دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

4/6/2020

Report Homework 3

Advanced Programming

Mohammad Javad Amin (9523008)
AMIRKABIR UNIVERSITY OF TECHNOLOGY

به نام خدا

هدف پروژه ؛ طراحی یک شبکه عصبی دو لایه است که از روی ورودی های داده شده به آن وزن هایی مناسب پیدا کند و با استفاده از وزن ها خروجی را پیش بینی کند. برای طراحی چنین شبکه ایی نیاز به تعریف چند class است .

اولین class matrix است که در پروژه قبل تعریف و الگوریتم های آن در گزارش کار قبلی توضیح داده شده است و اینجا ما فقط چند member function به آن اضافه کرده و آن‌ها را توضیح میدهیم. در زیرکل class matrix را مشاهده میکند.

```
class Matrix
{
private:                                            //member variables
    std::vector<std::vector<double>> Data{};        // elements of matrix
    std::array<size_t, 2> size{};                   //number of rows and columns
public:                                             //member functions
    Matrix(std::vector<std::vector<double>> data);  // constructors
    Matrix(size_t, size_t, bool ones = true);
    Matrix(const Matrix&);                          // copy constructor
    Matrix(std::unique_ptr<std::unique_ptr<double[]>[]>& data, size_t, size_t);
    Matrix();                                       // defult construactor
                                        //member variables
    std::array<size_t, 2> getSize();    //size of the matrix
    double det();                       //determinant of the matrix
    Matrix inv();                       // inverse of the matrix
    Matrix T();                         //transpose of the matrix
    void show();                        //displays the matrix
    Matrix delCol(size_t);              //removes the given (i-th) column of the matrix
    Matrix remain_matrix_until_col(size_t); //removes the given (i-th) columns and and after that of the matrix
    Matrix col(size_t);                 //returns the given (i-th) column of the matrix
    Matrix remain_matrix_after_col(size_t); //removes columns until (i-th) columns of the matrix
    Matrix re_row(size_t);              //give the i-th row of matrix
    Matrix del_last_row();              //remove last row
    Matrix attach(const Matrix&);       //attac two matrix
    void save(const char*);             //saves the matrix as a csv file
    void load(const char*);             //This function loads a csv file into a matrix.
                                        //operator
    Matrix operator+(const Matrix&);    // add matrix
    Matrix operator-(const Matrix&);    //subtraction matrixs
    Matrix operator*(const Matrix&);    //multiply matrix
    std::vector <double>& operator[](size_t);
};
```

حال به بررسی function هایی که تازه به این class اضافه شده است میروم.

۱) remain_matrix_until_col :کار این تابع حذف ستون های بعد از آن شماره ستونی است که به عنوان ورودی به آن داده ایم و برگرداندن بقیه ماتریس به عنوان خروجی. پیاده سازی این ماتریس به صورت زیر است.

```
Matrix Matrix::remain_matrix_until_col(size_t k)          //removes the given (k+1-th) columns and and after that of the matrix
{
    std::vector<std::vector<double>> remain_vector{};     // to store remain elements

    for (size_t i = 0; i < size[0]; i++)                  //row
    {
        std::vector<double> remain_vector_row{};          //to store a row of element

        for (size_t j = 0; j <= k; j++)                   // save element in  zero to i-th column
            remain_vector_row.push_back(Data[i][j]);      // store  remaining element
        remain_vector.push_back(remain_vector_row);       // store a row of element
    }

    return Matrix{ remain_vector };                       // return remaining matrix
}
```

۲) remain_matrix_after_col : کار این تابع حذف ستون های قبل از آن شماره ستونی است که به عنوان ورودی به آن داده ایم و برگرداندن بقیه ماتریس به عنوان خروجی . پیاده سازی این ماتریس به صورت زیر است.

```
Matrix Matrix::remain_matrix_after_col(size_t k)          //removes columns until (k-1-th) columns of the matrix
{
    std::vector<std::vector<double>> remain_vector{};     // to store remain elements

    for (size_t i = 0; i < size[0]; i++)                  //row
    {
        std::vector<double> remain_vector_row{};          //to store a row of element

        for (size_t j = k; j < size[1]; j++)              //save element which in column k and after that
            remain_vector_row.push_back(Data[i][j]);      // store  remaining element
        remain_vector.push_back(remain_vector_row);       // store a row of element
    }

    return Matrix{ remain_vector };                      // return remaining matrix
}
```

۳) re_row : کار این تابع برگرداندن یک سطر از ماتریسی است که شماره سطر مورد نظر را به عنوان ورودی به آن داده‌ایم . پیاده سازی این تابع به صورت زیر است.

```
Matrix Matrix::re_row(size_t k)                                    //return a row
{
    return Matrix{ std::vector<std::vector<double>>{Data[k]} };    // return k-th row as a matrix
}
```

۴) del_last_: کار این تابع حذف آخرین سطر از ماتریس و برگرداندن بقیه ماتریس است.

```
Matrix Matrix::del_last_row()
{
    std::vector<std::vector<double>> remain_vector{};  // to store remain elements

    for (size_t i = 0; i < size[0]-1; i++)             //row
        remain_vector.push_back(Data[i]);              //save row of matrix

    return Matrix{ remain_vector };                   // return remaining matrix
}
```

۵) attach : کار این تابع کنار هم چسباندن دو ماتریس و برگرداندن ماتریس جدید است دقت شود که باید تعداد سطرهای دو ماتریس برابر باشد. که به صورت زیر پیاده سازی می شود.

```
Matrix Matrix::attach(const Matrix& A)                          //attach to matrix
{
    if (size[0] != A.size[0])                                   //the numbers of two matrix must be same
    {
        std::cout << " can not to attech" << std::endl;
        return Matrix{};
    }

    Matrix result{ size[0],size[1] + A.size[1],false };         //make a new matrix to save attached of two matrixs

    for (size_t i = 0; i < A.size[0]; i++)                      //row
    {
        for (size_t j = 0; j < size[1]; j++)                    //column of first matrix
            result[i][j] = Data[i][j];                          //save element of first matrix
        for (size_t j = size[1]; j < A.size[1] + size[1]; j++)  //column of second matrix
            result[i][j] = A.Data[i][j-size[1]];                //save element of second matrix
    }

    return result;                                              //return attached of two matrixs
}
```

دومین class Dataset است که برای دسته بندی data ها است .

این class شامل member variable و همچنین member function هایی است که در ادامه به تعریف و بررسی تک تک آن ها می پردازیم.

member variable ها به صورت private :

۱)inputs : ماتریسی است که در آن data های ورودی قرار دارند.

۲)targets : ماتریسی است که در آن data های خروجی قرار دارند.

۳)train_inputs : ماتریسی است که در آن data های ورودی بخش train قرار دارند.

۴)train_targets : ماتریسی است که در آن data های خروجی بخش train قرار دارند.

۵)test_inputs : ماتریسی است که در آن data های ورودی بخش test قرار دارند.

۶)test _targets : ماتریسی است که در آن data های خروجی بخش test قرار دارند.

۷)percentage : مشخص میکننده سهم data های هرکدام از بخش های tarin وtest از مجموع کل data ها است.

۸)no_of_samples : مشخص میکننده تعدادکل نمونه ها است.

۹)input_dim : مشخص کننده‌ی ابعاد ورودی نمونه ها است.

۱۰)target_dim : مشخص کننده‌ی ابعاد خروجی نمونه ها است.

توجه: ورودی هر نمونه یک ستون از ماتریس inputs است.

کل class dataset در صفحه بعد قایل مشاهده است.

```
class Dataset
{
private:                        //Member variables
    Matrix inputs;              //Matrix of all input parts of samples of dataset
    Matrix targets;             //Matrix of all target parts of samples of dataset
    Matrix train_inputs;        //Matrix of inputs of the train part of dataset
    Matrix train_targets;       //Matrix of targets of the train part of dataset
    Matrix test_inputs;         //Matrix of inputs of the test part of dataset
    Matrix test_targets;        //Matrix of targets of the test part of dataset
    double percentage{ 70 };    //This variable tells the class how to divide data into train and test parts
    size_t no_of_samples;       //Number of all samples in the dataset
    size_t input_dim;           //Dimension of inputs of dataset
    size_t target_dim;          //Dimension of targets of dataset

public:
    Dataset(Matrix inputs, Matrix targets, double percentage = 70);       //Constructos
    Dataset();                                                            //Default constructor
                                                                          //Member functions
    size_t getNoOfSamples();                                              //get number of samples
    size_t getNoOfTrainSamples();                                         //get number of train samples
    size_t getNoOfTestSamples();                                          //get number of test samples
    size_t getInputDim();                                                 //get input dimension
    size_t getTargetDim();                                                //get target dimension
    Matrix getInputs();                                                   //get input matrix
    Matrix getTargets();                                                  //get target matrix
    Matrix getTrainInputs();                                              //get train inputs matrix
    Matrix getTrainTargets();                                             //get train target matrix
    Matrix getTestInputs();                                               //get test inputs matrix
    Matrix getTestTargets();                                              //get test targets matrix
    void shuffle();                                                       //rearranges the data samples
    void show();                                                          //show some properties of the dataset
                                                                          //operators
    std::vector<Matrix> operator[](size_t);                               //difine operator []
    Dataset operator+(const Dataset& dataset);                            //difine operator +
    friend std::ostream& operator<< (std::ostream& out, const Dataset& c);//difine operator <<
};
```

constructor : شامل سه ورودی است اولین آن ماتریس ورودی ها و دومی ماتریس خروجی ها و سومی که درصد تقسیم نمونه ها به دو بخش train و test است. پیاده سازی به صورت زیر است.

```
Dataset::Dataset(Matrix inputs, Matrix targets, double percentage)                                                  //constructor
{
    this->inputs = inputs;                                                                                          //save input matrix
    this->targets = targets;                                                                                        //save target matrix
    this->percentage = percentage;                                                                                  //save percentage
    no_of_samples = inputs.getSize()[1];                                                                            //save number of samples
    input_dim = inputs.getSize()[0];                                                                                //save input dimension
    target_dim = targets.getSize()[0];                                                                              //save target dimension
    train_inputs = inputs.remain_matrix_until_col(static_cast<size_t>(no_of_samples * (percentage / 100)-1));       //save train input matrix
    train_targets = targets.remain_matrix_until_col(static_cast<size_t>(no_of_samples * (percentage / 100) - 1));   //save train target matrix
    test_inputs = inputs.remain_matrix_after_col(static_cast<size_t>(no_of_samples-no_of_samples * (1 - (percentage / 100))));     //save test input matrix
    test_targets = targets.remain_matrix_after_col(static_cast<size_t>(no_of_samples-no_of_samples * (1 - (percentage / 100))));   //save test target matrix
}
```

default constructor به شرح زیر می‌باشد.

```
Dataset::Dataset()                                                                          //default construvtos
{
    inputs = targets = train_inputs = train_targets = test_inputs = test_targets = Matrix{};       //save member varibles
    no_of_samples = input_dim = target_dim = 0;
}
```

: member functions

ابتدا چند تابع getter را تعریف کرده که به صورت زیر پیاده سازی می‌شود.

```cpp
size_t Dataset::getNoOfSamples()                    //get number of samples
{
    return size_t{no_of_samples};
}

size_t Dataset::getNoOfTrainSamples()               //get number of train samples
{
    return size_t{train_inputs.getSize()[1]};
}

size_t Dataset::getNoOfTestSamples()                //get number of test samples
{
    return size_t{ test_inputs.getSize()[1] };
}

size_t Dataset::getInputDim()                       //get input dimension
{
    return size_t{input_dim};
}

size_t Dataset::getTargetDim()                      //get target dimension
{
    return size_t{target_dim};
}

Matrix Dataset::getInputs()                         //get input matrix
{
    return Matrix{inputs};
}

Matrix Dataset::getTargets()                        //get target matrix
{
    return Matrix{targets};
}

Matrix Dataset::getTrainInputs()                    //get train inputs matrix
{
    return Matrix{train_inputs};
}
```

```cpp
Matrix Dataset::getTrainTargets()                   //get train target matrix
{
    return Matrix{train_targets};
}

Matrix Dataset::getTestInputs()                     //get test inputs matrix
{
    return Matrix{test_inputs};
}

Matrix Dataset::getTestTargets()                    //get test targets matrix
{
    return Matrix{test_targets};
}
```

shuffle : کار این تابع عوض کردن ترتیب نمونه ها به صورت رندوم است در الگوریتم آن ابتدا دو ماتریس تعریف می‌شود که یکی برای ورودی های جدید و دیگری برای خروجی ها است . حال دو عدد به صورت رندوم میگریم که یکی شماره نمونه در ماتریس قدیم و یکی شماره نمونه در ماتریس جدید است . قبل از مقدار دهی بایدکنترل شود که آیا این نمونه از قبلا استفاده شده یا نه و آیا قبلا در ماتریس جدید در شماره ستونی که داریم نمونه ایی قبلا قرارگرفته یا نه ؟ بعد از بررسی این شرط ها نمونه در ماتریس قدیم را در یک ستون دیگر در ماتریس جدید قرار می‌دهیم و همین کار با ماتریس خروجی نیز میکنیم سپس برای علامت گزاری اینکه از این نمونه در ماتریس قدیم استفاده شده است اولین ورودی آن را برابر ۱۰۰۰- قرار می‌دهیم. سپس member varibles های جدید را ذخیره میکنیم. پیاده سازی آن به صورت زیر است.

```
void Dataset::shuffle()                                    //rearranges the data samples
{
    srand(static_cast<unsigned>(time(NULL)));

    size_t counter{};
    Matrix new_input{ std::vector<std::vector<double>>(inputs.getSize()[0], std::vector<double>(inputs.getSize()[1], -1000)) };
    Matrix new_target{ std::vector<std::vector<double>>(targets.getSize()[0], std::vector<double>(targets.getSize()[1], -1000)) };

    while (counter < no_of_samples)                                    //count until all samples rearrange
    {
        size_t i{ rand() % no_of_samples };                            //get a random column of  matrix
        size_t i_new{ rand() % no_of_samples };                        //get a random column of  new matrix

        if (new_input[0][i_new]==-1000 && inputs[0][i]!=-1000)          //check this column of new matrix is empty
        {                                                              //and check this sample of old matrix already use or not
            for (size_t row_input = 0; row_input < getInputDim(); row_input++)          //rearrange the element
                new_input[row_input][i_new] = inputs[row_input][i];
            for (size_t row_target = 0; row_target < getTargetDim(); row_target++)
                new_target[row_target][i_new] = targets[row_target][i];
            inputs[0][i] = -1000;                                       //this show this column of old matrix uesed
            counter++;
        }
    }

                                                                       //save member variables of nee matrix
    inputs = new_input;                                                //save new input matrix
    targets = new_target;                                              //save new target matrix
    train_inputs = inputs.remain_matrix_until_col(static_cast<size_t>(no_of_samples * (percentage / 100) - 1));                    //save train input matrix
    train_targets = targets.remain_matrix_until_col(static_cast<size_t>(no_of_samples * (percentage / 100) - 1));                  //save train target matrix
    test_inputs = inputs.remain_matrix_after_col(static_cast<size_t>(no_of_samples - no_of_samples * (1 - (percentage / 100))));   //save new test input matrix
    test_targets = targets.remain_matrix_after_col(static_cast<size_t>(no_of_samples - no_of_samples * (1 - (percentage / 100))));  //save new test target matrix
}
```

show : کار این تابع چاپ کردن بعضی از ویژگی های dataset است که به صورت زیر پیاده‌سازی می‌شود.

```
void Dataset::show()    //show some properties of the dataset
{
    std::cout << "Dataset:" << std::endl;
    std::cout << std::setw(20) << "No of samples: " << no_of_samples << std::endl;
    std::cout << std::setw(20) << "Train samples: " << getNoOfTrainSamples() << " samples" << std::endl;
    std::cout << std::setw(19) << "Test samples: " << getNoOfTestSamples() << " samples" << std::endl;
    std::cout << std::setw(23) << "Input dimensions: " << input_dim << std::endl;
    std::cout << std::setw(24) << "Target dimensions: " << target_dim << std::endl;
}
```

تعریف چند operator :

۱) operator[] : کار این operator این است که یک reference به object می‌باشدکه بیانگر شماره نمونه و آن object خود یک operator[] داردکه بیان گر این است ماتریس inputs یا targets است. به صورت زیر پیاده سازی میکنیم.

```
std::vector<Matrix> Dataset::operator[](size_t k)      //overload operator[] to return a reference to an object that represents the sample
{                                                      //and with itself has an appropriate operator[] to reference inputs matrix or target matrix
    return std::vector<Matrix>{ inputs.col(k),targets.col(k) };
}
```

۲) operator + : که کار این operator این است که دو dataset را درکنار هم قرار می‌دهد این کار را با تابع attach که در class matrix موجود است انجام می‌گیرد به این صورت که ماتریس های inputs راکنار هم و ماتریس های targets کنار هم میگذاریم و یک dataset جدید می‌سازیم. پیاده سازی این تابع به صورت زیر است.

```
Dataset Dataset::operator+(const Dataset& dataset)                                  //merge two datasets together
{
    return  Dataset{ inputs.attach(dataset.inputs),targets.attach(dataset.targets),percentage };  //return as a new dataset
}
```

۳) operator<< :کار این operator چاپ کردن بعضی از ویژگی های dataset است. قابل توجه اینکه خروجی ostream است و prototype را به صورتfriend و در داخل خود classتعریف می‌شودکه باعث می شود به متغیر های private اجازه دسترسی شود و به صورت زیر پیاده سازی می‌شود .

```
std::ostream& operator<<(std::ostream& out,const Dataset& c)          //show some properties of the dataset
{
    out << "Dataset:" << std::endl;
    out << std::setw(20) << "No of samples: " << c.no_of_samples << std::endl;
    out << std::setw(20) << "Train samples: " << static_cast<size_t>(c.no_of_samples * (c.percentage / 100)) << " samples" << std::endl;
    out << std::setw(19) << "Test samples: " << c.no_of_samples - static_cast<size_t>(c.no_of_samples * (c.percentage / 100)) << " samples" << std::endl;
    out << std::setw(23) << "Input dimensions: " << c.input_dim << std::endl;
    out << std::setw(24) << "Target dimensions: " << c.target_dim << std::endl;

    return out;
}
```

جواب Qusetion1 ۱)برای دسترسی به متغییر های private در یک باید prototype تابع را در خود class مدنظر قرار داد و قبل از آن کلمه friend را بگذاریم که باعث می‌شود در داخل تابع اجازه دسترسی به متغییرهای private داده می‌شود . مانند operator بالا.

۲)راه دوم این که از طریق آدرس متغییر های private به آن دسترسی پیدا کنیم. مانند مثال زیر:

```
class Test
{
private:
    int a{1};
    int data;
public:
    Test() { data = 0; }
    int getData() { return data; }
    int geta() { return a; }
};

int main()
{
    Test t;
    int* ptr = (int*)&t;
    *ptr = 100;          //access private variable
    *(ptr+1) = 10;
    std::cout << t.geta() << std::endl;
    std::cout << t.getData() << std::endl;

    return 0;
}
```

سومین class NeuralNet است که شبکه عصبی را در آن پیاده سازی می‌کنیم.

این class شامل member variable و همچنین member function هایی است که در ادامه به تعریف و بررسی تک تک آن ها می پردازیم.

member variable ها به صورت private :

۱)w1 : ماتریس وزن های لایه اول .

۲)w2 : ماتریس وزن های لایه دوم .

۳)b1 : ماتریس بایاس های لایه اول.

۴)b2 : ماتریس بایاس های لایه دوم.

۵)a1 : ماتریس خروجی لایه اول.

۶)a2 : ماتریس خروجی لایه دوم.

۷) n1 : ماتریس n لایه اول.

۸) n2 : ماتریس n لایه دوم.

۹) s1 : ماتریس s لایه اول.

۱۰) s2 : ماتریس s لایه دوم.

۱۱) af1 : نام activation function لایه اول.

۱۲) af2 : نام activation function لایه دوم.

۱۳) hidden_layer_neurons : تعداد نورن های لایه hidden.

۱۴) lr : مقدار learning rate برای اموزش شبکه .

۱۵) max_iters : مقدار حداکثر تکرار برای آموزش شبکه .

۱۶) min_loss : مقدار حداقل loss .

۱۷) dataset : dataها در آن قرار دارد.

در زیرکل class NeuralNet قابل مشاهده است.

```
class NeuralNet
{
private:                            //member functions
    Matrix w1;                      // Weights of layer 1
    Matrix w2;                      // Weights of layer 2
    Matrix b1;                      // Biases of layer 1
    Matrix b2;                      // Biases of layer 2
    Matrix a1;                      // Output of layer 1
    Matrix a2;                      // Output of layer 2
    Matrix n1;                      // n vector for layer 1
    Matrix n2;                      // n vector for layer 2
    Matrix s1;                      // s vector for layer 1
    Matrix s2;                      // s vector for layer 2

    const char* af1{ "Sigmoid" };   //activation function layer 1
    const char* af2{ "Sigmoid" };   //activation function layer 2
    size_t hidden_layer_neurons{ 3 };//No of neurons in the hidden layer of the network.
    double lr{ 0.01 };              //Learning rate for the training algorithm.
    size_t max_iters{ 1000 };       //Maximum iterations
    double min_loss{ 0.01 };        //Minimum test loss
    Dataset dataset;                //Given dataset

public:                                                 //member functions
    NeuralNet(Dataset dataset, size_t hidden_layer_neurons, const char* f1 = "Sigmoid", const char* f2 = "Linear", double lr = 0.1, size_t max_iters = 10000, double min_loss = 0.01);   //constructor

    Matrix forwardPropagate(Matrix& input);                 //do the forward propagations process
    void backPropagate(Matrix& input, Matrix& target);      //do the backpropagation for a sample data
    double trainLoss();                                     //computes the average loss function on the train part
    double testLoss();                                      //computes the average loss function on the test part
    Result fit();                                           //do all the training process of the neural network
    void show();                                            //shows a description of the neural network
                                                            //setter and getter functions
    void setW1(Matrix& w);                                  //set w1 matrix
    void setW2(Matrix& w);                                  //set w2 matrix
    void setB1(Matrix& b);                                  //set b1 matrix
    void setB2(Matrix& b);                                  //set b2 matrix
    Matrix getW1();                                         //get w1 matrix
    Matrix getW2();                                         //get w2 matrix
    Matrix getB1();                                         //get b1 matrix
    Matrix getB2();                                         //get b2 matrix

    friend std::ostream& operator<< (std::ostream& out, NeuralNet& c);//difine operator <<
};
```

قبل از نوشتن member function های class ابتدا چند تابع کمکی تعریف میکنم که در زیر prototaype های آن‌ها قابل مشاهده است.

```
Matrix sigmoid(Matrix);            //activation function sigmoid
Matrix linear(Matrix);             //activation function linear
double d_sigmoid(double);          //drivative function sigmoid
double d_linear(double);           //drivative function linear
Matrix operator*(double, Matrix);  //opreator multipluy scaler to matrix
```

حال به بررسی تک تک آن‌ها می‌پردازیم.

۱) sigmoid : وروی و خروجی آن به صورت ماتریس است و در آن هر المان ماتریس خروجی طبق فرمول زیر حساب می‌شود.

$$f(x) = \frac{1}{1 + exp(-x)}$$

پیاده‌سازی این تابع به صورت زیر است.

```
Matrix sigmoid(Matrix x)                            //activation function sigmoid
{
    for (size_t i = 0; i < x.getSize()[0]; i++)     //row
        for (size_t j = 0; j < x.getSize()[1]; j++) //column
            x[i][j] = 1 / (1 + exp(-x[i][j]));      //claculate elements
    return x;                                       //return  output matrix
}
```

۲) تابع linear : وروی و خروجی آن به صورت ماتریس است و در آن هر المان ماتریس خروجی طبق فرمول زیر حساب می‌شود.

$$f(x) = x$$

پیاده‌سازی این تابع به صورت زیر است.

```
Matrix linear(Matrix x)                             //activation function linear
{
    return x;                                       //return  output matrix
}
```

۳) d_sigmoid : وروی و خروجی آن به صورت ماتریس است و مشتق تابع sigmoid است. در آن هر المان ماتریس خروجی طبق فرمول زیر حساب می‌شود.

$$\frac{df}{dx} = f(x)(1 - f(x))$$

پیاده‌سازی این تابع به صورت زیر است.

```
double d_sigmoid(double x)                          //drivative function sigmoid
{
    return (1/(1+exp(-x)))*(1- (1 / (1 + exp(-x))));    //return drivative
}
```

۴) d_linear : وروی و خروجی آن به صورت ماتریس است و مشتق تابع linear است. در آن هر المان ماتریس خروجی طبق فرمول زیر حساب می‌شود.

$$\frac{df}{dx} = 1$$

پیاده‌سازی این تابع به صورت زیر است.

```
double d_linear(double x)                           //drivative function linear
{
    return 1;                                       //return drivative
}
```

operator* : کار این operator محاسبه ضرب کردن یک عدد در یک ماتریس است و به صورت زیر پیاده‌سازی می‌شود.

```
Matrix operator*(double a, Matrix B)                //opreator multipluy scaler to matrix
{
    for (size_t i = 0; i < B.getSize()[0]; i++)         //row
        for (size_t j = 0; j < B.getSize()[1]; j++)     //column
            B[i][j] *= a;                               //claculate each element
    return B;                                           //return matrix
}
```

member function های class NeuralNet :

تعریف constructor که شامل ورودی‌های زیر است.

۱) dataset : dataها در آن قرار دارد.

۲) hidden_layer_neurons : تعداد نورن های لایه hidden.

۳) f1 : نام activation function لایه اول .

۴) f2 : نام activation function لایه دوم .

۵) lr : مقدار learning rate برای اموزش شبکه .

۶) max_iters : مقدار حداکثر تکرار برای آموزش شبکه .

۷) min_loss : مقدار حداقل loss .

پیاده سازی این constructor به صورت زیر است.

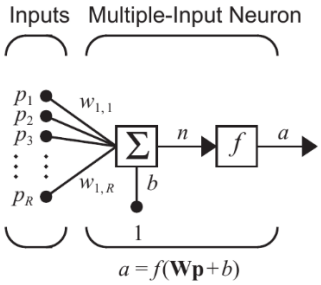
```cpp
NeuralNet::NeuralNet(Dataset dataset, size_t hidden_layer_neurons, const char* f1, const char* f2, double lr, size_t max_iters, double min_loss)   //constructor
{
    this->dataset = dataset;                            //save dataset
    this->hidden_layer_neurons = hidden_layer_neurons;  //save number of neurons in the hidden layer of the network
    this->af1 = f1;                                     //save activation function layer 1
    this->af2 = f2;                                     //save activation function layer 2
    this->lr = lr;                                      //save Learning rate
    this->max_iters = max_iters;                        //save Maximum iterations
    this->min_loss = min_loss;                          //save Minimum test loss
}
```

forwardpropagate : ورودی این تابع ماتریس ورودی های یک sample است.سپس با استفاده از الگوریتم زیر به محاسبه خروجی هر لایه می‌پردازیم.

$$\mathbf{n}^m = W^m\mathbf{p}^m + \mathbf{b}^m \qquad \text{m: index of layer}$$

$$\mathbf{a}^m = f^m(\mathbf{n}^m) \qquad \text{m: index of layer}$$

$$\mathbf{p}^m = \mathbf{a}^{m-1}$$



پیاده‌سازی این تابع به صورت زیر است.

```cpp
Matrix NeuralNet::forwardPropagate(Matrix& input)          //do the forward propagations process
{
    n1 = w1 * input + b1;                                  //claculate matrix n1
    std::string acf1{ af1 };
    std::string acf2{ af2 };

    if (acf1 == "Sigmoid")                                 //check activatin function layer 1
        a1 = sigmoid(n1);                                  //calculate matrix a1
    else if (acf1 == "Linear")
        a1 = linear(n1);
    else
    {
        std::cout << " inavalid activation function layer 1" << std::endl;
        return Matrix{};
    }

    n2 = w2 * a1 + b2;                                     //claculate matrix n2

    if (acf2 == "Sigmoid")                                 //check activatin function layer 2
        a2 = sigmoid(n2);                                  //calculate matrix a2
    else if (acf2 == "Linear")
        a2 = linear(n2);
    else
    {
        std::cout << " inavalid activation function layer 2" << std::endl;
        return Matrix{};
    }

    return a2;                                             //return matrix a2
}
```

backpropagate : دارای دو ورودی که یکی ماتریس ورودی و دیگری ماتریس خروجی یک sample است. کار این تابع آپدیت کردن وزن ها و بایاس ها در هر iteration است. الگوریتم آپدیت کردن وزن ها و بایاس به صورت زیر است .

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & \dot{f}^m(n_{S^m}^m) \end{bmatrix},$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}.$$

$$\mathbf{s}^2 = -2\dot{F}^2(\mathbf{n}^2)(\mathbf{t}-\mathbf{a})$$

$$\mathbf{s}^1 = \dot{F}^1(\mathbf{n}^1)(W^2)^T\mathbf{s}^2$$

$$W^m := W^m - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T \qquad m = 1,2$$

$$\mathbf{b}^m := \mathbf{b}^m - \alpha\mathbf{s}^m \qquad m = 1,2$$

پیاده سازی این تابع به صورت زیر است.

```
void NeuralNet::backPropagate(Matrix& input, Matrix& target)            //do the backpropagation for a sample data
{
    Matrix F1{hidden_layer_neurons,hidden_layer_neurons,false};         //matrix F1
    Matrix F2{ dataset.getTargetDim(),dataset.getTargetDim(),false };   //matrix F2

    if (af1 == "Sigmoid")                                               //check activatin function layer 1
        for (size_t i = 0; i < F1.getSize()[0]; i++)
            F1[i][i] = d_sigmoid(n1[i][0]);                             //calculate drivative
    if(af1=="Linear")
        for (size_t i = 0; i < F1.getSize()[0]; i++)
            F1[i][i] = d_linear(n1[i][0]);
    if (af2 == "Sigmoid")                                               //check activatin function layer 2
        for (size_t i = 0; i < F2.getSize()[0]; i++)
            F2[i][i] = d_sigmoid(n2[i][0]);                             //calculate drivative
    if (af2 == "Linear")
        for (size_t i = 0; i < F2.getSize()[0]; i++)
            F2[i][i] = d_linear(n2[i][0]);

    s2= -2 * F2 * (target - a2) ;                                       //calculate matrix s2
    s1 = F1 * (w2.T()) * s2;                                            //calculate matrix s1
    w1 = w1 - lr * s1 * (input.T());                                    //updata matrix w1
    w2 = w2 - lr * s2 * (a1.T());                                       //updata matrix w2
    b1 = b1 - lr * s1;                                                  //updata matrix b1
    b2 = b2 - lr * s2;                                                  //updata matrix b2
}
```

train loss : مقدار loss بخش train را طبق فرمول زیر حساب کند.

$$loss = \frac{\sum_{i=1}^{m}(y_{target} - y_{predicted})^2}{m}$$

پیاده سازی این تابع به صورت زیر است.

```cpp
double NeuralNet::trainLoss()                                   //computes the average loss function on the train part
{
    double loss{};                                              //loss

    for (size_t i = 0; i < dataset.getNoOfTrainSamples(); i++)  //each sample
    {
        Matrix input_f{ dataset.getTrainInputs().col(i) };      //matrix inputs each sample

        for (size_t j = 0; j < dataset.getTargetDim(); j++)     //calculate loss
            loss += (dataset.getTrainTargets().T()[i][j] - forwardPropagate(input_f)[j][0]) * (dataset.getTrainTargets().T()[i][j] - forwardPropagate(input_f)[j][0]);
    }

    return loss / dataset.getNoOfTrainSamples();                //return loss
}
```

train loss : مقدار loss بخش test را طبق فرمول بالا حساب کند. پیاده‌سازی این تابع به صورت زیر است.

```cpp
double NeuralNet::testLoss()                                    //computes the average loss function on the test part
{
    double loss{};                                              //loss

    for (size_t i = 0; i < dataset.getNoOfTestSamples(); i++)   //each sample
    {
        Matrix input_f{ dataset.getTestInputs().col(i) };       //matrix inputs each sample

        for (size_t j = 0; j < dataset.getTargetDim(); j++)     //calculate loss
            loss += (dataset.getTestTargets().T()[i][j] - forwardPropagate(input_f)[j][0]) * (dataset.getTestTargets().T()[i][j] - forwardPropagate(input_f)[j][0]);
    }

    return loss / dataset.getNoOfTestSamples();                 //return loss
}
```

fit : کار این تابع پردازش شبکه عصبی است . الگوریتم پیاده‌سازی آن به این صورت است که در هر iteration یک sample را به صورت رندوم انتخاب می‌کند و سپس آن را forwardpropagate میکند سپس با استفاده از تابع backpropagate وزن ها و بایاس ها را آپدیت می‌کند و این کار را تا وقتی انجام می‌دهدکه یا به max_iteration برسد یا loss train از min_loss کمتر بشود سپس در آخر یک object از class Result که در ادامه تعریف می‌کنیم برمی‌گرداند. این تابع به صورت زیر پیاده‌سازی می‌شود.

```cpp
Result NeuralNet::fit()
{
    size_t iteration{};                                                     //to count iteration
    w1 = Matrix{ hidden_layer_neurons,dataset.getInputDim(),false };        //initialize matrix w1
    w2 = Matrix{ dataset.getTargetDim(),hidden_layer_neurons,false };       //initialize matrix w2
    b1 = Matrix{ hidden_layer_neurons,1,true };                             //initialize matrix b1
    b2 = Matrix{ dataset.getTargetDim(),1,true };                           //initialize matrix b2
    srand(static_cast<unsigned>(time(NULL)));

    while (iteration < max_iters)
    {
        size_t no_sample_random{ rand() % dataset.getNoOfTrainSamples() };  //get a random sample
        Matrix sample_random_input{ dataset.getTrainInputs().col(no_sample_random) };   //get matrix inputs sample
        Matrix sample_random_output{ dataset.getTrainTargets().col(no_sample_random) }; //get matrix targets sample
        Matrix forwad_pro{ forwardPropagate(sample_random_input) };         //do forward propagate
        backPropagate(sample_random_input, sample_random_output);           //do backward propagate
        iteration++;
        if (trainLoss() < min_loss)                                         //check convergence
            break;
    }

    return Result{trainLoss(),testLoss(),hidden_layer_neurons,lr,iteration,af1,af2};    //return result
}
```

show : کار این تابع چاپ کردن بعضی از اطلاعات شبکه عصبی است و به صورت زیر پیاده‌سازی می‌شود.

```
void NeuralNet::show()                                          //shows a description of the neural network
{
    std::cout << "Neural Network:" << std::endl;
    std::cout << std::setw(26) << "No of hidden neurons: " << hidden_layer_neurons << std::endl;
    std::cout << std::setw(21) << "Input dimension: " << dataset.getInputDim() << std::endl;
    std::cout << std::setw(22) << "Output dimension: " << dataset.getTargetDim() << std::endl;
    std::cout << std::setw(32) << "Layer1 activation function: " << af1 << std::endl;
    std::cout << std::setw(32) << "Layer2 activation function: " << af2 << std::endl;
}
```

چند تابع getter و setter را به صورت زیر پیاده‌سازی می‌کنیم.

```
void NeuralNet::setW1(Matrix& w)          //set w1 matrix
{
    w1 = w;
}

void NeuralNet::setW2(Matrix& w)         //set w2 matrix
{
    w2 = w;
}

void NeuralNet::setB1(Matrix& b)         //set b1 matrix
{
    b1 = b;
}

void NeuralNet::setB2(Matrix& b)         //set b2 matrix
{
    b2 = b;
}

Matrix NeuralNet::getW1()                //get w1 matrix
{
    return w1;
}

Matrix NeuralNet::getW2()               //get w2 matrix
{
    return w2;
}

Matrix NeuralNet::getB1()               //get b1 matrix
{
    return b1;
}

Matrix NeuralNet::getB2()              //get b2 matrix
{
    return b2;
}
```

در آخر یک operator<<را تعریف می‌کنیم ؛ که کار این operator چاپ کردن بعضی از اطلاعات شبکه عصبی است. فقط باید توجه کردکه خروجی ostream است و prototype را به صورت friend و در داخل خود classتعریف می‌شودکه باعث می شود به متغیر های private اجازه دسترسی داده شود و به صورت زیر پیاده سازی می‌شود.

```cpp
std::ostream& operator<<(std::ostream& out, NeuralNet& c)          //shows a description of the neural network
{
    out << "Neural Network:" << std::endl;
    out << std::setw(26) << "No of hidden neurons: " << c.hidden_layer_neurons << std::endl;
    out << std::setw(21) << "Input dimension: " << c.dataset.getInputDim() << std::endl;
    out << std::setw(22) << "Output dimension: " << c.dataset.getTargetDim() << std::endl;
    out << std::setw(32) << "Layer1 activation function: " << c.af1 << std::endl;
    out << std::setw(32) << "Layer2 activation function: " << c.af2 << std::endl;

    return out;
}
```

چهارمین class Result است که اطلاعاتی از شبکه عصبی را ذخیره می‌کند.

این class شامل member variable و همچنین member function هایی است که در ادامه به تعریف و بررسی تک تک آن ها می پردازیم.

تعریف member variable ها به صورت private :

۱)train_loss : مقدار loss بخش trian.

۲)test_loss : مقدار loss بخش test.

۳)hidden_layer_neurons : تعداد نورن های لایه hidden.

۴)lr : مقدار learning rate برای اموزش شبکه.

۵)iters :تعداد iteration ها.

۶)af1 : نام activation function لایه اول.

۷) af2 : نام activation function لایه دوم.

در زیرکل class Result را مشاهده می‌کنیم.

```cpp
class Result
{
private:                            //member variable
    double train_loss;              //average loss function on the train part
    double test_loss;               //average loss function on the test part
    size_t no_of_hidden_neurons;    //No of neurons in the hidden layer of the network
    double lr{};                    //Learning rate for the training algorithm.
    size_t iters{};                 //iterations
    const char* af1;                //activation function layer 1
    const char* af2;                //activation function layer 2

public:                             //member functions
    Result(double train_loss, double test_loss, size_t no_of_hidden_neurons, double lr = 0.01, size_t iters = 10000, const char* af1 = "Sigmoid", const char* af2 = "Linear");  //constructor
    explicit Result( double test_loss);                             //explicit keyword makes this constructor ineligible for implicit conversions
    Result();                                                       //default constructor

    double getTestLoss();                                           //get loss on the test part
    void show();                                                    //shows a description of the result object
                                                                    //operator
    bool operator<(const Result&) const;                            //define operator <
    bool operator==(const Result&) const;                           //define operator ==
    friend std::ostream& operator<< (std::ostream& out, const Result& c);//difine operator <<
};
```

member function ها:

تعریف سه constructor برای این class :

constructor اول ؛ چند ورودی به شرح زیر دارد :

۱)train_loss : مقدار loss بخش trian .

۲)test_loss : مقدار loss بخش test .

۳)hidden_layer_neurons : تعداد نورن های لایه hidden .

۴)lr : مقدار learning rate برای اموزش شبکه .

۵)iters :تعداد iteration ها .

۶) af1 : نام activation function لایه اول .

۷)af2 : نام activation function لایه دوم .

پیاده‌سازی این constructor به صورت زیر است.

```
Result::Result(double train_loss, double test_loss, size_t no_of_hidden_neurons, double lr, size_t iters, const char* af1, const char* af2)
{
    this->train_loss = train_loss;                          //save train loss
    this->test_loss = test_loss;                            //save test loss
    this->no_of_hidden_neurons = no_of_hidden_neurons;      //save Number of neurons in the hidden layer
    this->lr = lr;                                          //save Learning rate
    this->iters = iters;                                    //save iterations
    this->af1 = af1;                                        //save activation function layer 1
    this->af2 = af2;                                        //save activation function layer 2
}
```

constructor دوم ؛ یک ورودی دارد که آن test_loss است و به صورت زیر پیاده‌سازی می‌شود.

```
Result::Result(double test_loss)
{
    this->test_loss = test_loss;                            //save train loss
    train_loss = -1;                                        //save test loss
    no_of_hidden_neurons = 0;                               //save Number of neurons in the hidden layer
    lr = 0.01;                                              //save Learning rate
    iters = 10000;                                          //save iterations
    af1 = "Sigmoid";                                        //save activation function layer 1
    af2 = "Linear";                                         //save activation function layer 2
}
```

default constructor سوم ؛ به صورت زیر تعریف می‌شود.

```
Result::Result()                                            //default constructor
{
    test_loss = 0;                                          //save train loss
    train_loss = -1;                                        //save test loss
    no_of_hidden_neurons = 0;                               //save Number of neurons in the hidden layer
    lr = 0.01;                                              //save Learning rate
    iters = 10000;                                          //save iterations
    af1 = "Sigmoid";                                        //save activation function layer 1
    af2 = "Linear";                                         //save activation function layer 2
}
```

getTestLoss : کار آن برگرداندن مقدار test_loss و به صورت زیر پیاده‌سازی می‌شود.

```
double Result::getTestLoss()                                    //get loss on the test part
{
    return test_loss;
}
```

show : کار این تابع چاپ کردن بعضی از اطلاعات شبکه عصبی است و به صورت زیر پیاده‌سازی می‌شود.

```
void Result::show()                                             //shows a description of the result object
{
    std::cout << "Result:" << std::endl;
    std::cout << std::setw(16) << "Train loss: " << train_loss << std::endl;
    std::cout << std::setw(15) << "Test loss: " << test_loss << std::endl;
    std::cout << std::setw(26) << "No of hidden neurons: " << no_of_hidden_neurons << std::endl;
    std::cout << std::setw(32) << "Layer1 activation function: " << af1 << std::endl;
    std::cout << std::setw(32) << "Layer2 activation function: " << af2 << std::endl;
}
```

تعریف چند operator :

۱)operator<() است ؛که کار مقایسه test loss های دو object از class Result است و به صورت زیر پیاده‌سازی می‌شود.

```
bool Result::operator<(const Result& A) const          //define operator <
{
    if (test_loss < A.test_loss)                       //compare test loss
        return true;
    else
        return false;
}
```

۲)operator>() است ؛ که کار مقایسه test loss های دو object از class Result است و به صورت زیر پیاده‌سازی می‌شود.

```
bool Result::operator==(const Result& A) const         //define operator ==
{
    if (test_loss == A.test_loss)                      //compare test loss
        return true;
    else
        return false;
}
```

۳) operator<<() است ؛ که کار این operator چاپ کردن بعضی از اطلاعات شبکه عصبی است. باید توجه که خروجی ostream است و prototype را به صورتfriend و در داخل خود classتعریف می‌شود که باعث می شود به متغیر های private اجازه دسترسی داده می‌شود و به صورت زیر پیاده سازی می‌شود.

```
std::ostream& operator<<(std::ostream& out, const Result& c)  //difine operator <<
{                                                             //shows a description of the result object
    out << "Result:" << std::endl;
    out << std::setw(16) << "Train loss: " << c.train_loss << std::endl;
    out << std::setw(15) << "Test loss: " << c.test_loss << std::endl;
    out << std::setw(26) << "No of hidden neurons: " << c.no_of_hidden_neurons << std::endl;
    out << std::setw(32) << "Layer1 activation function: " << c.af1 << std::endl;
    out << std::setw(32) << "Layer2 activation function: " << c.af2 << std::endl;

    return out;
}
```

جواب Qusetion2 : برای اینکه بدون نوشتن تمام comparison operator ها آن ها را پیاده‌سازی کنیم باید ابتدا utility library را include کنیم سپس using namespace std::rel_ops را بنویسم سپس باید دو operator< و operator== را تعریف شود سپس تمام comparison operator ها به صورت خودکار تعریف می‌گردد.

```
#include<utility>

using namespace std::rel_ops;
```

جواب Qusetion3 : این کد درست کار می‌کند چون compiler به طور خوکار object r1 ؛

cast به double می‌شود و باعث می‌شود r1>10 اجرا شود . برای جلوگیری از این کار باید قبل از constructor آن کلمه‌ی explicit را بنوسیم.

```
explicit Result( double test_loss);
```

بعد از اتمام تعریف class ها چند تابع دیگر لازم است تعریف شود . prototype های آن ها را در زیر مشاهده می‌کنید.

```
Dataset loadFuncDataset(const char* filename);                                                                                                       //gets a csv filenameand turn it into a dataset
std::vector<Result> testNeuralNets(Dataset& dataset, std::vector<size_t>& hidden_neurons, double lr = 0.01, size_t max_iters = 10000, const char* af1 = "Sigmoid", const char* af2 = "Linear"); //This function generates a bunch of neural networks with a given dataset
Result findBestNeuralNet(Dataset& dataset, std::vector<size_t>& hidden_neurons, double lr = 0.01, size_t max_iters = 10000, const char* af1 = "Sigmoid", const char* af2 = "Linear");          //generate a bunch of neural networks and then return the best result
void estimateFunction(const char* filename, size_t hidden_neurons_no);                                                                               //generate neural networks and estimate
```

۱) loadfuncDataset : ورودی این تابع اسم فایلی است که data ها در آن قرار دارند و کار این تابع خواندن این فایل و تبدیل آن به یک dataset است که به صورت زیر پیاده‌سازی می‌شود.

```
Dataset loadFuncDataset(const char* filename)                        //gets a csv filenameand turn it into a dataset
{
    Matrix data{};
    data.load(filename);                                             //load csv file

    return Dataset{ data.del_last_row(),data.re_row(data.getSize()[0] - 1) };    //return dataset
}
```

۲) testNeralNest : که شامل ورودی‌های زیر است.

۱)dataset : dataها در آن قرار دارد.

۲)hidden _neurons : یک وکتور که شمال تعداد نورن های لایه hidden .

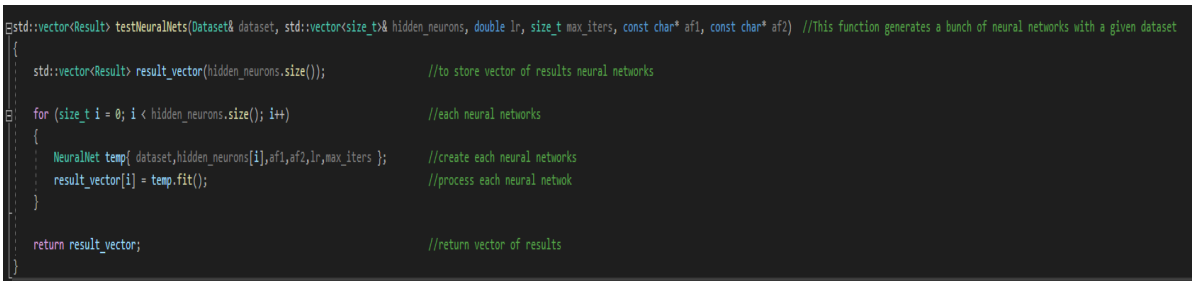
۳)lr : مقدار learning rate برای آموزش شبکه .

۴)max_iters : مقدار حداکثر تکرار برای آموزش شبکه .

۵) af1 : نام activation function لایه اول .

۶)af2 : نام activation function لایه دوم .

کار ین تابع این است که از روی dataset داده شده به آن چند شبکه با تعداد نورن های لایه hidden مختلف که در ورودی به صورت یک وکتور داده شده است ؛ به وجود آورد و نتایج شبکه ها را به صورت یک وکتور که object آن از class Result است ؛برگرداند و به صورت زیر پیاده‌سازی می‌شود.

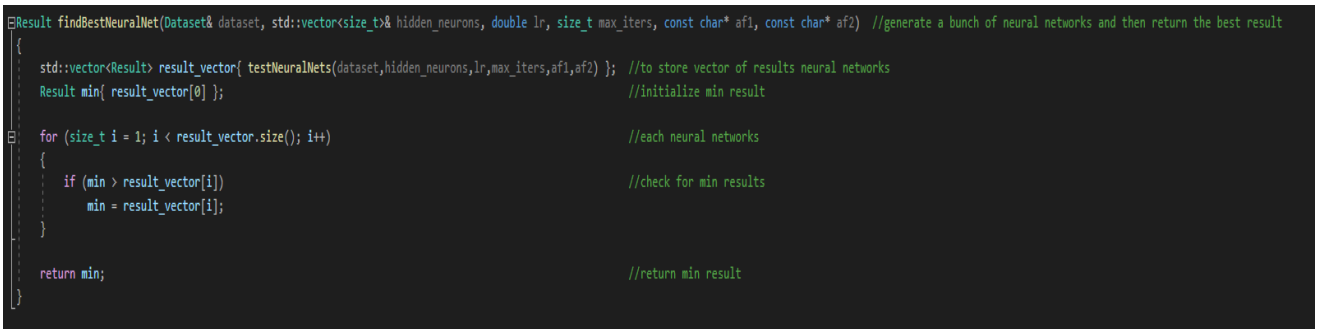


```
std::vector<Result> testNeuralNets(Dataset& dataset, std::vector<size_t>& hidden_neurons, double lr, size_t max_iters, const char* af1, const char* af2)  //This function generates a bunch of neural networks with a given dataset
{
    std::vector<Result> result_vector(hidden_neurons.size());                //to store vector of results neural networks

    for (size_t i = 0; i < hidden_neurons.size(); i++)                       //each neural networks
    {
        NeuralNet temp{ dataset,hidden_neurons[i],af1,af2,lr,max_iters };    //create each neural networks
        result_vector[i] = temp.fit();                                       //process each neural netwok
    }

    return result_vector;                                                    //return vector of results
}
```

۳) findBestNeuralNet : که دقیقا مثل تابع قبل است با این تفاوت که در خروجی Result شبکه ایی را برمی‌گرداندکه دارای کمترین test loss است و به صورت زیر پیاده‌سازی می‌شود.



```
Result findBestNeuralNet(Dataset& dataset, std::vector<size_t>& hidden_neurons, double lr, size_t max_iters, const char* af1, const char* af2)  //generate a bunch of neural networks and then return the best result
{
    std::vector<Result> result_vector{ testNeuralNets(dataset,hidden_neurons,lr,max_iters,af1,af2) };  //to store vector of results neural networks
    Result min{ result_vector[0] };                                                                     //initialize min result

    for (size_t i = 1; i < result_vector.size(); i++)                                                   //each neural networks
    {
        if (min > result_vector[i])                                                                     //check for min results
            min = result_vector[i];
    }

    return min;                                                                                         //return min result
}
```

۴) estimationFunction : دارای دو ورودی است.

۱)filename : اسم فایلی است که data ها در آن قرار دارد.

۲)hideen_neurons_no : تعداد نورن های لایه hidden.

کار این تابع ابتدا درست کردن object dataset از روی فایل csv. است سپس ساختن و آموزش دادن یک شبکه با تعداد نورن‌های داده شده است و در تنها مقادیر واقعی و تخمینی که به وسیله شبکه عصبی به‌دست آمده را چاپ میکند. پیاده‌سازی این تابع به صورت صفحه بعد است.

```
void estimateFunction(const char* filename, size_t hidden_neurons_no)
{
    Dataset data{ loadFuncDataset(filename) };            //gets a csv filenameand turn it into a dataset
    NeuralNet net_data{ data,hidden_neurons_no };         //create neural network
    net_data.fit();                                       //train neural netwotk
                                                          //show the estimate numbers
    std::cout <<std::setw(10)<< "No" << std::setw(18) << "Target" << std::setw(22) << "Estimated" << std::endl;
    for (size_t i = 0; i < 10; i++)
        std::cout << "------";
    std::cout << std::endl;

    for (size_t i = 0; i < data.getNoOfSamples(); i++)    //show each target and estimate
    {
        Matrix input_matrix{ data.getInputs().col(i) };   //get inputs matrix of each sample
        std::cout << "        ";
        std::cout<<std::setiosflags(std::ios::left)<<std::setw(14)<< i + 1 << std::setw(19) << data.getTargets()[0][i];
        std::cout<< net_data.forwardPropagate(input_matrix)[0][0] << std::endl;
    }
}
```

توجه۱ : در ورودی بعضی از تابع ها ؛ دلیل اینکه ورودی را به صورت refernce می‌دهیم این است که باعث افزایش performance و صرفه‌جویی در حافظه می‌شود. و در برخی موارد ورودی را به صورت const هم می‌دهیم که این کار برای جلوگیری از اجازه تغییر در تابع است.

توجه۲: توضیحات مربوط به پیاده سازی الگوریتم ها در داخل کد به صورت کامنت نوشته شده است و در این گزارش کار سعی شده در مورد الگوریتم توابع توضیح داده شود.

توجه۳: این پروژه همانطورکه در تصاویر مشخص است ابتدا در visual studio 2019 نوشته و سپس در visual studio code به وسیله docker اجرا و نتایج صفحه بعد بدست آمده است.

نتایج google test به شرح زیر است.

```
RUNNING TESTS ...
[==========] Running 5 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 5 tests from APHW3Test
[ RUN      ] APHW3Test.DatasetTest
[       OK ] APHW3Test.DatasetTest (2 ms)
[ RUN      ] APHW3Test.NeuralNetFPTest
[       OK ] APHW3Test.NeuralNetFPTest (0 ms)
[ RUN      ] APHW3Test.NeuralNetFitTest
[       OK ] APHW3Test.NeuralNetFitTest (1045 ms)
[ RUN      ] APHW3Test.ResultTest
[       OK ] APHW3Test.ResultTest (0 ms)
[ RUN      ] APHW3Test.SumTest
[       OK ] APHW3Test.SumTest (0 ms)
[----------] 5 tests from APHW3Test (1051 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (1053 ms total)
[  PASSED  ] 5 tests.
Here!
<<<SUCCESS>>>
```

همانطورکه قابل مشاهده است نتایج google test موفقیت آمیز بوده همچین خروجی تابع estimationFunction به صورت زیر است.

```
       No           Target           Estimated
------------------------------------------------------------
        1                0            0.083617
        2       0.00308267            0.104433
        3        0.0123117            0.127115
        4        0.0276301            0.151784
        5        0.0489435            0.178557
        6        0.0761205            0.207544
        7         0.108993            0.238851
        8          0.14736            0.272571
        9         0.190983            0.308787
       10         0.239594            0.347559
       11         0.292893             0.38893
       12         0.350552            0.432916
       13         0.412215            0.479505
       14         0.477501            0.528654
       15          0.54601            0.580281
       16         0.617317            0.634272
       17         0.690983            0.690472
```

تمام