV file and returns a vector of vectors of floats.

Parameters

fileName	The name of the file to read
startColumn	The column to start reading from
endColumn	The column to stop reading at

Returns

: A vector of vectors of floats

3.3.1.4 vectorizeOutputs()

```
vector< vector< float > > CSVReader::vectorizeOutputs (
            std::vector< float > outputs,
            int numClasses)  [static]
```

Vectorizes a vector of ints.

Parameters

outputs	The outputs to vectorize
numClasses	The number of classes

Returns

: The vectorized outputs

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

- include/CSVReader.h
- src/CSVReader.cpp

3.4 EvaluationFunctions Class Reference

Static Public Member Functions

- static int **three_linear_sections** (float x, float y)

  Divides the data points into three linear sections on the screen Recommended MAX_POINT_VALUE: 5-10 Number of classes: 3.
- static int **four_squares** (float x, float y)

  Divides the data points into four square sections Recommended MAX_POINT_VALUE: 10 Number of classes: 4.
- static int **cubic_function** (float x, float y)

  Seperates data via a cubic function Recommended MAX_POINT_VALUE: 1 Number of classes: 2.
- static int **quadratic_function** (float x, float y)

  Seperates the data points via a quadtaric function Recommended MAX_POINT_VALUE: 1 Number of classes: 2.
- static int **circle_function** (float x, float y)

  Divides the data points based on whether they are inside a circle Recommended MAX_POINT_VALUE: 10 Number of classes: 2.
- static int **three_class_circle** (float x, float y)

  Seperates the data points into 3 classes based on distance from the center Recommended MAX_POINT_VALUE: 10 Number of classes: 3.
- static int **four_class_circle** (float x, float y)

  Seperates the data points into 4 classes based on distance from the center Recommended MAX_POINT_VALUE: 10 Number of classes: 4.
- static int **tanh_function** (float x, float y)

  Seperates the data points via a tanh function Recommended MAX_POINT_VALUE: 10 Number of classes: 2.

Static Public Attributes

- static const std::map< std::string, EvalFunctionPtr > function_map

  Map of evaluation function names to function pointers.
- static const std::map< std::string, int > num_classes_map

  Map of evaluation function names to number of classes of data points.

3.4.1 Member Data Documentation

3.4.1.1 function_map

```
const std::map< std::string, EvalFunctionPtr > EvaluationFunctions::function_map  [static]
```

Initial value:

```
= {
    {"linear", EvaluationFunctions::three_linear_sections},
    {"4_squares", EvaluationFunctions::four_squares},
    {"cubic", EvaluationFunctions::cubic_function},
    {"quadratic", EvaluationFunctions::quadratic_function},
    {"circle", EvaluationFunctions::circle_function},
    {"3circles", EvaluationFunctions::three_class_circle},
    {"4circles", EvaluationFunctions::four_class_circle},
    {"tanh", EvaluationFunctions::tanh_function},
}
```

Map of evaluation function names to function pointers.

3.4.1.2 num_classes_map

```
const std::map< std::string, int > EvaluationFunctions::num_classes_map  [static]
```

Initial value:

```
= {
    {"3linear", 3},
    {"4squares", 4},
    {"cubic", 2},
    {"quadratic", 2},
    {"circle", 2},
    {"3circles", 3},
    {"4circles", 4},
    {"tanh", 2},
}
```

Map of evaluation function names to number of classes of data points.

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

- include/EvaluationFunctions.h
- src/EvaluationFunctions.cpp

3.5 Layer Class Reference

Class for a layer in a neural network.

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

Public Member Functions

- Layer (int num_neurons, int num_neuron)

  Constructor for class layer.
- Layer (int num_neurons, int num_neuron, ActivationFunction activationFunction)

  Constructor for class layer.
- vector< float > calculateOutputs (vector< float > inputs)

  Calculates the outputs of the layer and stores the inputs, outputs, and activations.
- string toString ()

  Returns a string representation of the layer.
- size_t getNumInputs ()

  Returns the number of inputs to the layer.
- size_t getNumOutputs ()

  Returns the number of outputs from the layer.
- float getWeight (int neuronIndex, int inputIndex)

  Returns the weight of a neuron.
- float getBias (int neuronIndex)

  Returns the bias of a neuron.
- void setNeuronWeight (int neuronIndex, int inputIndex, float weight)

  Sets the weight of a neuron at a specified index.
- void setNeuronBias (int neuronIndex, float bias)

  Sets the bias of a neuron at a specified index.
- void calculateOutputLayerPartialDerivatives (CostFunction costFunction, vector< float > expectedOutputs)

  Calculates the partial derivatives of the weights and biases if this is an output Stores the weight derivatives in weight↩ Derivatives and derivatves needed for future backpropogation in derivatvesCostRespectToOutputs.

- void calculateHiddenLayerPartialDerivatives (Layer nextLayer)

  Calculates the partial derivatives of the weights and biases if this is a hidden layer Stores the weight derivatives in weightDerivatives and derivatves needed for future backpropogation in derivatvesCostRespectToOutputs.
- void calculateBiasPartialDerivatives (Layer nextLayer)

  Calculates the partial derivatives of the biases Stores the bias derivatives in biasDerivatives.
- vector< vector< float > > **getWeightDerivatives** ()
- vector< float > **getBiasDerivatives** ()
- void **resetDerivatives** ()

  Resets the derivatives of the weights and biases to 0.

3.5.1 Detailed Description

Class for a layer in a neural network.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Layer() [1/2]

```
Layer::Layer (
              int num_inputs,
              int num_outputs)
```

Constructor for class layer.

Parameters

num_inputs	The number of inputs to the layer
num_outputs	The number of outputs from the layer Will use the identity activation function

3.5.2.2 Layer() [2/2]

```
Layer::Layer (
              int num_inputs,
              int num_outputs,
              ActivationFunction activationFunction)
```

Constructor for class layer.

Parameters

num_inputs	The number of inputs to the layer
num_outputs	The number of outputs from the layer
activationFunction	The activation function of the layer

3.5.3 Member Function Documentation

3.5.3.1 calculateBiasPartialDerivatives()

```
void Layer::calculateBiasPartialDerivatives (
              Layer nextLayer)
```

Calculates the partial derivatives of the biases Stores the bias derivatives in biasDerivatives.

Parameters

nextLayer	The next layer in the network

3.5.3.2 calculateHiddenLayerPartialDerivatives()

```
void Layer::calculateHiddenLayerPartialDerivatives (
             Layer nextLayer)
```

Calculates the partial derivatives of the weights and biases if this is a hidden layer Stores the weight derivatives in weightDerivatives and derivatves needed for future backpropogation in derivatvesCostRespectToOutputs.

Parameters

nextLayer	The next layer in the network

3.5.3.3 calculateOutputLayerPartialDerivatives()

```
void Layer::calculateOutputLayerPartialDerivatives (
             CostFunction costFunction,
             vector< float > expectedOutputs)
```

Calculates the partial derivatives of the weights and biases if this is an output Stores the weight derivatives in weightDerivatives and derivatves needed for future backpropogation in derivatvesCostRespectToOutputs.

Parameters

costFunction	The cost function of the network
expectedOutputs	The expected outputs for the layer

3.5.3.4 calculateOutputs()

```
vector< float > Layer::calculateOutputs (
             vector< float > inputs)
```

Calculates the outputs of the layer and stores the inputs, outputs, and activations.

Parameters

inputs	The inputs to the layer

Returns

: The activations of the outputs of the layer

3.5.3.5 getBias()

```
float Layer::getBias (
             int neuronIndex)
```

Returns the bias of a neuron.

Parameters

neuronIndex	The index of the neuron

Returns

: The bias of the neuron

3.5.3.6 getNumInputs()

```
size_t Layer::getNumInputs ()
```

Returns the number of inputs to the layer.

Returns

: The number of inputs to the layer

3.5.3.7 getNumOutputs()

```
size_t Layer::getNumOutputs ()
```

Returns the number of outputs from the layer.

Returns

: The number of outputs from the layer

3.5.3.8 getWeight()

```
float Layer::getWeight (
             int neuronIndex,
             int inputIndex)
```

Returns the weight of a neuron.

Parameters

neuronIndex	The index of the neuron
inputIndex	The index of the weight

Returns

: The specified weight of the neuron

3.5.3.9 setNeuronBias()

```
void Layer::setNeuronBias (
             int neuronIndex,
             float bias)
```

Sets the bias of a neuron at a specified index.

Parameters

neuronIndex	The index of the neuron
bias	The new bias of the neuron

3.5.3.10 setNeuronWeight()

```
void Layer::setNeuronWeight (
             int neuronIndex,
             int inputIndex,
             float weight)
```

Sets the weight of a neuron at a specified index.

Parameters

neuronIndex	The index of the neuron
inputIndex	The index of the weight
weight	The new weight of the neuron

3.5.3.11 toString()

```
string Layer::toString ()
```

Returns a string representation of the layer.

Returns

: A string representation of the layer

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

- include/Layer.h
- src/Layer.cpp

3.6 Network Class Reference

Class for a neural network.

```
#include <Network.h>
```

Public Member Functions

- **Network** ()

  Default constructor for Network.
- Network (Layer layer)

  Constructor for class Network.
- Network (vector< Layer > layers, float learn_rate=0.1, int epochs_per_decay=5, int batch_size=32, CostFunction costFunction=errorSquared)

  Constructor for class Network.
- int classify (vector< float > inputs)

  Classifies a given input to the network.
- string toString ()

  Gets the string representation of the network.
- vector< float > getOutput (vector< float > inputs)

  Gets the output of the network for a given input.
- vector< vector< float > > getOutputs (vector< vector< float > > inputs)

  Gets the outputs of the network for a given set of inputs.
- float cost (vector< float > inputs, vector< float > expected_output)

  Calculates the cost of the network for a given output and expected output.
- float averageCost (vector< vector< float > > inputs, vector< vector< float > > expected_outputs)

  Calculates the average cost of the network for a given set of outputs and expected outputs.
- float accuracy (vector< vector< float > > inputs, vector< int > expected_integer_outputs)

  Calculates the accuracy of the network for a given set of inputs and expected outputs.
- float accuracy (vector< vector< float > > inputs, vector< vector< float > > expected_outputs)

  Calculates the accuracy of the network for a given set of inputs and expected outputs.
- void updateDerivatives (vector< float > input, vector< float > expected_output)

  Given an input and output, calculate how much to adjust each weight and bias in the network.
- void quickLearn (vector< vector< float > > inputs, vector< vector< float > > expected_outputs)

  Given a set of inputs and expected outputs, using backpropogation to adjust the weights and biases of the network.
- void applyDerivatives (int batch_size)

  Applies the previously calculated derivatives to the weights and biases of the network.
- void learnWithBatchSize (vector< vector< float > > inputs, vector< vector< float > > expected_outputs)

  Divide data into batches then feed into quickLearn for backpropogation.
- void train (vector< vector< float > > inputs, vector< vector< float > > expected_outputs, int epochs)

  Trains the network on a given set of inputs and expected outputs for a given number of epochs.
- vector< float > **intToVector** (int num)

  Converts an integer to a vector of length num_outputs with a 1 in the index of the integer.
- int getEpoch ()

  Gets the batch size of the network.

3.6.1 Detailed Description

Class for a neural network.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 Network() [1/2]

```
Network::Network (
            Layer layer)
```

Constructor for class Network.

Parameters

layer	The layer to add to the network

3.6.2.2 Network() [2/2]

```
Network::Network (
            vector< Layer > layers,
            float learn_rate = 0.1,
            int epochs_per_decay = 5,
            int batch_size = 32,
            CostFunction costFunction = errorSquared)
```

Constructor for class Network.

Parameters

layers	The layers to add to the network
learn_rate	The learning rate of the network
epochs_per_decay	The number of epochs before the learning rate is halved
batch_size	The batch size for learning
costFunction	The cost function of the network

3.6.3 Member Function Documentation

3.6.3.1 accuracy() [1/2]

```
float Network::accuracy (
            vector< vector< float > > inputs,
            vector< int > expected_integer_outputs)
```

Calculates the accuracy of the network for a given set of inputs and expected outputs.

Parameters

inputs	The inputs to the network
expected_integer_outputs	The expected outputs of the network

Returns

: The percentage of data points classified correctly

3.6.3.2 accuracy() [2/2]

```
float Network::accuracy (
            vector< vector< float > > inputs,
            vector< vector< float > > expected_outputs)
```

Calculates the accuracy of the network for a given set of inputs and expected outputs.

Parameters

inputs	The inputs to the network
expected_outputs	The expected outputs of the network

Returns

: The percentage of data points classified correctly

3.6.3.3 applyDerivatives()

```
void Network::applyDerivatives (
            int batch_size)
```

Applies the previously calculated derivatives to the weights and biases of the network.

Parameters

batch_size	how many points the derivatives are calculated from

3.6.3.4 averageCost()

```
float Network::averageCost (
            vector< vector< float > > outputs,
            vector< vector< float > > expected_outputs)
```

Calculates the average cost of the network for a given set of outputs and expected outputs.

Parameters

outputs	The outputs of the network
expected_outputs	The expected outputs of the network

Returns

: The average cost of the network for the given outputs and expected outputs

3.6.3.5 classify()

```
int Network::classify (
            vector< float > input)
```

Classifies a given input to the network.

Parameters

input	The input to the network

Returns

: The integer classification of the inputs

3.6.3.6 cost()

```
float Network::cost (
            vector< float > inputs,
            vector< float > expected_output)
```

Calculates the cost of the network for a given output and expected output.

Parameters

output	The output of the network
expected_output	The expected output of the network

Returns

: The cost of the network

3.6.3.7 getEpoch()

```
int Network::getEpoch ()
```

Gets the batch size of the network.

Returns

: The batch size of the network

3.6.3.8 getOutput()

```
vector< float > Network::getOutput (
            vector< float > inputs)
```

Gets the output of the network for a given input.

Parameters

inputs	The inputs to the network

Returns

: The output of the network

3.6.3.9 getOutputs()

```
vector< vector< float > > Network::getOutputs (
            vector< vector< float > > inputs)
```

Gets the outputs of the network for a given set of inputs.

Parameters

inputs	The inputs to the network

Returns

: The outputs of the network

3.6.3.10 learnWithBatchSize()

```
void Network::learnWithBatchSize (
            vector< vector< float > > inputs,
            vector< vector< float > > expected_outputs)
```

Divide data into batches then feed into quickLearn for backpropogation.

Parameters

inputs	The inputs to the network
expected_outputs	The expected outputs of the network

3.6.3.11 quickLearn()

```
void Network::quickLearn (
            vector< vector< float > > inputs,
            vector< vector< float > > expected_output)
```

Given a set of inputs and expected outputs, using backpropogation to adjust the weights and biases of the network.

Parameters

input	The input to the network
expected_output	The expected output of the network

3.6.3.12 toString()

```
string Network::toString ()
```

Gets the string representation of the network.

Returns

: a string representation of the network

3.6.3.13 train()

```
void Network::train (
            vector< vector< float > > inputs,
            vector< vector< float > > expected_outputs,
            int epochs)
```

Trains the network on a given set of inputs and expected outputs for a given number of epochs.

Parameters

inputs	The inputs to the network
expected_outputs	The expected outputs of the network
epochs	The number of epochs to train the network for

3.6.3.14 updateDerivatives()

```
void Network::updateDerivatives (
            vector< float > input,
            vector< float > expected_output)
```

Given an input and output, calculate how much to adjust each weight and bias in the network.

Parameters

input	The input to the network
expected_output	The expected output of the network

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

- include/Network.h
- src/Network.cpp

3.7 VisualizeClassification Class Reference

This is a class used to visualize the process of neural networks learning to classify points.

```
#include <VisualizeClassification.h>
```

Public Member Functions

- **VisualizeClassification** (int(∗eval_function)(float, float), Network network)

  Constructor.
- VisualizeClassification (int(∗eval_function)(float, float), Network network, int num_points, int screen_size, float max_value=10)

  Constructor for class VisualizeClassification.
- void **initializeTransparentColors** ()

  Initializes the transparent colors to be used later.
- void **generateRandomData** ()

  Generates random data points for the class.
- void **generateOutputs** ()

  Generates the outputs for the data points.
- void **classifyPointsTransparently** ()

  Classifies the points with a transparent mask over the screen.
- void **drawPoints** ()

  Draws the points to the screen.
- void **runMainLoop** ()

  Runs the main loop of the visualization.
- void **showNetworkInfo** ()

  Shows the loss and number of epochs on the screen.

3.7.1 Detailed Description

This is a class used to visualize the process of neural networks learning to classify points.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 VisualizeClassification()

```
VisualizeClassification::VisualizeClassification (
            int(* eval_function )(float, float),
            Network network,
            int num_points,
            int screen_size,
            float max_value = 10)
```

Constructor for class VisualizeClassification.

Parameters

eval_function	function to cassify points
network	network learning classify points
num_points	number of data points to create and classify
screen_size	size of the screen to draw to
max_value	maximum value of x and y for data points

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

- include/VisualizeClassification.h
- src/VisualizeClassification.cpp

Chapter 4

File Documentation

4.1 ActivationFunction.h

```
#ifndef ACTIVATION_FUNCTION_H
#define ACTIVATION_FUNCTION_H

class ActivationFunction {
public:
    float (*activation)(float x);
    float (*derivative)(float x);
    ActivationFunction();
    ActivationFunction(float (*activationFunction)(float x), float (*derivative)(float x));
};

extern ActivationFunction sigmoid;
extern ActivationFunction tanH;
extern ActivationFunction relu;
extern ActivationFunction identity;

#endif // ACTIVATION_FUNCTION_H
```

4.2 CostFunction.h

```
#ifndef COST_FUNCTION_H
#define COST_FUNCTION_H

#include <vector>

using namespace std;

class CostFunction {
public:
    float (*cost)(float output, float expectedOutput);

    float (*derivative)(float output, float expectedOutput);
    CostFunction();
    CostFunction(float (*cost)(float output, float expectedOutput), float (*derivative)(float output, float expectedOutput));
};

extern CostFunction errorSquared;

#endif // COST_FUNCTION_H
```

4.3 CSVReader.h

```
#ifndef CSVREADER_H
#define CSVREADER_H


#include <vector>
#include <string>
```

```

class CSVReader{
public:
    static std::vector<std::vector<float» readfloatRows(std::string fileName, int startColumn, int endColumn, int startRow, int endRow);
    static std::vector<float> readfloatColumn(std::string fileName, int columnIndex, int startRow, int endRow);
    static std::vector<std::vector<float» normalize(std::vector<std::vector<float» data, float max);
    static std::vector<std::vector<float» vectorizeOutputs(std::vector<float> outputs, int numClasses);
};

#endif // CSVREADER_H

```

4.4 EvaluationFunctions.h

```
#ifndef EVALUATION_FUNCTIONS_H
#define EVALUATION_FUNCTIONS_H

#include <map>
#include <string>

using EvalFunctionPtr = int (*)(float, float);

class EvaluationFunctions {
    public:
        static int three_linear_sections(float x, float y);
        static int four_squares(float x, float y);
        static int cubic_function(float x, float y);
        static int quadratic_function(float x, float y);
        static int circle_function(float x, float y);
        static int three_class_circle(float x, float y);
        static int four_class_circle(float x, float y);
        static int tanh_function(float x, float y);
        const static std::map<std::string, EvalFunctionPtr> function_map;
        const static std::map<std::string, int> num_classes_map;
};


#endif // EVALUATION_FUNCTIONS_H
```

4.5 Layer.h

```
#ifndef LAYER_H
#define LAYER_H

#include <vector>
#include <string>
#include "ActivationFunction.h"
#include "CostFunction.h"

using namespace std;

class Layer {

private:
    size_t num_inputs;                     // Number of inputs to the layer
    size_t num_outputs;                    // Number of outputs from the layer
    vector<vector<float» weights;     // Stores the weights with indexes [output][input]
    vector<float> biases;              // Stores the biases for each neuron


    static bool seeded;
    ActivationFunction activationFunction;

    // Variables for backpropagation
    vector<float> previousInputs;
    vector<float> previousOutputs;
    vector<float> previousActivations;
    vector<float> derivatvesCostRespectToOutputs;
    vector<vector<float» weightDerivatives;
    vector<float> biasDerivatives;


public:
    Layer(int num_neurons, int num_neuron);     // Constructor
    Layer(int num_neurons, int num_neuron, ActivationFunction activationFunction);
    vector<float> calculateOutputs(vector<float> inputs);  // Calculates the outputs of the layer
    string toString();
    size_t getNumInputs();
    size_t getNumOutputs();
    float getWeight(int neuronIndex, int inputIndex);
    float getBias(int neuronIndex);
    void setNeuronWeight(int neuronIndex, int inputIndex, float weight);
    void setNeuronBias(int neuronIndex, float bias);
    void calculateOutputLayerPartialDerivatives(CostFunction costFunction, vector<float> expectedOutputs);
    void calculateHiddenLayerPartialDerivatives(Layer nextLayer);
    void calculateBiasPartialDerivatives(Layer nextLayer);
    vector<vector<float» getWeightDerivatives();
    vector<float> getBiasDerivatives();
    void resetDerivatives();
};

#endif // LAYER_H
```

4.6 Network.h

```
#ifndef NETWORK_H
#define NETWORK_H

#include <random>
#include <algorithm>
#include <string>

#include "Layer.h"
#include "CostFunction.h"

using namespace std;

class Network{
private:
    vector<Layer> layers;      // Stores the layers of the network
    size_t num_inputs;         // Number of inputs to the network
    size_t num_outputs;        // Number of outputs from the network
    size_t num_layers;         // Number of layers in the network
    float learn_rate;      // Learning rate of the network
    int batch_size;          // Batch size for learning
    int epoch;               // Epochs run of the network
    int epoch_decay_rate;    // Number of epochs before the learning rate is halved
    CostFunction costFunction;  // Cost function of the network
    std::mt19937 rng;

public:
    Network();
    Network(Layer layer);
    Network(vector<Layer> layers, float learn_rate = 0.1, int epochs_per_decay = 5, int batch_size = 32, CostFunction costFunction = errorSquared);
    int classify(vector<float> inputs);
    string toString();
    vector<float> getOutput(vector<float> inputs);
    vector<vector<float» getOutputs(vector<vector<float» inputs);
    float cost(vector<float> inputs, vector<float> expected_output);
    float averageCost(vector<vector<float» inputs, vector<vector<float» expected_outputs);
    float accuracy(vector<vector<float» inputs, vector<int> expected_integer_outputs);
    float accuracy(vector<vector<float» inputs, vector<vector<float» expected_outputs);
    void updateDerivatives(vector<float> input, vector<float> expected_output);
    void quickLearn(vector<vector<float» inputs, vector<vector<float» expected_outputs);
    void applyDerivatives(int batch_size);
    void learnWithBatchSize(vector<vector<float» inputs, vector<vector<float» expected_outputs);
    void train(vector<vector<float» inputs, vector<vector<float» expected_outputs, int epochs);
    vector<float> intToVector(int num);
    int getEpoch();
};

#endif // NETWORK_H
```

4.7 VisualizeClassification.h

```
#ifndef VISUALIZE_CLASSIFICATION_H
#define VISUALIZE_CLASSIFICATION_H

#include <SFML/Graphics.hpp>
#include "Network.h"
#include <chrono>

class VisualizeClassification {
```

```
private:
    int num_points;             // number of data points
    std::vector<std::vector<float» inputs;                // input values of data points
    std::vector<int> outputs;                               // outputs of data points from eval function
    std::vector<std::vector<float» vectorizedOutputs;     // outputs of data points from eval function
    float max_value;      // maximum value of x and y
    // Colors for each output
    const sf::Color colors[8] = {sf::Color::Green, sf::Color::Red,
                                 sf::Color::Blue, sf::Color::Yellow,
                                 sf::Color::Magenta, sf::Color::Cyan,
                                 sf::Color::White, sf::Color::Black};
    sf::Color transparentColors[8];     // transparent versions of the colors

    float screen_size_x;       // size of the screen in x direction
    float screen_size_y;       // size of the screen in y direction
    sf::RenderWindow window;    // window to draw to
    int (*eval_function)(float, float);   // function to generate outputs
    Network network;   // network to generate outputs
    std::chrono::time_point<std::chrono::system_clock> start_time;  // time when the visualization started

public:
    VisualizeClassification(int (*eval_function)(float, float), Network network);

    VisualizeClassification(int (*eval_function)(float, float), Network network, int num_points, int screen_size, float max_value = 10);
    void initializeTransparentColors();
    void generateRandomData();
    void generateOutputs();
    void classifyPointsTransparently();
    void drawPoints();
    void runMainLoop();
    void showNetworkInfo();
};

#endif // VISUALIZE_CLASSIFICATION_H
```