github.com/christycty

REPORT OF MY FIRST MACHINE LEARNING PROJECT

1.1 INTRODUCTION

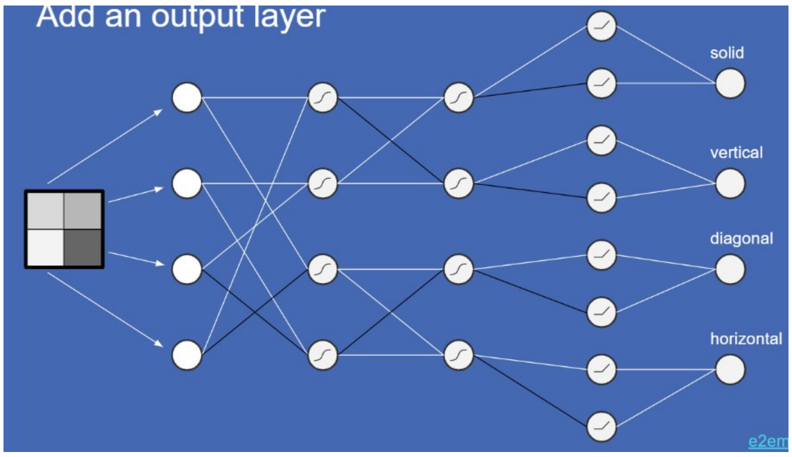
Inspired by a machine learning tutorial online[1], I tried to implement a machine learning algorithm to identify patterns (solid, horizontal, vertical, diagonal) from grayscale 2x2 pixels.

The training process consists mainly of two parts: data generation and model training.

For the data generation, I generated 4 random numbers between -1 and 1. Then I implemented the "correct model" shown in the tutorial to generate the optimal value for each node of the output layer.
(The C++ code of the data generator and training model are attached in appendix)

In each training, the results generated by the model is compared to the optimal value to compute the total error. With the use of back propagation, each weight is slightly adjusted to reduce the error.

Eventually, after thousands of trainings, the weights shall be adjusted in a way that can minimize the error.



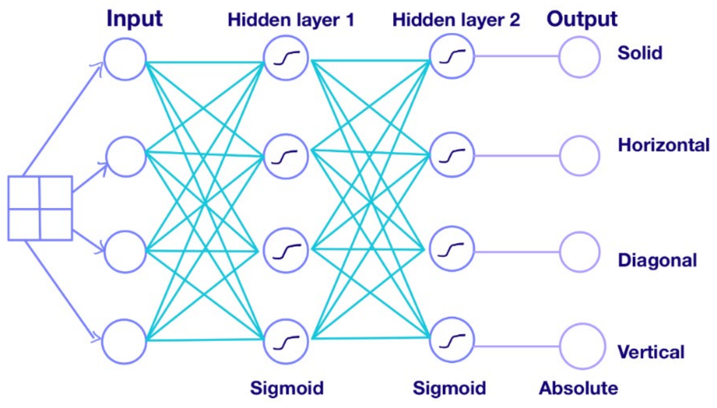
This is the "Correct model" extracted from the tutorial

Black and white edges represent weights -1 and 1 respectively, while omitted edges are weighted 0.

The third hidden layer is a rectified linear unit function

1.2 MODEL

I have simplified the training model in the tutorial into the one shown in the diagram below.
Each node of input represents a pixel, it is valued from -1 (black) to 1 (white).
There are 4 * 4 = 16 weighted edges connecting input layer and hidden layer 1, all edges values between -1 and 1. In hidden layer 1, values of input layer * edge weight are summed up, then squashed with sigmoid function, such that the value remains within -1 and 1.
A similar operation is carried out in hidden layer 2.
Finally, the output layer is the absolute value of the corresponding nodes of hidden layer 2.



[1] Neural networks tutorial by Brandon Rohrer https://www.youtube.com/watch?v=ILsA4nyG7I0

1.3 TRAINING PROGRESS

1.3.1 Change in Edge Weights

Initially, random float numbers between -1 and 1 are assigned to the two layers of edge weight. After 10 trials, the edge weights are slightly adjusted.

Initial		**After 10 Trials**	
First Layer Weight	Second Layer Weight	First Layer Weight	Second Layer Weight
0.377055	0.256963	0.376055	0.256463
-0.112175	0.282627	-0.111875	0.282927
-0.00853242	-0.389842	-0.00803242	-0.389342
-0.476013	0.0556133	-0.476513	0.0553133
-0.183633	-0.167848	-0.184133	-0.166948
-0.277689	0.20392	-0.277189	0.20402
0.693171	0.377055	0.693071	0.376155
-0.391522	-0.112175	-0.391422	-0.111875
0.407575	0.00853242	0.406875	0.00823242
-0.87908	0.476013	-0.87708	0.475713
-0.137144	-0.183633	-0.137044	-0.183333
0.692602	-0.277689	0.692102	-0.277389
-0.311144	0.693171	-0.310844	0.692871
0.534157	-0.391522	0.534257	-0.390622
-0.464477	0.407575	-0.463777	0.407475
0.864577	0.87908	0.864277	0.87818

After 100 trials, the change in weights has become larger.

After 350000 trials, the weights have moved towards -1/0/1, which matches the optimal solution.

After 100 Trials		**After 350000 Trials**	
First Layer Weight	Second Layer Weight	First Layer Weight	Second Layer Weight
0.372756	0.257263	0.0593689	0.0088608
-0.115275	0.280327	0.9976	0.000527722
-0.0120324	-0.383743	0.9996	-0.9999
-0.472913	0.0573133	-0.0795412	0.138715
-0.187333	-0.166148	-0.0763306	0.00145453
-0.279789	0.20502	-0.9998	0.109621
0.68987	0.371756	1	0.00196843
-0.387822	-0.112675	-0.0544364	0.00352628
0.406675	0.0132324	0.0790924	0.112831
-0.872779	0.469714	0.9988	-0.00085831
-0.134044	-0.177933	0.904795	0.00386946
0.689102	-0.27159	0.0558982	0.00261084
-0.304045	0.68427	-0.0546487	0.0612718
0.538858	-0.381423	0.9978	-0.00023763
-0.457778	0.401276	0.899231	0.174494
0.857876	0.868778	0.0728507	0.9999

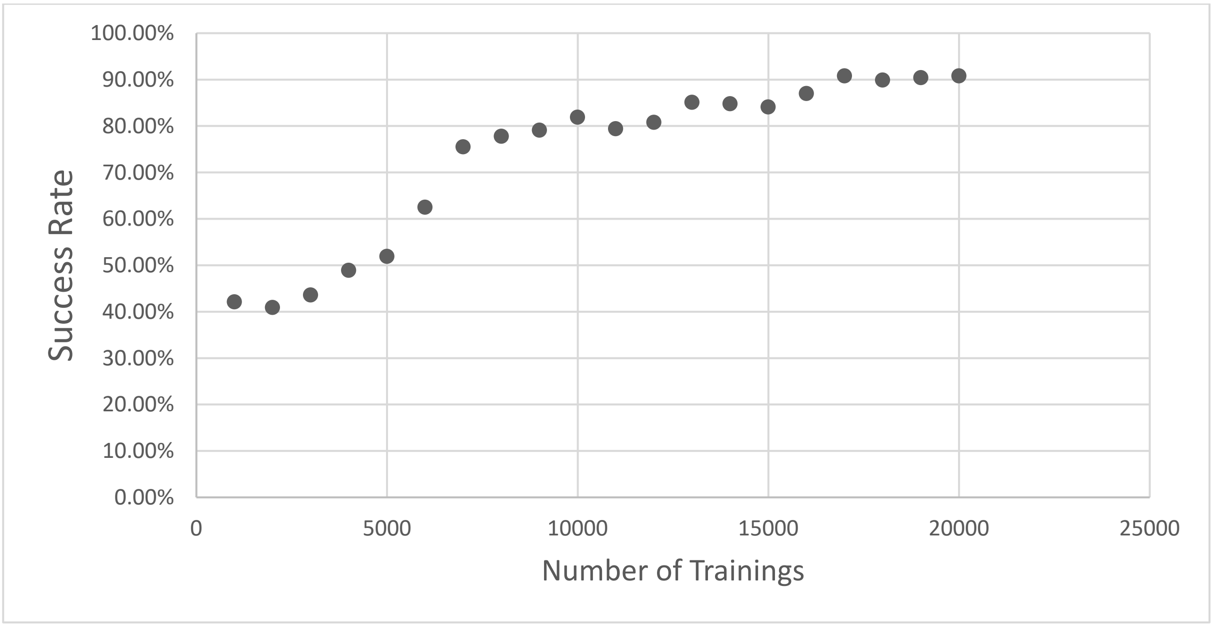
1.3.2 Change in Success Rate

I defined a "successful trial" as one in which the model correctly identifies the pattern (the value of that pattern is highest in the output layer)

In first 10 trials, there are only 2 successful trials (20%).

After 100 trials, the number of successful trials is 42. After another 100 trials, the number of success trials is 37.

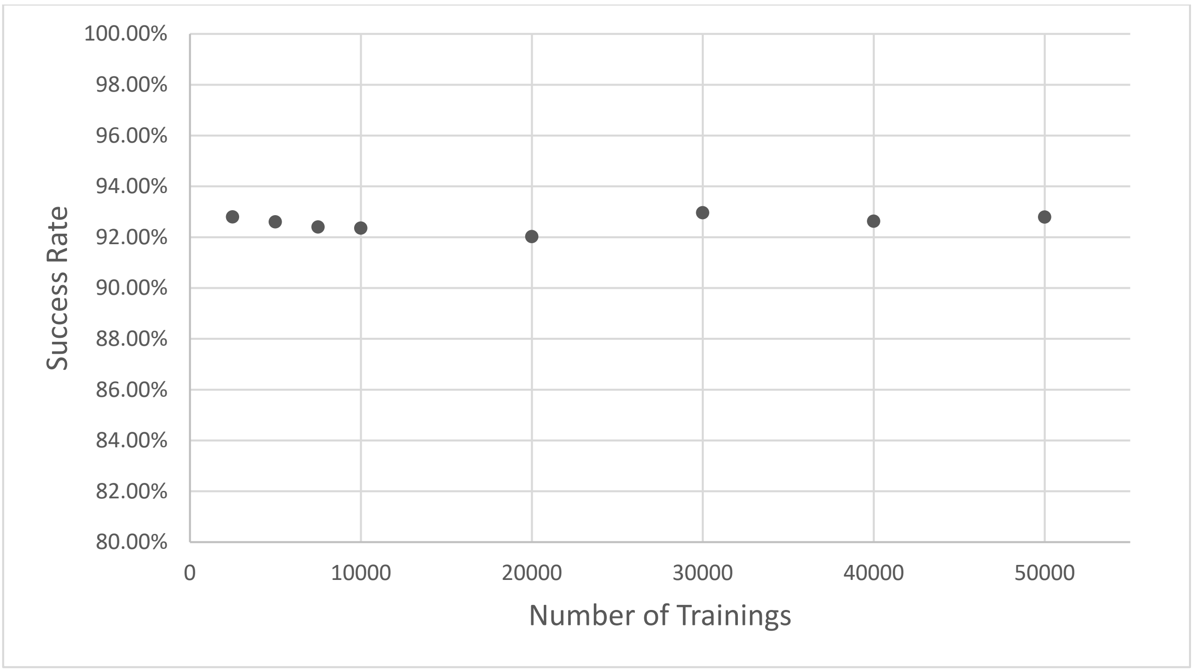
Afterwards, I carried out 20 times of trainings, each with 1000 sets of data (total 20000 trainings)



It can be observed that the success rate grew fast in the first 7000 trainings.

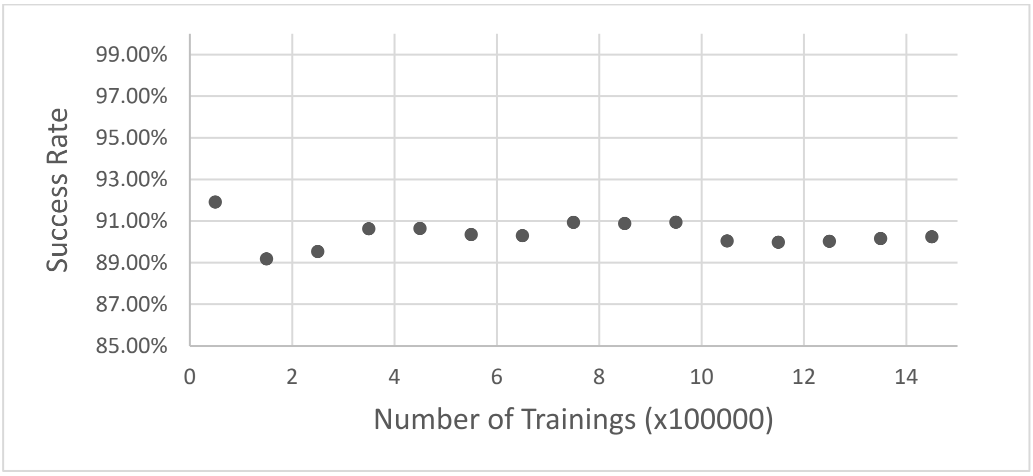
Afterward, the growth slowed down and eventually the success rate remains 90%.

Another 50000 trainings are carried out



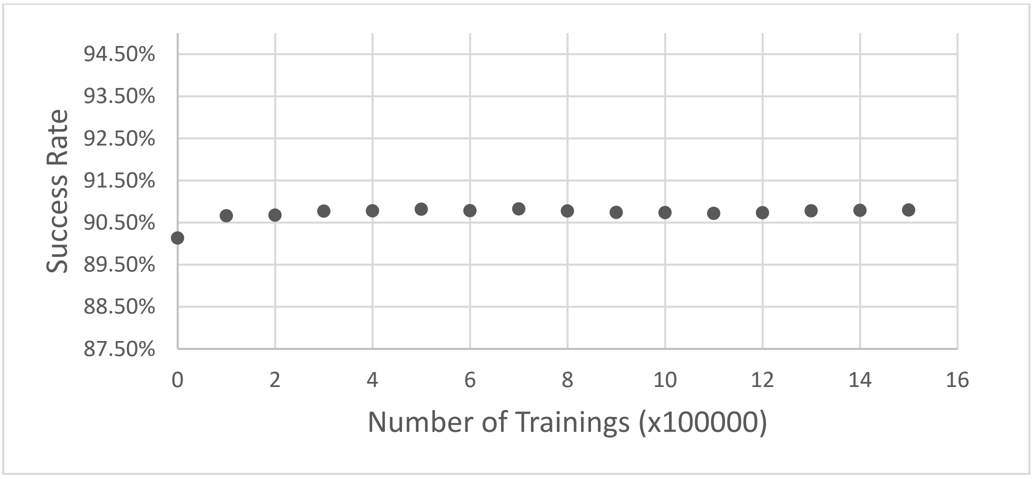
The success rate remains at around 92% to 93%.

Another 1550000 trainings are carried out.



Interestingly, the success rate slightly dropped and remains around 90% to 91%.

The reason may be the adjustment magnitude is too large. Therefore, the adjust magnitude is decreased from 0.0001 to 0.000001.



However, the success rate remains relatively constant without much improvements. It is hence concluded that this is the best that can be achieved by this model and training method.

github.com/christycty

1.4 Reflection

Even though it is just a mini scale AI which is quite simple and straight-forward, I still gained a lot from the process of building it from scratch.

First and foremost, it is the first time for me to carry out self-initiated projects. From going online and learn about machine learning and neural networks, to implementing and training the model, I managed to complete it independently without any guidance. I learnt to overcome the sense of helplessness and doubt when facing difficulties, as well as building up the mindset to take the initiative to learn and applying new knowledge. It is something that is rarely covered in mainstream schools, and something that I lacked before starting the project. I was amazed by the fact that I managed to understand the tutorials and even university lectures online, which I never thought I could. Through the project, I become more motivated to start my own projects for the sake of self-improvement and have more courage to carry out my plans. I believe this ability is crucial to my future studies in universities, in which students should be self-learners and be able to carry out independent investigations and projects.

Secondly, my knowledge in machine learning and artificial intelligence significantly improved. Because of the project, I went online to learn not only the basic principles, but also the implementation details and techniques like squashing and back propagation. I believe the experience, knowledge and skill enable me to better understand and master the related concepts and techniques of building and applying artificial intelligence in my future studies.

After trying out this mini model, I found the process of designing, implementing and training a machine learning model interesting and fulfilling. After DSE, I will love to build more complicated models, such as an AI for Tic Tac Toe or digits identification. To write a larger variety of models, I plan to learn Python and more advanced concepts and techniques like genetic algorithms in the future.

Overall, this project is meaningful for me because it is my first ever AI model. I found the process challenging and rewarding, and I hope to carry out more similar self-initiated AI projects in the future.

1.5 Program Code

1.5.1 DATA GENERATOR

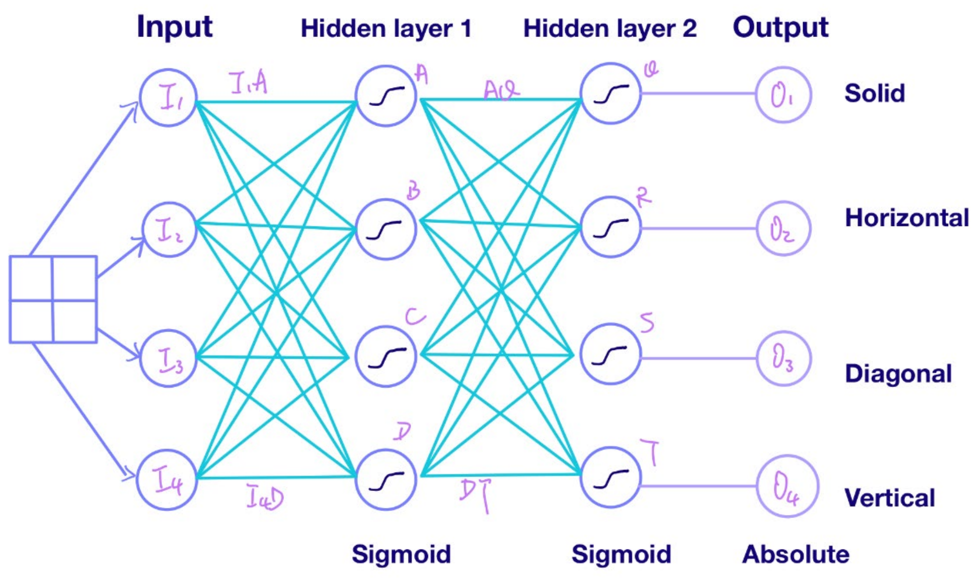
```cpp
#include<bits/stdc++.h>
using namespace std;

int dataCount;
float pixel[4]; //input layer
float firstLayer[4], secondLayer[4], thirdLayer[8], outLayer[4];

string genFileName(int n){
    string name = "C:\\projects\\patternMatching\\Generate_Data\\data\\", number;
    //add trailing 0s
    int count = 0, tmp = n;  //number of digits of n
    while (tmp > 0) {
        tmp /= 10;
        ++count;
    }
    for (int i = 0; i < 10 - count; ++i) name += "0";
    //convert number to string
    stringstream ss;
    ss << n;
    ss >> number;

    name += number;
    name += ".txt";
    return name;
}

void genSolution(){
    //first layer
    firstLayer[0] = pixel[0] + pixel[3];
    firstLayer[1] = pixel[1] + pixel[2];
    firstLayer[2] = pixel[0] - pixel[3];
    firstLayer[3] = pixel[1] - pixel[2];
    //tanh squashing function
    for (int i = 0; i < 4; ++i) firstLayer[i] = tanh(firstLayer[i]);

    //second layer
    secondLayer[0] = firstLayer[0] + firstLayer[1];
    secondLayer[1] = -firstLayer[0] + firstLayer[1];
    secondLayer[2] = firstLayer[2] - firstLayer[3];
    secondLayer[3] = firstLayer[2] + firstLayer[3];
    //tanh squashing function
    for (int i = 0; i < 4; ++i) secondLayer[i] = tanh(secondLayer[i]);

    //third layer
    for (int i = 0; i < 4; ++i){
        //rectified linear units
        thirdLayer[2*i] = (secondLayer[i] > 0 ? secondLayer[i] : 0);
        thirdLayer[2*i+1] = (-secondLayer[i] > 0 ? -secondLayer[i] : 0);
    }

    //out layer
    for (int i = 0; i < 4; ++i){
        outLayer[i] = thirdLayer[2*i] + thirdLayer[2*i+1];
    }

    //output check
    /*for (int i = 0; i < 4; ++i) cout << firstLayer[i] << " "; cout << "\n";
    for (int i = 0; i < 4; ++i) cout << secondLayer[i] << " "; cout << "\n";
    for (int i = 0; i < 8; ++i) cout << thirdLayer[i] << " "; cout << "\n";
    for (int i = 0; i < 4; ++i) cout << outLayer[i] << " "; cout << "\n============\
n";*/
}

int main(){
    srand(time(NULL));
    ifstream inFile;
    inFile.open("datacount.txt");
    inFile >> dataCount;
    inFile.close();
    ofstream outFile;
    for (int i = 1; i <= dataCount; ++i){   //generate 10000 data
        outFile.open( genFileName(i) );
        //generate 4 random float between -1 and 1
        for (int i = 0; i < 4; ++i){
            pixel[i] = ((2) * ((float)rand() / RAND_MAX)) -1;
        }
        //generate solution
        genSolution();
        //output
        for (int i = 0; i < 4; ++i) outFile << pixel[i] << "\n";
        for (int i = 0; i < 4; ++i) outFile << outLayer[i] << "\n";
        outFile.close();
    }
    return 0;
}
```

1.5.2 MACHINE LEARNING MODEL

```
#include<bits/stdc++.h>
using namespace std;
const float adjustMagnitude = 0.000001;

int dataCount;
float firstLayerWeight[4][4], secondLayerWeight[4][4];
float inLayer[4], firstLayer[4], secondLayer[4], outLayer[4];
float actualOutput[4];
float firstLayerSlope[4], secondLayerSlope[4], outLayerSlope[4], secondWeightSlope[4
][4], firstWeightSlope[4][4];
float preFirstLayer[4], preSecondLayer[4];    //before sigmoid

void inputWeight(){
    ifstream inFile;

    inFile.open("firstLayerWeight.txt");
    for (int i = 0; i < 4; ++i){
        for (int j = 0; j < 4; ++j){
            inFile >> firstLayerWeight[i][j];
        }
    }
    inFile.close();

    inFile.open("secondLayerWeight.txt");
    for (int i = 0; i < 4; ++i){
        for (int j = 0; j < 4; ++j){
            inFile >> secondLayerWeight[i][j];
        }
    }
    inFile.close();
}

string genFileName(int n){
    string name = "C:\\projects\\patternMatching\\Generate_Data\\data\\", number;
    //add trailing 0s
    int count = 0, tmp = n;  //number of digits of n
    while (tmp > 0) {
        tmp /= 10;
        ++count;
    }
    for (int i = 0; i < 10 - count; ++i) name += "0";
    //convert number to string
    stringstream ss;
    ss << n;
    ss >> number;

    name += number;
    name += ".txt";
    return name;
}
```

```
void inputData(int n){
    ifstream inFile;
    inFile.open( genFileName(n) );
    for (int i = 0; i < 4; ++i){
        inFile >> inLayer[i];
    }
    for (int i = 0; i < 4; ++i){
        inFile >> actualOutput[i];
    }
}

void outputWeight(){
    ofstream outFile;

    outFile.open("firstLayerWeight.txt");
    for (int i = 0; i < 4; ++i){
        for (int j = 0; j < 4; ++j){
            outFile << firstLayerWeight[i][j] << "\n";
        }
    }
    outFile.close();

    outFile.open("secondLayerWeight.txt");
    for (int i = 0; i < 4; ++i){
        for (int j = 0; j < 4; ++j){
            outFile << secondLayerWeight[i][j] << "\n";
        }
    }
    outFile.close();
}

void calculateResult(){
    //first layer
    for (int i = 0; i < 4; ++i){
        preFirstLayer[i] = 0;
        for (int j = 0; j < 4; ++j){
            preFirstLayer[i] += inLayer[j] * firstLayerWeight[j][i];
        }
        firstLayer[i] = tanh(preFirstLayer[i]);    //tanh squashing function
    }
    //second layer
    for (int i = 0; i < 4; ++i){
        preSecondLayer[i] = 0;
        for (int j = 0; j < 4; ++j){
            preSecondLayer[i] += firstLayer[j] * secondLayerWeight[j][i];
        }
        secondLayer[i] = tanh(preSecondLayer[i]);  //tanh squashing function
    }
    //output layer
    for (int i = 0; i < 4; ++i){
        outLayer[i] = 0;
        for (int j = 0; j < 8; ++j){
            outLayer[i] += abs(secondLayer[i]);
        }
    }
}

float sigmoid(float x){
    return 1/(1+exp(-x));
}

void adjustWeight(){
    //out layer slope
    for (int i = 0; i < 4; ++i){
        if (outLayer[i] > actualOutput[i]){
            outLayerSlope[i] = 1;
        }
        else{
            outLayerSlope[i] = -1;
        }
    }
    //second layer slope
    for (int i = 0; i < 4; ++i){
        if (secondLayer[i] > 0){
            secondLayerSlope[i] = 1;
        }
        else{
            secondLayerSlope[i] = -1;
        }
        secondLayerSlope[i] *= outLayerSlope[i];
    }
    //second layer weight
    for (int i = 0; i < 4; ++i){        //first layer
        for (int j = 0; j < 4; ++j){    //second layer
            secondWeightSlope[i][j] = sigmoid(preSecondLayer[j]) * (1 - sigmoid(preS
econdLayer[j])) * firstLayer[i];
            secondWeightSlope[i][j] *= secondLayerSlope[j];
            if (secondWeightSlope[i][j] > 0){   //aim is to minimize error -> adjust
 to opp of slope
                secondLayerWeight[i][j] -= adjustMagnitude;
                if (secondLayerWeight[i][j] < -1) secondLayerWeight[i][j] = -1;
            }
            else{
                secondLayerWeight[i][j] += adjustMagnitude;
                if (secondLayerWeight[i][j] > 1) secondLayerWeight[i][j] = 1;
            }
        }

    }
    //first layer slope
    float tmp;
    for (int i = 0; i < 4; ++i){        //first layer
        firstLayerSlope[i] = 0;
        for (int j = 0; j < 4; ++j){    //second layer
            tmp = sigmoid(preSecondLayer[j]) * (1 - sigmoid(preSecondLayer[j]));
            tmp *= secondLayerWeight[i][j] * secondLayerSlope[j];
            firstLayerSlope[i] += tmp;
        }
    }
    //first layer weight
    for (int i = 0; i < 4; ++i){        //input layer
        for (int j = 0; j < 4; ++j){    //first layer
            firstWeightSlope[i][j] = sigmoid(preFirstLayer[j]) * (1 - sigmoid(preFir
stLayer[j])) * inLayer[i];
            firstWeightSlope[i][j] *= firstLayerSlope[j];
            if (firstWeightSlope[i][j] > 0){   //aim is to minimize error -> adjust
to opp of slope
                firstLayerWeight[i][j] -= adjustMagnitude;
                if (firstLayerWeight[i][j] < -1) firstLayerWeight[i][j] = -1;
            }
            else{
                firstLayerWeight[i][j] += adjustMagnitude;
                if (firstLayerWeight[i][j] > 1) firstLayerWeight[i][j] = 1;
            }
        }
    }
}

int actualResult(){
    float mx = -5, type;
    for (int i = 0; i < 4; ++i){
        if (actualOutput[i] > mx){
            mx = actualOutput[i];
            type = i;
        }
    }
    return type;
}

int genResult(){
    float mx = -5, type;
    for (int i = 0; i < 4; ++i){
        if (outLayer[i] > mx){
            mx = outLayer[i];
            type = i;
        }
    }
    return type;
}

int main(){
    inputWeight();
    ifstream inFile;
    inFile.open("datacount.txt");
    inFile >> dataCount;
    inFile.close();
    int successCount = 0;
    for (int i = 1; i <= dataCount; ++i){
        inputData(i);
        calculateResult();
        if (actualResult() == genResult()) ++successCount;
        adjustWeight();
    }
    outputWeight();
    ofstream outFile;
    outFile.open("report.txt");
    outFile << "success trials: " << successCount << "\n";
    outFile.close();
    return 0;
}
```

1.6 Back Propagation Details



One node or edge of each layer is used in the following calculation. The differentiate values of other nodes or edge in the same layers can be obtained similarly. The edges and nodes are denoted as above for the sake of calculation.

By obtaining d(Error)/d(weight) of each edge, the weight can be adjusted in a magnitude which decreases error. For example, when d(Error)/d(weight) > 0, weight should be decreased such that the error decreases as well.

github.com/christycty

Denote actual values of output layer as a_1, a_2, a_3, a_4.

$$\text{Error} = |O_1 - a_1| + |O_2 - a_2| + |O_3 - a_3| + |O_4 - a_4|$$

$$\therefore \frac{d\text{Error}}{dO_1} = \begin{cases} 1 & \text{if } O_1 > a_1 \\ -1 & \text{otherwise} \end{cases}$$

$$\because O_1 = \text{abs } Q$$

$$\therefore \frac{dO_1}{dQ} = \begin{cases} 1 & \text{if } Q > 0 \\ -1 & \text{otherwise} \end{cases}$$

By chain rule,

$$\frac{d\text{Error}}{dQ} = \frac{dO_1}{dQ} \cdot \frac{d\text{Error}}{dO_1}$$

Denote $A \cdot AQ + B \cdot BQ + C \cdot CQ + D \cdot DQ$ as k_1. Denote sigmoid function as σ.

$$Q = \sigma(k_1).$$

$$\frac{dQ}{dk_1} = \sigma(k_1)[1 - \sigma(k_1)] \quad \text{and} \quad \frac{dk_1}{dAQ} = A$$

$$\therefore \frac{dQ}{dAQ} = \frac{dk_1}{dAQ} \cdot \frac{dQ}{dk_1} = \sigma(k_1)[1 - \sigma(k_1)] \cdot A$$

$$\frac{d\text{Error}}{dAQ} = \sigma(k_1)[1 - \sigma(k_1)] \cdot A \cdot \frac{d\text{Error}}{dQ}$$

$$A \cdot AQ = Q, \quad A \cdot AR = R, \quad A \cdot AS = S, \quad A \cdot AT = T$$

$$\therefore \frac{dQ}{dA} = AQ, \quad \frac{dR}{dA} = AR, \quad \frac{dS}{dA} = AS, \quad \frac{dT}{dA} = AT$$

$$\therefore \frac{d\text{Error}}{dA} = AQ \cdot \frac{d\text{Error}}{dQ} + AR \cdot \frac{d\text{Error}}{dR} + AS \cdot \frac{d\text{Error}}{dS} + AT \cdot \frac{d\text{Error}}{dT}$$

Denote $I_1 \cdot I_1A + I_2 \cdot I_2A + I_3 \cdot I_3A + I_4 \cdot I_4A$ as p_1.

$$A = \sigma(p_1)$$

$$\therefore \frac{dA}{dp_1} = \sigma(p_1)[1 - \sigma(p_1)] \quad \text{and} \quad \frac{dp_1}{dI_1A} = I_1$$

$$\therefore \frac{dA}{dI_1A} = \sigma(p_1)[1 - \sigma(p_1)] \cdot I_1$$

$$\frac{d\text{Error}}{dI_1A} = \sigma(p_1)[1 - \sigma(p_1)] \cdot I_1 \cdot \frac{d\text{Error}}{dA}$$