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 ) / (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 - 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
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) + (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 =  5.029465611572384
Step =  0 Cost =  4.963991581328153
Step =  1000 Cost =  1.006161601408068
Step =  2000 Cost =  0.6599060137088674
Step =  3000 Cost =  0.4911978505636255
Step =  4000 Cost =  0.39012187510894786
Step =  5000 Cost =  0.3227899351783622
Step =  6000 Cost =  0.2748086115692702
Step =  7000 Cost =  0.23895077907877427
Step =  8000 Cost =  0.21117988543371188
Step =  9000 Cost =  0.189064593355241
Step =  10000 Cost =  0.17105435236612593
```

- 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 =  1.7268148014531293
Step =  0 Cost =  1.7234771388181978
Step =  1000 Cost =  0.6910070618854313
Step =  2000 Cost =  0.420275580313862
Step =  3000 Cost =  0.2977549562927667
Step =  4000 Cost =  0.22900112186791557
Step =  5000 Cost =  0.1853765938565975
Step =  6000 Cost =  0.1553862400633751
Step =  7000 Cost =  0.13357085618218967
Step =  8000 Cost =  0.11702286695213433
Step =  9000 Cost =  0.104058360240497
Step =  10000 Cost =  0.09363756415897866
```

- 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 =  3.187925848876683
Step =  0 Cost =  3.1777075078037247
Step =  1000 Cost =  1.0794963458611433
Step =  2000 Cost =  0.6896988419753853
Step =  3000 Cost =  0.5076821159385578
Step =  4000 Cost =  0.4006013945299704
Step =  5000 Cost =  0.33002523494245006
Step =  6000 Cost =  0.28009220005043634
Step =  7000 Cost =  0.24297084461275548
Step =  8000 Cost =  0.21433652578513923
Step =  9000 Cost =  0.19160603192470932
Step =  10000 Cost =  0.17314255659483982
```

- 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 =  3.2457413587637394
Step =  0 Cost =  3.2340649830701027
Step =  1000 Cost =  2.774000534320967
Step =  2000 Cost =  2.772571708835415
Step =  3000 Cost =  2.772511411226759
Step =  4000 Cost =  2.7725088379747804
Step =  5000 Cost =  2.772508727955279
Step =  6000 Cost =  2.772508723250006
Step =  7000 Cost =  2.7725087230487624
Step =  8000 Cost =  2.772508723040155
Step =  9000 Cost =  2.7725087230397873
Step =  10000 Cost =  2.772508723039772
```

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

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
```

3) XOR_Gate Learning

(1) XOR_Gate Class

```
class XOR_Gate:

    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_1, b_1 Member 초기화
        self.W_1 = np.random.rand(2, 2)
        self.b_1 = np.random.rand(2)

# W_2, b_2 Member 초기화
        self.W_2 = np.random.rand(2, 1)
        self.b_2 = np.random.rand(1)

# learning_rate Member 지정
        self.learning_rate = 0.01

# Cost_Function(CEE) Method
    def cost_func(self):

        z_1 = np.dot(self.X_input, self.W_1) + self.b_1     # Hidden Layer
        a_1 = sigmoid(z_1)

        z_2 = np.dot(a_1, self.W_2) + self.b_2            # Output Layer
        y_hat = sigmoid(z_2)

        delta = 0.00001
        return -np.sum(self.y_output * np.log(y_hat + delta) + (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(20001):
            self.W_1 = self.W_1- self.learning_rate * gradient(machine, self.W_1)
            self.b_1 = self.b_1 - self.learning_rate * gradient(machine, self.b_1)

            self.W_2 = self.W_2 - self.learning_rate * gradient(machine, self.W_2)
            self.b_2 = self.b_2 - self.learning_rate * gradient(machine, self.b_2)

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

# Predict Method
    def predict(self, input_data):

        z_1 = np.dot(input_data, self.W_1) + self.b_1     # Hidden Layer
        a_1 = sigmoid(z_1)

        z_2 = np.dot(a_1, self.W_2) + self.b_2            # Output Layer
        y_prob = sigmoid(z_2)
        if y_prob > 0.5:
            result = 1
        else:
            result = 0

        return y_prob, result
```

(2) X_input, y_output

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

(3) XOR_Gate_2.learn( )

```
XOR_Gate_2 = XOR_Gate('XOR_GATE', X_input, y_output)

XOR_Gate_2.learn()
```

```
Initial Cost =  3.2766959441649743
Step =  0 Cost =  3.2579992632013948
Step =  1000 Cost =  2.763096651627254
Step =  2000 Cost =  2.7492321954128167
Step =  3000 Cost =  2.7171262461817194
Step =  4000 Cost =  2.6429984923597365
Step =  5000 Cost =  2.4998912097728785
Step =  6000 Cost =  2.309773609119135
Step =  7000 Cost =  2.1293671723462224
Step =  8000 Cost =  1.960109682659135
Step =  9000 Cost =  1.7489719548246685
Step =  10000 Cost =  1.304838030339951
Step =  11000 Cost =  0.829652452124708
Step =  12000 Cost =  0.5372334483490177
Step =  13000 Cost =  0.3782842267443513
Step =  14000 Cost =  0.28618780721700915
Step =  15000 Cost =  0.22800180394312214
Step =  16000 Cost =  0.18850679335909
Step =  17000 Cost =  0.1601724265921784
Step =  18000 Cost =  0.13895638512554417
Step =  19000 Cost =  0.12252723038681454
Step =  20000 Cost =  0.10945752918469978
```

(4) XOR_Gate_2.predict( )

```
print(XOR_Gate_2.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_2.predict(input_data)
    print(input_data, ' = ', logical_val)
```

```
XOR_GATE

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

#

#

#

THE END

#

#

#