Deep Learning

Assignment#2

2019028313

Kim Hyewon

1. Source code for model & training

[parameter 및 model 선언]

```
import numpy as np
import random
from math import exp,log
import matplotlib.pyplot as plt
import pandas as pd
```

[9] ✓ 0.4s

```
# Logical  Gate 학습을 위한 data
X = np.array([(0,0),(1,0),(0,1),(1,1)])
Y_AND = np.array([0,0,0,1])
Y_OR = np.array([0,1,1,1])
Y_XOR = np.array([0,1,1,0])
epoch_num = 10000
learning_rate = 0.1
# learning_rate = 0.01
# learning_rate = 0.001
```

[10] ✓ 0.3s

```
class logistic_regression_model():
    def __init__(self):
        self.w = np.random.normal(size=2)
        self.b = np.random.normal(size=1)

    def sigmoid(self,z):
        return 1/(1+exp(-z))

    def predict(self, x):
        z = np.inner(self.w, x) + self.b
        a = self.sigmoid(z)
        return a
```

[11] ✓ 0.3s

[train function]

각 parameter에 대한 cost function을 계산하여 parameter를 갱신한다.

```
def train(X, Y, model, lr = 0.1):
    dw = np.array([0.0,0.0])
    db = 0.0
    m = len(X)
    cost = 0.0

    for x,y in zip(X, Y):
        a = model.predict(x)
        if y == 1:
            cost -= log(a)
        else:
            cost -= log(1-a)

        dw += (a-y)*x
        db += (a-y)

    cost /= m
    model.w -= lr * dw/m
    model.b -= lr*db/m
    return cost
```

[4] ✓ 0.3s

[AND, OR, XOR Gate]

앞에서 numpy배열로 선언해둔 output(Y_AND, Y_OR, Y_XOR)을 이용하여 각각의 Logical gate를 학습시킨다.

```
def AND(model):
    cost_list = []
    for epoch in range(epoch_num):
        cost = train(X, Y_AND, model, learning_rate)
        cost_list.append(cost)
    return cost_list

def OR(model):
    cost_list = []
    for epoch in range(epoch_num):
        cost = train(X, Y_OR, model, learning_rate)
        cost_list.append(cost)
    return cost_list

def XOR(model):
    cost_list = []
    for epoch in range(epoch_num):
        cost = train(X, Y_XOR, model, learning_rate)
        cost_list.append(cost)
    return cost_list
```

[13] ✓ 0.2s

parameter의 학습과정에서 변화하는 cost를 graph와 table로 표현한다.

```
# AND, OR, XOP paremeter들의 학습과정에서 얻는 cost들을 graph로 표현한다.
def graph(epoch_num, And, Or, Xor):
    x = range(0,epoch_num)
    plt.figure(figsize=(18,6))
    # AND
    plt.subplot(131)
    plt.title("AND")
    plt.plot(x,And)
    #OR
    plt.subplot(132)
    plt.title("OR")
    plt.plot(x,Or)
    #XOR
    plt.subplot(133)
    plt.title("XOR")
    plt.plot(x,Xor)
    plt.show()

# AND, OR, XOP paremeter들의 학습과정에서 얻는 cost들을 table로 표현한다..
def table(epoch_num, And, Or, Xor):
    source = {'Logical AND':And ,'Logical OR':Or ,'Logical XOR':Xor }
    data = pd.DataFrame(source)
    print("0~10000")
    print(data)

    extrac = []
    for i in range(epoch_num):
        if i % 100 == 0:
            extrac.append([And[i],Or[i],Xor[i]])
    E = np.array(extrac)
    Esource = {'Logical AND':E[:,0] ,'Logical OR':E[:,1] ,'Logical XOR':E[:,2]}
    Edata = pd.DataFrame(Esource)
    print(Edata)
```

[14] ✓ 0.3s

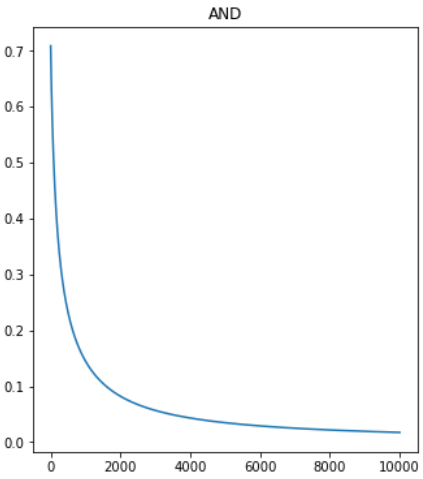
학습을 진행하고 결과를 확인한다.

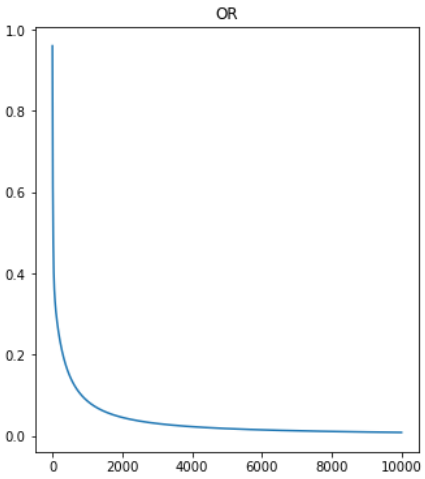
```python
# 학습시킨 model의 결과를 확인한다.
def model_test(And, Or, Xor,X):
    gate = [And, Or, Xor]
    gate_name = ["AND","OR","XOR"]
    for i in range(3):
        print(gate_name[i])
        for x in X:
            print(x, gate[i].predict(x))

# and, or, xor model을 생성하고 학습한다.
#parameter별 model
and_model = logistic_regression_model()
or_model = logistic_regression_model()
xor_model = logistic_regression_model()

And = AND(and_model)
Or = OR(or_model)
Xor = XOR(xor_model)

# cost변화 확인하기
graph(epoch_num, And, Or, Xor)
table(epoch_num, And, Or, Xor)
model_test(and_model, or_model, xor_model,X)
```

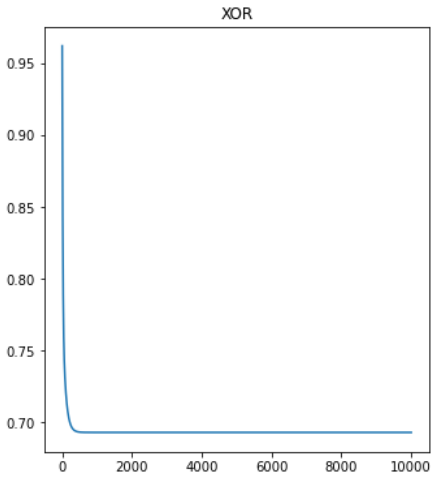
[25] ✓ 1.8s

2. Results

[learning rate: 0.1, epoch: 10000]







0~10000

	Logical AND	Logical OR	Logical XOR
0	0.708244	0.959946	0.961931
1	0.704051	0.918188	0.951293
2	0.699942	0.879251	0.941095
3	0.695917	0.843013	0.931327
4	0.691971	0.809342	0.921976
...	...	...	...
9995	0.017469	0.009266	0.693147
9996	0.017468	0.009265	0.693147
9997	0.017466	0.009264	0.693147
9998	0.017464	0.009263	0.693147
9999	0.017462	0.009262	0.693147

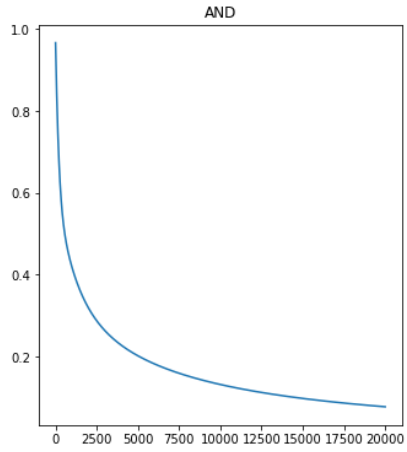
[10000 rows x 3 columns]

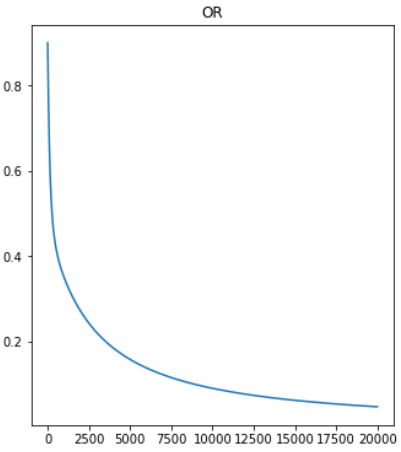
	Logical AND	Logical OR	Logical XOR
0	0.708244	0.959946	0.961931
1	0.479165	0.306303	0.722705
2	0.372061	0.242352	0.701973
3	0.307279	0.200232	0.695736
4	0.263309	0.169944	0.693907
..	...	...	...
95	0.018383	0.009756	0.693147
96	0.018191	0.009653	0.693147
97	0.018002	0.009552	0.693147
...			
98	0.017818	0.009453	0.693147
99	0.017638	0.009356	0.693147

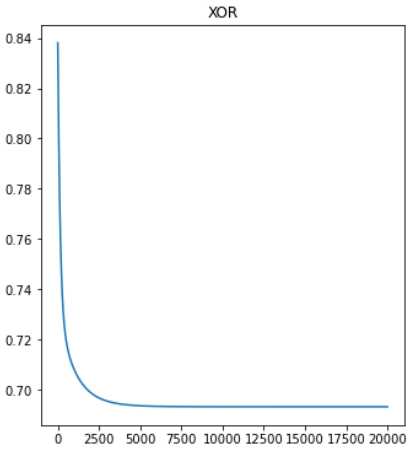
[100 rows x 3 columns]

```
AND
[0 0] 1.2546820381448l2e-05
[1 0] 0.02029077393820609
[0 1] 0.02029078279577317
[1 1] 0.9715805976448177
OR
[0 0] 0.020438795648928344
[1 0] 0.9918351576103295
[0 1] 0.9918385745812653
[1 1] 0.9999985866281421
XOR
[0 0] 0.5
[1 0] 0.5
[0 1] 0.5
[1 1] 0.5
```

[learning rate: 0.01, epoch: 20000]







0~10000

	Logical AND	Logical OR	Logical XOR
0	0.966298	0.900456	0.838163
1	0.964309	0.897094	0.837400
2	0.962327	0.893754	0.836642
3	0.960352	0.890438	0.835888
4	0.958384	0.887146	0.835139
...	...	...	...
19995	0.077675	0.047326	0.693147
19996	0.077672	0.047323	0.693147
19997	0.077669	0.047321	0.693147
19998	0.077665	0.047319	0.693147
19999	0.077662	0.047316	0.693147

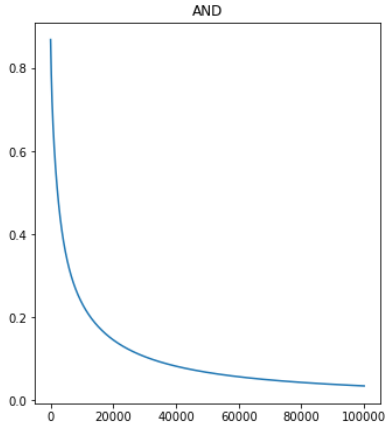
[20000 rows x 3 columns]

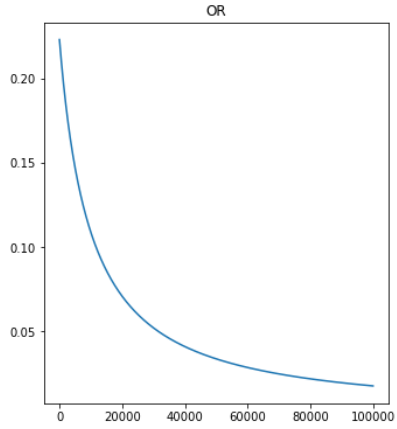
	Logical AND	Logical OR	Logical XOR
0	0.966298	0.900456	0.838163
1	0.798726	0.660697	0.780912
2	0.683806	0.544476	0.750513
3	0.606513	0.482845	0.734333
4	0.554066	0.445657	0.725308
..	...	...	...
195	0.079316	0.048498	0.693147
196	0.078979	0.048257	0.693147
197	0.078645	0.048018	0.693147
...			
198	0.078314	0.047781	0.693147
199	0.077985	0.047546	0.693147

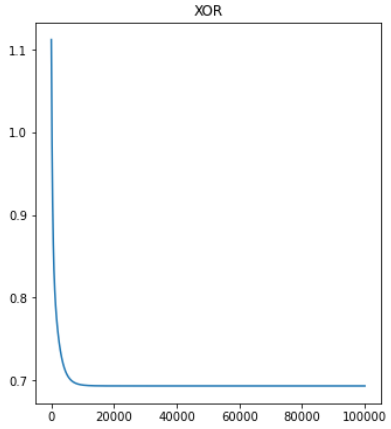
[200 rows x 3 columns]

```
AND
[0 0] 0.0001010855053449396
[1 0] 0.03974740843707284
[0 1] 0.03974740239268657
[1 1] 0.9442829217606771
OR
[0 0] 0.04127623692510122
[1 0] 0.9835934394846635
[0 1] 0.9834978450373619
[1 1] 0.9999879504579583
XOR
[0 0] 0.4999999992008479
[1 0] 0.4999999998745819
[0 1] 0.49999999987459676
[1 1] 0.5000000005483308
```

[learning rate: 0.005, epoch: 100000]







0~10000

	Logical AND	Logical OR	Logical XOR
0	0.868703	0.222975	1.112123
1	0.868244	0.222952	1.111418
2	0.867785	0.222929	1.110713
3	0.867327	0.222906	1.110010
4	0.866870	0.222883	1.109308
...	...	...	...
99995	0.034825	0.017750	0.693147
99996	0.034825	0.017749	0.693147
99997	0.034824	0.017749	0.693147
99998	0.034824	0.017749	0.693147
99999	0.034824	0.017749	0.693147

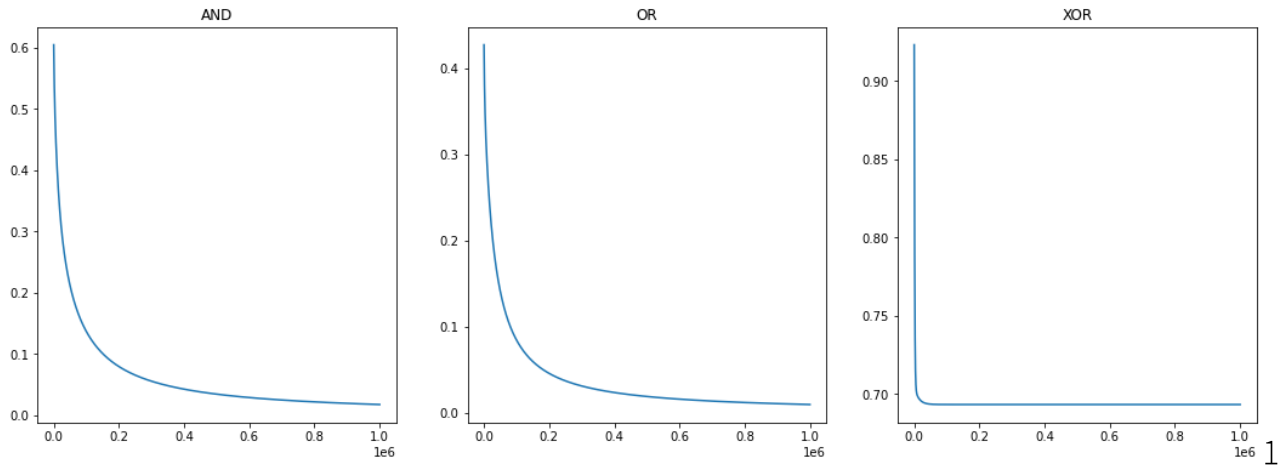
[100000 rows x 3 columns]

	Logical AND	Logical OR	Logical XOR
0	0.868703	0.222975	1.112123
1	0.826921	0.220692	1.047942
2	0.792177	0.218458	0.995671
3	0.762740	0.216271	0.953591
4	0.737323	0.214130	0.919931
..	...	...	...
995	0.034995	0.017834	0.693147
996	0.034960	0.017817	0.693147
997	0.034926	0.017800	0.693147
...			
998	0.034892	0.017783	0.693147
999	0.034857	0.017766	0.693147

[1000 rows x 3 columns]

```
AND
[0 0] 0.000102758387997585
[1 0] 0.03995467589909155
[0 1] 0.03995620970666051
[1 1] 0.9439906553329915
OR
[0 0] 0.038923479579718784
[1 0] 0.9845361075135772
[0 1] 0.9844238423005939
[1 1] 0.9999899349785712
XOR
[0 0] 0.5000000006415464
[1 0] 0.5000000001006655
[0 1] 0.5000000001006898
[1 1] 0.499999999559809
```

[learning rate: 0.001, epoch: 1000000]



위의 결과들을 확인하면 AND, OR operator는 모든 learning rate에서 cost가 0.05 이하의 값으로 거의 0과 가까운 값을 보이며 각 operator의 학습된 model을 이용하여 test를 진행한 결과 또한 알맞게 학습되었음을 보여준다. 그러나 XOR operator의 경우 모든 learning rate와 epoch수에서 cost값이 0.7 이상의 값을 갖으며 model의 predict값또한 0.5로 operator가 제대로 학습되지 못한 것을 확인할 수 있다.

Q: 왜 XOR operator만 학습결과가 안좋은가?

A: XOR의 경우 선형적으로 영역을 분리할 수 있는 AND, OR operator와 다르게 input에 대한 output의 영역을 선형적으로 분리할 수 없기 때문에 학습이 잘 이루어지지 않은 것이다.