Homework #1

2017012251 윤영훈

Data Preparation					
AND		OR		XOR	
0,0	0	0,0	0	0,0	0
0,1	0	0,1	1	0,1	1
1,0	0	1,0	1	1,0	1
1,1	1	1,1	1	1,1	0

1. Boolean operators : AND

Logistic regression (AND)

In [1]:
```
import random
from math import exp, log
import numpy as np
```

Data prepration

In [2]:
```
X = np.array([(0,0),(1,0),(0,1),(1,1)])
Y = np.array([0,0,0,1])
```

◀ Data Set을 numpy array로 나타냄

Model

In [3]:
```
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
```

◀ self.w, self.b를 numpy array로 나타냄

◀ numpy에서 지원하는 inner 함수 사용

In [4]:
```
model = logistic_regression_model()
```

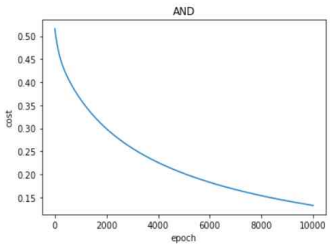
Training

In [5]:
```
def train(X, Y, model, lr = 0.1):
    dw0 = 0.0
    dw1 = 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)

        dw0 += np.multiply(a-y, x[0])
        dw1 += np.multiply(a-y, x[1])
        db += (a-y)

    cost /= m
    model.w[0] -= lr * dw0/m
    model.w[1] -= lr * dw1/m
    model.b -= lr * db/m

    return cost
```

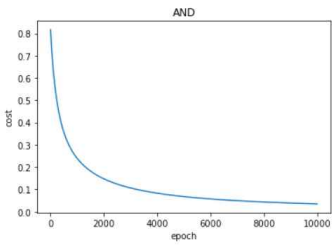
◀ numpy에서 지원하는 multiply 함수 사용

In [6]:
```
for epoch in range(10000):
    cost = train(X, Y, model, 0.1)
    if epoch % 100 == 0:
        print(epoch, cost)
```

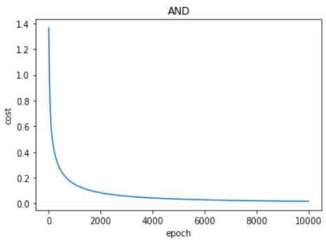
```
0 1.141848974190514
100 0.44571765404746155
200 0.3522160157117974

...

9700 0.017952426502209032
9800 0.017769159010130174
9900 0.017589554356108004
```



learning rate = 0.01



learning rate = 0.05



learning rate = 0.1

※ plot graph의 경우 epoch와 cost 값들을 각각 numpy array로 만들어 plot 함수를 통해 그래프화

▲ cost의 값이 점점 감소 → **학습이 정상적으로 진행중**

Testing

```
In [7]: model.predict((0,0))
Out[7]: 1.2444400780361084e-05

In [8]: model.predict((0,1))
Out[8]: 0.020236703586892318

In [9]: model.predict((1,0))
Out[9]: 0.020236692418202265

In [10]: model.predict((1,1))
Out[10]: 0.9716563914463304
```

∴ (0,0), (0,1), (1,0)에서는 0과 가까운 값이, (1,1)에서는 1과 가까운 값이 출력되는 것을 볼 때 AND 연산자가 잘 학습되었음

2. Boolean operators : OR

Logistic regression (OR)

```
In [1]: import random
        from math import exp, log
        import numpy as np
```

Data preparation

```
In [2]: X = np.array([(0,0),(1,0),(0,1),(1,1)])
        Y = np.array([0,1,1,1])
```

◀ Data Set을 numpy array로 나타냄

Model

```
In [3]: 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
```

◀ self.w, self.b를 numpy array로 나타냄

◀ numpy에서 지원하는 inner 함수 사용

```
In [4]: model = logistic_regression_model()
```

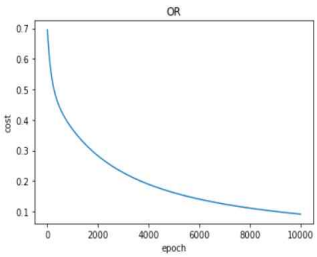
Training

```
In [5]: def train(X, Y, model, lr = 0.1):
            dw0 = 0.0
            dw1 = 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)

                dw0 += np.multiply(a-y, x[0])
                dw1 += np.multiply(a-y, x[1])
                db += (a-y)

            cost /= m
            model.w[0] -= lr * dw0/m
            model.w[1] -= lr * dw1/m
            model.b -= lr * db/m

            return cost
```

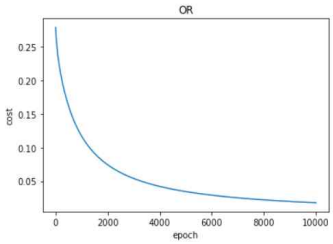
◀ numpy에서 지원하는 multiply 함수 사용

```
In [6]: for epoch in range(10000):
            cost = train(X, Y, model, 0.1)
            if epoch % 100 == 0:
                print(epoch, cost)
```

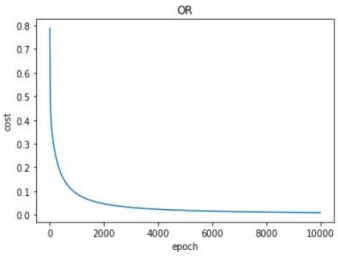
```
0 1.6654057288269368
100 0.3330699164779229
200 0.2541081392257821

...

9700 0.00955360761852166
9800 0.009454766373164133
9900 0.009357934477127045
```



learning rate = 0.01



learning rate = 0.05



learning rate = 0.1

※ plot graph의 경우 epoch와 cost 값들을 각각 numpy array로 만들어 plot 함수를 통해 그래프화

▲ cost의 값이 점점 감소 → **학습이 정상적으로 진행중**

Testing

```
In [7]: model.predict((0,0))
Out[7]: 0.020442614722577

In [8]: model.predict((0,1))
Out[8]: 0.9918433567595195

In [9]: model.predict((1,0))
Out[9]: 0.9918273348811616

In [10]: model.predict((1,1))
Out[10]: 0.9999985858289085
```

∴ (0,0)에서는 0과 가까운 값이, (0,1), (1,0), (1,1)에서는 1과 가까운 값이 출력되는 것을 볼 때 OR 연산자가 잘 학습되었음

3. Boolean operators : XOR

Logistic regression (XOR)

In [1]:
```
import random
from math import exp, log
import numpy as np
```

Data preparation

In [2]:
```
X = np.array([(0,0),(1,0),(0,1),(1,1)])
Y = np.array([0,1,1,0])
```

◀ Data Set을 numpy array로 나타냄

Model

In [3]:
```
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
```

◀ self.w, self.b를 numpy array로 나타냄

◀ numpy에서 지원하는 inner 함수 사용

In [4]:
```
model = logistic_regression_model()
```

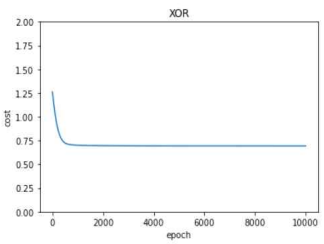
Training

In [5]:
```
def train(X, Y, model, lr = 0.1):
    dw0 = 0.0
    dw1 = 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)

        dw0 += np.multiply(a-y, x[0])
        dw1 += np.multiply(a-y, x[1])
        db += (a-y)

    cost /= m
    model.w[0] -= lr * dw0/m
    model.w[1] -= lr * dw1/m
    model.b -= lr * db/m

    return cost
```

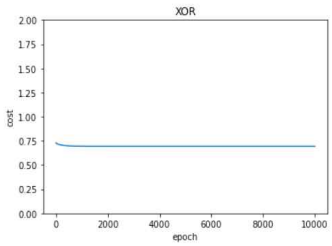
◀ numpy에서 지원하는 multiply 함수 사용

In [6]:
```
for epoch in range(10000):
    cost = train(X, Y, model, 0.1)
    if epoch % 100 == 0:
        print(epoch, cost)
```

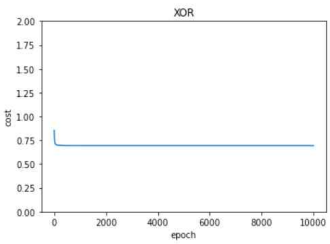
```
0 0.7053910348959286
100 0.6970513620804439
200 0.6945353790751042

...

9700 0.6931471805599453
9800 0.6931471805599453
9900 0.6931471805599453
```



learning rate = 0.01



learning rate = 0.05



learning rate = 0.1

※ plot graph의 경우 epoch와 cost 값들을 각각 numpy array로 만들어 plot 함수를 통해 그래프화

▲ cost의 값이 감소하지 않음 → 학습이 정상적으로 진행되지 않음

Testing

```
In [7]: model.predict((0,0))
Out[7]: 0.5

In [8]: model.predict((0,1))
Out[8]: 0.5

In [9]: model.predict((1,0))
Out[9]: 0.5

In [10]: model.predict((1,1))
Out[10]: 0.5
```

∴ 이론상 (0,0), (1,1)에서는 0과 가까운 값이, (0,1), (1,0)에서는 1과 가까운 값이 출력되어야 하지만 그렇지 못함

→ **XOR 연산자가 잘 학습되지 못함**