수치최적화 과제

인공지능사이버보안학과
2021271314 김건희

#Start

```
import numpy as np
import csv
from scipy.stats import norm
import random
```

0.0s

```
def sigmoid(x):
    return round(1 / (1+np.exp(-x)), 3)
```

0.0s

```
def derivative_sig(x):
    return round(sigmoid(x) * (1-sigmoid(x)), 3)
```

0.0s

-> 코드를 사용하여 입력 값을 시그모이드 함수에 적용하여 0과 1 사이의 값으로 변환

-> 시그모이드 함수의 도함수 값을 계산 가능함. 시그모이드 함수의 도함수는 주로 역전파(backpropagation) 알고리즘에서 사용되며, 신경망의 가중치 업데이트에 사용됨

Data read

```
#data parsing
raw_file = list()
file_loc = "/mnt/c/Users/82102/downloads/data.csv"
with open(file_loc, 'r', encoding="UTF-8") as f:
    for i in csv.reader(f):
        print(i)
        raw_file.append(i)
```

-> 이 코드는 "data.csv" 파일을 열고, 각 행을 출력하고 raw_file 리스트에 저장

Data extract

```
#data extract
#input: 2-dimension array from csv
#output: 3-dimension array with 2-dimension data
data = list()
label = list()

row = 0

data_row = 3
data_col = 4
data_num = 66

for i in range(data_num):
    col_temp = list()
    for j in range(data_col):
        #print(raw_file[j+1][row:row+3])
        col_temp.append(raw_file[j+1][row:row+3])

        #print(col_temp)
    row = row + 4
    data.append(col_temp)
    label.append(raw_file[5][row-1])
```

->2차원 배열인 raw_file에서 데이터를 추출하여 2+1차원 배열인 data에 저장

-> 해당 데이터에 대한 레이블을 label에 저장

-> 데이터 추출 과정에서는 raw_file의 특정 범위를 슬라이싱하여 데이터를 추출

-> 추출한 데이터를 data 리스트에 추가

-> 또한, 해당 데이터에 대한 레이블을 추출하여 label 리스트에 추가

Define network parameter

```
#data extract
#input: 2-dimension array from csv
#output: 3-dimension array with 2-dimension data
data = list()
label = list()

row = 0

data_row = 3
data_col = 4
data_num = 66

for i in range(data_num):
    col_temp = list()
    for j in range(data_col):
        #print(raw_file[j+1][row:row+3])
        col_temp.append(raw_file[j+1][row:row+3])

        #print(col_temp)
    row = row + 4
    data.append(col_temp)
    label.append(raw_file[5][row-1])
```

-> 훈련 데이터와 테스트 데이터의 개수 및 인덱스를 정의

-> 네트워크의 매개변수를 설정

-> 훈련 데이터의 인덱스는 앞 64개를 선택함

-> 테스트 데이터의 인덱스는 훈련 데이터를 제외한 나머지 인덱스로 구성

Train set과 Test set 분할

```
train_data = list()
test_data = list()
for i in all_index:
    if i in test_index:
        test_data.append(data[i])
    else:
        train_data.append(data[i])
```

->전체 데이터를 훈련 데이터와 테스트 데이터로 나누는 작업을 수행

->all_index 리스트를 기반으로 전체 데이터를 훈련 데이터와 테스트 데이터로 분할하여 각각 train_data와 test_data 리스트에 저장

확률 밀도 함수(PDF)를 기반으로 난수를 생성하는 함수를 구현

```python
def my_pdf(x):
    return np.exp(-0.5 * x**2) / np.sqrt(2 * np.pi)

# Generate a random number from the PDF
def ret_val():
    rand = np.random.choice(np.linspace(-10, 10, 1000), p=my_pdf(np.linspace(-10, 10, 1000))/np.sum(my_pdf(np.linspace(-10, 10, 1000))))
    rand = round(rand, 3)
    return rand
```

-> 주어진 확률 분포 함수를 기반으로 임의의 숫자를 생성하는 함수를 정의

-> ret_val() 함수는 주어진 확률 분포에 따라 -10부터 10까지의 범위에서 임의의 숫자를 생성하여 반환

-> 생성된 숫자는 확률 밀도 함수를 기반으로 선택되며, 소수점 셋째 자리까지 반올림된 값을 가진다

#Output dimension

```python
#output dimension: n_node * n_row * n_col
def update(n_node:int, n_row:int, n_col:int, init=True, prev_weight:list=None, back_weight:list=None):
    weight = list()
    if init:
        for i in range(n_node):
            col_temp = list()
            for j in range(n_col):
                row_temp = list()
                for k in range(n_row):
                    row_temp.append(ret_val())
                col_temp.append(row_temp)
            weight.append(col_temp)
    else:
        for i in range(n_node):
            col_temp = list()
            for j in range(n_col):
                row_temp = list()
                for k in range(n_row):
                    row_temp.append(prev_weight[i][j][k]-learning_rate*back_weight[i][j][k])
                col_temp.append(row_temp)
            weight.append(col_temp)
    weight = np.array(weight)
    return weight
```

-> 신경망의 가중치 업데이트를 수행하는 함수를 정의하는 것

-> update 함수는 초기화 여부에 따라 가중치를 생성하거나 이전 가중치를 업데이트하여 반환한다

-> update 함수는 초기화 여부에 따라 새로운 가중치를 생성하거나 이전 가중치를 업데이트하여 반환하는 역할함

```python
#forward propagation
def forw_prop(prev_n_w=None, prev_n_b=None, prev_o_w=None, prev_o_b=None, b_node_weight=None, b_node_bias=None, b_out_weight=None, b_out_bias=None, data_num=train_num):
    z2s = list()
    a2s = list()
    da2s = list()
    z3s = list()
    a3s = list()
    da3s = list()
    ca3s = list()
    d3s = list()
    swd3s = list()
    d2s = list()

    cost_sum = 0

    if prev_n_w is None and prev_n_b is None and prev_o_w is None and prev_o_b is None:
        print("Initialize weight")
        node_weight = update(node_num, 3, 4)
        node_bias = update(node_num, 1, 1)
        out_weight = update(out_num, 3, 1)
        out_bias = update(out_num, 1, 1)
        print_matrix("node weight",node_weight)
        print_matrix("node bias", node_bias)
        print_matrix("output weight", out_weight)
        print_matrix("output bias", out_bias)
    else:
        print("Update weight")
        node_weight = update(node_num, 3, 4, False, prev_n_w, b_node_weight)
        node_bias = update(node_num, 1, 1, False, prev_n_b, b_node_bias)
        out_weight = update(out_num, 3, 1, False, prev_o_w, b_out_weight)
        out_bias = update(out_num, 1, 1, False, prev_o_b, b_out_bias)
        print_matrix("node weight",node_weight)
        print_matrix("node bias", node_bias)
        print_matrix("output weight", out_weight)
        print_matrix("output bias", out_bias)
```

#forward propagation

-> 주어진 가중치와 편향을 사용하여 입력 데이터를 네트워크에 통과시켜 출력값을 계산

1. 가중치 및 편향 값의 초기화 또는 업데이트: prev_n_w, prev_n_b, prev_o_w, prev_o_b 값이 None인 경우 초기화되며, 그렇지 않은 경우에는 prev_n_w, prev_n_b, prev_o_w, prev_o_b, b_node_weight, b_node_bias, b_out_weight, b_out_bias 값을 사용하여 업데이트된다.

```python
if data_num == train_num:
    index = train_index
else:
    index = test_index

for i in range(data_num):
    z2 = list()
    a2 = list()
    da2 = list()
    z3 = list()
    a3 = list()
    da3 = list()
    ca3 = list()
    d3 = list()
    swd3 = list()
    d2 = list()
    cost = 0

    if label[i] == 0:
        ans = [1, 0]
    else:
        ans = [0, 1]

    for j in range(node_num):
        sum = 0
        mul = 0
        for t1 in range(data_col):
            for t2 in range(data_row):
                temp = data[index[i]][t1][t2]*node_weight[j][t1][t2]
                mul = mul + temp
        sum = mul + node_bias[j][0][0]
        z2.append(round(sum, 3))
        a2.append(round(sigmoid(z2[j]), 3))
        da2.append(round(derivative_sig(z2[j]), 3))
    for j in range(out_num):
        sum = 0
        mul = 0
        for t1 in range(node_num):
            temp = out_weight[j][0][t1] * a2[t1]
            mul = mul + temp
        sum = mul + out_bias[j][0][0]
        z3.append(round(sum, 3))
        a3.append(round(sigmoid(sum), 3))
        da3.append(round(derivative_sig(sum), 3))
        ca3.append(round(a3[j]-ans[j], 3))
        d3.append(round(da3[j]*ca3[j], 3))
        cost = cost + (ans[j] - a3[j])**2
    cost = cost / 2
    cost = round(cost, 3)
    for j in range(out_num):
        temp = 0
        for k in range(node_num):
            temp = temp + out_weight[j][0][k] * d3[j]
            swd3.append(temp)
    for t in range(len(swd3)):
        swd3[t] = round(swd3[t], 3)

    for j in range(node_num):
        d2.append(round(da2[j]*swd3[j], 3))
    cost_sum = cost_sum + cost
    cost_sum = round(cost_sum, 3)
    z2s.append(z2)
    a2s.append(a2)
    da2s.append(da2)
    z3s.append(z3)
    a3s.append(a3)
    da3s.append(da3)
    ca3s.append(ca3)
    d3s.append(d3)
    swd3s.append(swd3)
    d2s.append(d2)

return (z2s, a2s, da2s, z3s, a3s, da3s, ca3s, d3s, swd3s, d2s, cost_sum), (node_weight, node_bias, out_weight, out_bias)
```

2. train과 test여부에 따라 인덱스를 선택 (train_num인 경우 train_index를, 그렇지 않은 경우 test_index를 사용).

3. 각 데이터에 대해 순전파 과정을 수행

-> 입력 데이터를 가중치와 편향을 사용하여 출력을 계산

-> 활성화 함수(sigmoid)를 적용하여 각 레이어의 출력값을 계산

-> 손실 함수의 값(cost)을 계산

-> 역전파의 출력층(d3)을 계산

-> 역전파의 은닉층(d2)을 계산

-> 중간 결과 및 오차 값을 저장

4. 중간 결과 및 업데이트된 가중치, 편향 값을 반환

Backpropagation

```python
def back_prop(a2, d2, d3, data_num=train_num):
    b_node_bias = np.array(d2).sum(axis=0)
    b_output_bias = np.array(d3).sum(axis=0)

    b_node_weight = list()
    b_output_weight = list()

    if data_num == train_num:
        index = train_index
    else:
        index = test_index

    for i in range(data_num):
        temp = list()
        temp.append(np.array(data[index[i]])*np.array(d2[i][0]))
        temp.append(np.array(data[index[i]])*np.array(d2[i][1]))
        temp.append(np.array(data[index[i]])*np.array(d2[i][2]))
        b_node_weight.append(temp)
        b_output_weight.append(np.dot(np.array(d3).T, a2))

    sum_b_node_weight = 0
    for i in range(len(b_node_weight)):
        sum_b_node_weight = b_node_weight[i]

    sum_b_output_weight = 0
    for i in range(len(b_output_weight)):
        sum_b_output_weight = b_output_weight[i]

    return (sum_b_node_weight, b_node_bias, sum_b_output_weight, b_output_bias)
```

1. 중간 레이어의 편향(b_node_bias)을 계산한다.

d2를 열끼리 더하여 각 노드의 편향을 얻는다.

2. 출력 레이어의 편향(b_output_bias)을 계산한다. D3를 열끼리 더하여 출력 레이어의 편향을 얻는다.

3. 각 데이터에 대해 중간 레이어의 가중치 변화량(b_node_weight)을 계산한다. 데이터와 d2를 요소별로 곱하여 가중치 변화량을 얻는다.

4. 각 데이터에 대해 출력 레이어의 가중치 변화량(b_output_weight)을 계산한다. D3와 a2의 전치(transpose)의 곱을 통해 가중치 변화량을 얻는다.

5. 역전파 층의 총합(sum_b_node_weight, sum_b_output_weight)을 계산한다. 각각 b_node_weight와 b_output_weight의 총합을 구한다.

6. 가중치 변화량과 편향 변화를 반환한다.

```python
def network(dataset:int):
    b_node_weight, b_node_bias, b_out_weight, b_out_bias = None, None, None, None
    node_weight, node_bias, out_weight, out_bias = None, None, None, None
    for i in range(epoch):
        print("#%d epoch" %(i+1))

        (z2, a2, da2, z3, a3, da3, ca3, d3, swd3, d2, cost_sum), (node_weight, node_bias, out_weight, out_bias)  = forw_prop(node_weight, node_bias, out_weight, out_bias, b_node_weight, b_node_bias, b_out_weight, b_out_bi
        print("\n")
        print("z2", z2)
        print("a2", a2)
        print("da2", da2)
        print("z3", z3)
        print("a3", a3)
        print("da3", da3)
        print("ca3", ca3)
        print("d3", d3)
        print("swd3", swd3)
        print("d2", d2)
        print("Cost", cost_sum)
        b_node_weight, b_node_bias, b_out_weight, b_out_bias = back_prop(a2, d2, d3, dataset)
        print()
        print("Backpropagation weight")
        print_matrix("node weight", b_node_weight)
        print_matrix("node bias", b_node_bias)
        print_matrix("output weight", b_out_weight)
        print_matrix("output bias", b_out_bias)
        print("\n")
```

#신경망의 학습 과정

-> 주어진 데이터셋에 대해 신경망을 학습하는 과정을 구현하고, 각 epoch마다 중간 결과와 가중치, 편향의 변화량을 출력

1. 초기에 가중치와 편향의 값을 None으로 설정합니다.

2. 지정된 epoch 횟수만큼 반복합니다.

-> 순전파(forward propagation)를 수행하여 중간 결과와 업데이트된 가중치, 편향 값을 얻습니다.

-> 중간 결과와 가중치, 편향의 변화량을 출력합니다.

-> 역전파(backpropagation)를 수행하여 가중치와 편향의 변화량을 얻습니다.

-> 가중치와 편향의 변화를 출력합니다.

network(train_num)

✓ 1.1s

```
#1 epoch
Initialize weight

node weight
1 [[-0.611  1.311 -0.871]
 [ 0.19  -0.39  -0.27 ]
 [ 0.05   0.951  0.45 ]
 [ 1.051  0.791 -0.45 ]]
2 [[ 1.211  1.071  1.091]
 [ 0.49   0.33  -0.531]
 [-0.23  -3.033 -1.351]
 [-0.991  0.09  -0.13 ]]
3 [[ 2.272 -0.27   0.15 ]
 [-0.13  -1.191 -0.751]
 [-0.33   0.931  0.11 ]
 [ 1.672  0.41   0.831]]

node bias
1 [[-0.731]]
2 [[-0.591]]
3 [[-0.17]]

output weight
1 [[-0.631  1.311 -1.712]]
2 [[-0.13   0.07   0.591]]
...
1 0.01800000000000001
2 -0.01400000000000005
```

network(test_num)

✓ 0.2s

```
#1 epoch
Initialize weight

node weight
1 [[-0.831 -0.891  1.131]
 [ 0.991  0.891  1.431]
 [ 0.17   1.051  0.07 ]
 [-1.532 -0.611 -0.31 ]]
2 [[ 0.15  -0.21  -0.991]
 [-0.27   1.131 -0.891]
 [-1.351 -0.37  -1.592]
 [ 1.291  2.032 -0.33 ]]
3 [[-0.45  -0.531 -1.171]
 [-1.351  0.811  0.47 ]
 [-0.791  0.791 -0.01 ]
 [-1.191  0.671  0.47 ]]

node bias
1 [[0.23]]
2 [[0.811]]
3 [[0.711]]

output weight
1 [[ 0.951 -0.971 -0.15 ]]
2 [[-0.03  -2.412  1.672]]
...
1 0.044
2 -0.03
```

결과물 출력

Network(train_num)

-> 각 epoch마다 순전파와 역전파가 수행되고, 중간 결과와 가중치/편향의 변화가 출력

-> 신경망이 학습을 진행하며 손실 함수 값이 감소하고, 가중치/편향이 업데이트되는 것을 확인가능

Network(test_num)

->각 epoch마다 순전파와 역전파가 수행되고, 중간 결과와 가중치/편향의 변화가 출력

-> 신경망이 평가 데이터셋에 대한 예측을 수행하고, 손실 함수 값을 확인가능