Neural Network I: Fundamental Theory and Applications

Project I

Project member: ①Atsushi Takamiya（s1260013）

②Tomonori Suzuki （s1260032）

③Taiki Watanabe （m5251152）

④Kaito Suzuki （m5251144）

⑤Kaito Iwamatsu （m5251115）

[ Part 1: 1](#_Toc69519461)

[a,b,c) 1](#_Toc69519462)

[d,e,f) 2](#_Toc69519463)

[From output with same initial weight, Perceptron learning rule is faster than Delta learning rule. 2](#_Toc69519464)

[Part1-program 2](#_Toc69519465)

[・Case delta learning rule (Major changes): 2](#_Toc69519466)

[・Case perceptron learning rule: 3](#_Toc69519467)

[ Part 2: 3](#_Toc69519468)

[a,b,c) 3](#_Toc69519469)

[d,e,f) 4](#_Toc69519470)

[Part2-program 5](#_Toc69519471)

[・Case 1 (Single layer NN with discrete neurons) 5](#_Toc69519472)

[・Case 2 (Single layer NN with continuous neurons) 5](#_Toc69519473)

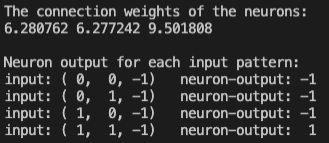
# Part 1:

## a,b,c)

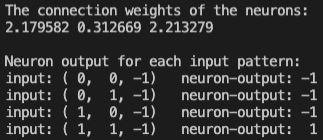
a) Train a neuron using our program to realize the AND gate.

b) delta learning rule, perceptron learning rule

c) Delta learning rule output



Perceptron learning rule output



c )Discussion:

・Weights are initialized randomly, so a number of cycles required varies.

・However, in case of binary classification with one neuron, it is faster to learn using the perceptron rule.

## d,e,f)

d) Because weights are initialized randomly, it cannot analyze Delta learning rule and Perceptron learning rule accurately.

e) Same initialized values set Delta learning rule and Perceptron learning rule and analyze results.

f)

テキスト

自動的に生成された説明

## From output with same initial weight, Perceptron learning rule is faster than Delta learning rule.

## Part1-program

### ・Case delta learning rule (Major changes):

int main() {

int i, p, q = 0;

double delta, Error = DBL\_MAX;

Initialization(); // 重み初期化

while (Error > desired\_error) {

q++;

for (Error = 0, p = 0; p < n\_sample; p++) {

FindOutput(p); // o = sigmoid(x[p][]に対して)

Error += 0.5 \* pow(d[p] - o, 2.0);

for (i = 0; i < I; i++){

delta = (d[p] - o) \* (1 - o \* o) / 2;

w[i] += eta \* delta \* x[p][i]; // 重みの更新

}

printf("Error in the %d-th learning cycle=%f\n", q, Error);

}

PrintResult();

return 0;

}

void FindOutput(int p) {

int i;

double temp = 0;

for (i = 0; i < I; i++){

temp += w[i] \* x[p][i];

}

o = sigmoid(temp);

}

### ・Case perceptron learning rule:

int main() {

int i, p, q = 0;

double delta, Error = DBL\_MAX;

Initialization(); // 重み初期化

while (Error > desired\_error) {

q++;

for (Error = 0, p = 0; p < n\_sample; p++) {

FindOutput(p);

Error += 0.5 \* pow(d[p] - o, 2.0);

double LearningSignal = eta \* (d[p] - o);

for (i = 0; i < I; i++) {

w[i] += LearningSignal \* x[p][i]; // 重みの更新

}

printf("Error in the %d-th learning cycle=%f\n", q, Error);

}

}

PrintResult();

return 0;

}

void FindOutput(int p) {

int i;

double temp = 0;

for (i = 0; i < I; i++) {

temp += w[i] \* x[p][i];

}

if(temp > 0)

o = 1;

else

o = -1;

}

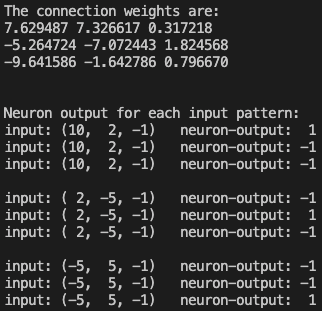
# Part 2:

## a,b,c)

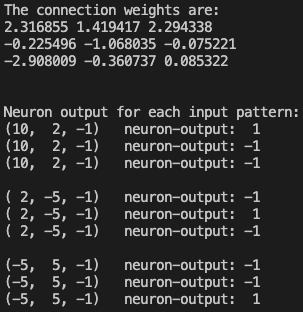
a) Extend the program written in the first step to learning of single layer neural networks.

b) Case1: perceptron learning rule, Case2: delta learning rule

c)Case1 output (Single layer NN with discrete neurons)



Case2 output (Single layer NN with continuous neurons)



c)Discussion:

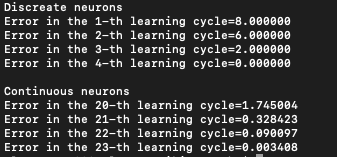
・Increasing the number of neurons reduces the difference of the number of cycles between the two rules.

## d,e,f)

d) In Part2, there are not a lot of the learning cycle difference between Delta learning rule and Perceptron learning rule. We analyzed the factor is the value of “desired\_error” and “the number of neurons”.

e) The value of “desired\_error” in Part2 changed and compared with Part1. (The value of desired\_error : 01 -> 0.01)

f)



From the output, the value of “desired\_error” is not the main factor. Thus, we concluded the main factor is the number of neurons.

## Part2-program

### ・Case 1 (Single layer NN with discrete neurons)

int main() {

int i, j, p, q = 0;

double Error = DBL\_MAX;

double delta;

Initialization();

while (Error > desired\_error) {

q++;

for (Error = 0, p = 0; p < n\_sample; p++) {

FindOutput(p);

for (i = 0; i < R; i++) {

Error += 0.5 \* pow(d[p][i] - o[i], 2.0);

double LearningSignal = eta \* (d[p][i] - o[i]);

for (j = 0; j < N; j++) {

w[i][j] += LearningSignal \* x[p][j];

}

}

}

printf("Error in the %d-th learning cycle=%f\n", q, Error);

}

PrintResult();

return 0;

}

void FindOutput(int p) {

int i, j;

double temp;

for (i = 0; i < R; i++) {

temp = 0;

for (j = 0; j < N; j++) {

temp += w[i][j] \* x[p][j];

}

if (temp > 0)

o[i] = 1;

else

o[i] = -1;

}

}

### ・Case 2 (Single layer NN with continuous neurons)

int main() {

int i, j, p, q = 0;

double Error = DBL\_MAX;

double delta;

Initialization();

while (Error > desired\_error) {

q++;

for (Error = 0, p = 0; p < n\_sample; p++) {

FindOutput(p);

for (i = 0; i < R; i++) {

Error += 0.5 \* pow(d[p][i] - o[i], 2.0);

delta = (d[p][i] - o[i]) \* (1 - o[i] \* o[i]) / 2;

for (j = 0; j < N; j++) {

w[i][j] += eta \* delta \* x[p][j];

}

}

}

printf("Error in the %d-th learning cycle=%f\n", q, Error);

}

PrintResult();

return 0;

}

void FindOutput(int p) {

int i, j;

double temp;

for (i = 0; i < R; i++) {

temp = 0;

for (j = 0; j < N; j++) {

temp += w[i][j] \* x[p][j];

}

o[i] = sigmoid(temp);

}

}