I. 수치미분(Numerical Derivative)

```
import warnings
warnings.filterwarnings('ignore')
```

1) Import numpy

```
import numpy as np
```

2) gradient( ) 함수 정의

- 다변수 함수의 수치미분

```
def gradient(machine, param):

    if param.ndim == 1:
        temp_param = param
        delta = 0.00005
        learned_param = np.zeros(param.shape)

        for index in range(len(param)):
            target_param = float(temp_param[index])
            temp_param[index] = target_param + delta
            param_plus_delta = machine(temp_param)
            temp_param[index] = target_param - delta
            param_minus_delta = machine(temp_param)
            learned_param[index] = (param_plus_delta - param_minus_delta ) / W
                                                                (2 * delta)
            temp_param[index] = target_param

        return learned_param


    elif param.ndim == 2:
        temp_param = param
        delta = 0.00005
        learned_param = np.zeros(param.shape)

        rows = param.shape[0]
        columns = param.shape[1]

        for row in range(rows):
            for column in range(columns):
                target_param = float(temp_param[row, column])
                temp_param[row, column] = target_param + delta
                param_plus_delta = machine(temp_param)
```

```
                temp_param[row, column] = target_param - delta
                param_minus_delta = machine(temp_param)
                learned_param[row, column] = (param_plus_delta - W
                                              param_minus_delta) / (2 * delta)
                temp_param[row, column] = target_param

        return learned_param
```

II. Logic Gate( ) - 'AND', 'OR', 'NAND'

1) sigmoid( ) 함수 정의

```
import numpy as np

def sigmoid(x):
    y_hat = 1 / (1 + np.exp(-x))
    return y_hat
```

2) LogicGate 클래스 선언

```
class LogicGate:

    def __init__(self, gate_Type, X_input, y_output):

# gate_Type 문자열 지정 Member
        self.Type = gate_Type

# X_input, y_output Member 초기화
        self.X_input = X_input.reshape(4, 2)
        self.y_output = y_output.reshape(4, 1)

# W, b Member 초기화
        self.W = np.random.rand(2, 1)
        self.b = np.random.rand(1)

# learning_rate Member 지정
        self.learning_rate = 0.01

# Cost_Function(CEE) Method
    def cost_func(self):
        z = np.dot(self.X_input, self.W) + self.b
        y_hat = sigmoid(z)
        delta = 0.00001
        return -np.sum(self.y_output * np.log(y_hat + delta) + W
                       (1 - self.y_output) * np.log((1 - y_hat) + delta))

# Learning Method
```

```
    def learn(self):
        machine = lambda x : self.cost_func()
        print("Initial Cost = ", self.cost_func())

        for step in  range(10001):
            self.W = self.W - self.learning_rate * gradient(machine, self.W)
            self.b = self.b - self.learning_rate * gradient(machine, self.b)

            if (step % 1000 == 0):
                print("Step = ", step, "Cost = ", self.cost_func())

# Predict Method
    def predict(self, input_data):

        z = np.dot(input_data, self.W) + self.b
        y_prob = sigmoid(z)

        if y_prob > 0.5:
            result = 1
        else:
            result = 0

        return y_prob, result
```

3) AND_Gate

- X_input, y_output 지정

```
X_input  = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_output = np.array([0, 0, 0, 1])
```

- AND_Gate 객체 생성 및 학습

```
AND_Gate = LogicGate("AND_GATE", X_input, y_output)

AND_Gate.learn()
```

```
Initial Cost =  4.471843925599968
Step =  0 Cost =  4.419298893028864
Step =  1000 Cost =  1.0472468570522806
Step =  2000 Cost =  0.6767942670566427
Step =  3000 Cost =  0.5005889266046376
Step =  4000 Cost =  0.396108617293048
Step =  5000 Cost =  0.3269307562494621
Step =  6000 Cost =  0.2778363169993137
Step =  7000 Cost =  0.2412566491283161
Step =  8000 Cost =  0.2129918627337813
Step =  9000 Cost =  0.1905243127148315
Step =  10000 Cost =  0.17225434388461153
```

- AND_Gate 테스트

```
print(AND_Gate.Type, "₩n")

test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

for input_data in test_data:
    (sigmoid_val, logical_val) = AND_Gate.predict(input_data)
    print(input_data, " = ", logical_val)
```

```
 AND_GATE

 [0 0]  =  0
 [0 1]  =  0
 [1 0]  =  0
 [1 1]  =  1
```

4) OR_Gate

- X_input, y_output

```
X_input  = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_output = np.array([0, 1, 1, 1])
```

- OR_Gate 객체 생성 및 학습

```
OR_Gate = LogicGate("OR_GATE", X_input, y_output)

OR_Gate.learn()
```

```
 Initial Cost =  1.8146148690661121
 Step =  0 Cost =  1.8109010270686765
 Step =  1000 Cost =  0.7069996326661037
 Step =  2000 Cost =  0.4265310856157141
 Step =  3000 Cost =  0.30100202135004567
 Step =  4000 Cost =  0.23096255917990757
 Step =  5000 Cost =  0.1866801295107886
 Step =  6000 Cost =  0.15631132194118408
 Step =  7000 Cost =  0.13425952066035818
 Step =  8000 Cost =  0.11755450555713279
 Step =  9000 Cost =  0.10448064667329153
 Step =  10000 Cost =  0.0939807759693659
```

- OR_Gate 테스트

```
print(OR_Gate.Type, "₩n")

test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

for input_data in test_data:
    (sigmoid_val, logical_val) = OR_Gate.predict(input_data)
    print(input_data, " = ", logical_val)
```

```
OR_GATE

[0 0]  =  0
[0 1]  =  1
[1 0]  =  1
[1 1]  =  1
```

5) NAND_Gate

- X_input, y_output

```
X_input  = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_output = np.array([1, 1, 1, 0])
```

- NAND_Gate 객체 생성 및 학습

```
NAND_Gate = LogicGate("NAND_GATE", X_input, y_output)

NAND_Gate.learn()
```

```
Initial Cost =  2.812320147595384
Step =  0 Cost =  2.8066384176552743
Step =  1000 Cost =  1.0577543706191264
Step =  2000 Cost =  0.6809511023598829
Step =  3000 Cost =  0.5028782868060374
Step =  4000 Cost =  0.39756118680033053
Step =  5000 Cost =  0.3279324336972359
Step =  6000 Cost =  0.2785671699904667
Step =  7000 Cost =  0.24181236797911926
Step =  8000 Cost =  0.2134280051214637
Step =  9000 Cost =  0.1908753120825363
Step =  10000 Cost =  0.1725426502585749
```

- NAND_Gate 테스트

```
print(NAND_Gate.Type, "₩n")

test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

for input_data in test_data:
    (sigmoid_val, logical_val) = NAND_Gate.predict(input_data)
    print(input_data, " = ", logical_val)
```

```
NAND_GATE

[0 0]  =  1
[0 1]  =  1
[1 0]  =  1
[1 1]  =  0
```

III. XOR_Gate Issue

1) XOR_Gate Failure

- X_input, y_output

```
X_input  = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_output = np.array([0, 1, 1, 0])
```

- XOR_Gate 객체 생성 및 학습

```
XOR_Gate = LogicGate("XOR_GATE", X_input, y_output)

XOR_Gate.learn()
```

```
Initial Cost =  3.7071999250561998
Step =  0 Cost =  3.684778408199767
Step =  1000 Cost =  2.7726143000500776
Step =  2000 Cost =  2.7725100478386473
Step =  3000 Cost =  2.7725087583311407
Step =  4000 Cost =  2.77250872440606
Step =  5000 Cost =  2.772508723097248
Step =  6000 Cost =  2.7725087230422227
Step =  7000 Cost =  2.7725087230398757
Step =  8000 Cost =  2.772508723039775
Step =  9000 Cost =  2.7725087230397705
Step =  10000 Cost =  2.7725087230397705
```

- XOR_Gate 테스트

```
print(XOR_Gate.Type, "₩n")

test_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

for input_data in test_data:
    (sigmoid_val, logical_val) = XOR_Gate.predict(input_data)
    print(input_data, " = ", logical_val)
```

```
XOR_GATE

[0 0]  =  1
[0 1]  =  1
[1 0]  =  1
[1 1]  =  0
```

2) XOR_Gate Succeed

- XOR를 (NAND + OR) 계층 및 AND 계층의 조합으로 연산
- 이전 학습된 Parametrer로 XOR 수행

```
input_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

HL1_1 = []    # NAND 출력
HL1_2 = []    # OR   출력

new_input_data = []  # AND      입력
final_output = []    # AND(XOR) 출력

for index in range(len(input_data)):

    HL1_1 = NAND_Gate.predict(input_data[index])  # NAND 출력
    HL1_2 = OR_Gate.predict(input_data[index])    # OR   출력

    new_input_data.append(HL1_1[-1])    # AND 입력
    new_input_data.append(HL1_2[-1])    # AND 입력

    (sigmoid_val, logical_val) = AND_Gate.predict(np.array(new_input_data))

    final_output.append(logical_val)    # AND(XOR) 출력
    new_input_data = []                 # AND 입력 초기화
```

```
print(XOR_Gate.Type, "₩n")

for index in range(len(input_data)):
    print(input_data[index], " = ", final_output[index])
```

```
  XOR_GATE

  [0 0]  =  0
  [0 1]  =  1
  [1 0]  =  1
  [1 1]  =  0
```

#

#

#

THE END

#

#

#