

## ▼ Perceptrons - Training

Note for 717005@ Hallym University !

- Make a prediction with weights

```
def predict(X, w):  
    bias = w[0]  
    activation = bias + w[1]* X[0] + w[2]* X[1]  
    if activation >= 0.0:  
        return 1.0  
    else:  
        return 0.0
```

- Estimate Perceptron weights using stochastic gradient descent

```
def train_weights(train, l_rate, n_epoch): # train은 트레이닝 데이터셋, l_rate은 학습률(learning rate)  
    # weights = [0.0 for i in range(len(train[0]))] # weights가 주어지지 않아서 0.0을 len(train[0])만큼 초기화  
    weights = [0, 0, 0]  
    print("-----")  
    print(weights[0])  
    print("-----")  
    vb = []  
    vw0 = []  
    vw1 = []  
    for epoch in range(n_epoch):  
        sum_error = 0.0  
        for row in train: # 데이터 셋을 다 돌려라.  
            prediction = predict(row, weights)  
            error = row[-1] - prediction # 미분 기반  
            sum_error += error**2  
            weights[0] = weights[0] + l_rate * error # bias  
            for i in range(len(row)-1):  
                weights[i + 1] = weights[i + 1] + l_rate * error * row[i] # weights  
                vb.append(weights[0])  
                vw0.append(weights[1])  
                vw1.append(weights[2])  
        print('epoch={}, error={}'.format(epoch, sum_error))  
    return weights, vb, vw0, vw1
```

```
# training set for AND gates  
dataset = [[0,0,0],  
           [1,0,0],  
           [0,1,0],  
           [1,1,1]]
```

- Hyperparameters

```
l_rate = 0.1 # 에러를 수정하는 수치의 비율이라고 "일단은" 생각해두자.  
n_epoch = 5
```

```
weights, vb, vw0, vw1 = train_weights(dataset, l_rate, n_epoch)
```



```
print(weights)
```



```
pred = predict([1,0],weights) # 임의의 테스트 수행  
print(pred)
```



- Why ?

```
# AND Test  
AND_test = [0,0,0,0]  
AND_test[0] = predict([0,0], weights)  
AND_test[1] = predict([0,1], weights)  
AND_test[2] = predict([1,0], weights)  
AND_test[3] = predict([1,1], weights)
```

```
print(AND_test)
```



```
# Another Type Test  
Another = list()
```

```
#Another.append(predict([2,0], weights))  
for i in range(20,30):  
    Another.append(predict([i*0.1,0], weights))
```

```
print(Another)  
# 2.1 부터 1로 바뀐다
```



```
import matplotlib.pyplot as plt
```

```
plt.plot(vb, "r")  
plt.plot(vw0, "b")  
plt.plot(vw1, "g")
```



partial derivative with respect to m

$$\begin{aligned}\frac{\partial J(m, b)}{\partial m} &= \frac{1}{n} \sum_{i=1}^n -2x^{(i)}(y_i - (mx^{(i)} + b)) \\ &= \frac{2}{n} \sum_{i=1}^n x^{(i)}((mx^{(i)} + b) - y^{(i)}) \\ &= \frac{2}{n} \sum_{i=1}^n x^{(i)}(\hat{y}^{(i)} - y^{(i)})\end{aligned}$$

partial derivative with respect to b

$$\begin{aligned}\frac{\partial J(m, b)}{\partial b} &= \frac{1}{n} \sum_{i=1}^n -2(y^{(i)} - (mx^{(i)} + b)) \\ &= \frac{2}{n} \sum_{i=1}^n ((mx^{(i)} + b) - y^{(i)}) \\ &= \frac{2}{n} \sum_{i=1}^n (\hat{y}^{(i)} - y^{(i)})\end{aligned}$$

Partial derivatives : <https://www.mathsisfun.com/calculus/derivatives-partial.html>

- References

<https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/>

