# Deep Learning practice #1-2 report

컴퓨터소프트웨어학부 2018008768 윤정

**[구현 과정]**

```
def generate_data(m):
    X = np.empty((0, 1))
    y = np.empty((0, 1))
    for i in range(m):
        degree_value = random.uniform(0, 360)
        sine_value = math.sin(math.radians(degree_value))
        X = np.append(X, np.array([[degree_value]]), axis=0)
        if sine_value > 0:
            y = np.append(y, np.array([[1]]), axis=0)
        else:
            y = np.append(y, np.array([[0]]), axis=0)
    return X, y
```

Input number m에 대해 x1으로 구성된 degree value X와 0과 1로 구성된 y데이터를 생성하는 함수입니다. 이를 이용해서 X_train, X_test, y_train, y_test를 생성하였습니다.

Logistic regression을 Back propagation을 이용하여 W = [w] 와 b를 구하였습니다.

이 과정에서, 계산상의 에러를 방지하기 위해 degree value인 x값을 radian value값으로 변환한 뒤 계산을 진행하였습니다.

구한 parameter W, b값에 대하여 Cost와 Accuracy를 계산하였습니다.

**[결과]**

m = 10000, n = 1000, K = 5000, alpha = 0.1 일 때 결과값 (매 500회마다 출력, 최종 w와 b)

```
w:  -11.993717404934907
b:  0.9558214825549214
w:  -1.712365379874368
b:  8.77710284795465
w:  -4.770052830646121
b:  10.77341310565361
w:  -5.3639733289835325
b:  12.101498182321073
w:  -5.2027998842346594
b:  12.953467618109299
w:  -5.024666222943029
b:  13.502764008459954
w:  -4.815386339524656
b:  13.857935308285665
w:  -4.553106292176588
b:  14.0942261056433
w:  -4.582314868347667
b:  14.29653554145919
w:  -4.644471615113573
b:  14.493070139047644
Final w:  -4.704886043017936
Final b:  14.68405868665479
```

```
predict(X_train, y_train, m, W, b)
✓ 0.1s

Cost:  0.0557485723384108
Accuracy:  99.71
```

```
predict(X_test, y_test, n, W, b)
✓ 0.5s

Cost:  0.05300868958379096
Accuracy:  99.7
```

Train data m을 변화시키며 예측한 결과값입니다. (alpha=0.01)

| | m=10, n=1000, K=5000 | m=100, n=1000, K=5000 | m=10000, n=1000, K=5000 |
|---|---|---|---|
| Cost (with 'm' train samples) | 0.089492384244199 | 0.169387771885098 | 0.182426788841505 |
| Cost (with 'n' test samples) | 0.253474784781345 | 0.177400746263431 | 0.183106566901287 |

| | m=10, n=1000, K=5000 | m=100, n=1000, K=5000 | m=10000, n=1000, K=5000 |
|---|---|---|---|
| Accuracy (with 'm' train samples) | 100.0 | 97.0 | 96.04 |
| Accuracy (with 'n' test samples) | 86.7 | 94.699999999 | 95.89999999999999 |

K를 변화시키며 예측한 결과값입니다. (alpha = 0.01)

| | m=10000, n=1000, K=10 | m=10000, n=1000, K=100 | m=10000, n=1000, K=5000 |
|---|---|---|---|
| Cost (with 'm' train samples) | 0.428141695419192 | 0.410404007317525 | 0.182084021688916 |
| Cost (with 'n' test samples) | 0.437320107650953 | 0.407479171482288 | 0.176275973356337 |

| | m=10000, n=1000, K=10 | m=10000, n=1000, K=100 | m=10000, n=1000, K=5000 |
|---|---|---|---|
| Accuracy (with 'm' train samples) | 79.67999999999999 | 81.67 | 96.19 |
| Accuracy (with 'n' test samples) | 79.10000000000001 | 82.699999999 | 97.2 |

alpha값을 변화시키며 예측한 결과값입니다. (M = 10000, N = 1000, k = 5000)

| | alpha=0.01 | alpha=0.05 | alpha=0.1 | alpha=0.3 | alpha=0.5 |
|---|---|---|---|---|---|
| Cost (with 'n' test samples) | 0.18639207744429828 | 0.09069277269712503 | 0.05665583551136608 | 0.06122744213148597 | 0.062703821920263 |
| Accuracy (with 'n' test samples) | 96.3 | 98.8 | 99.4 | 96.3 | 97.399999 |

**[분석]**

Practice 1에서 구했던 것보다 전반적으로 cost와 accuracy가 조금 낮게 나왔습니다.

Train data의 크기가 커질수록 train data를 이용하여 predict한 결과값과 test data를 이용하여 predict한 결과값의 차이가 줄어들었습니다. 이로 인해 train data가 커질수록 새로운 데이터에 대한 예측의 정확도가 높아짐을 알 수 있었습니다.

또한, 학습 반복횟수인 K (과제에서는 epoch으로 표현)을 증가시킬수록 cost는 감소하며 accuracy는 증가하는 것을 알 수 있었습니다. 이를 통해 학습 반복횟수가 증가할수록 더 정확한 W와 b값을 추정할 수 있음을 알 수 있었습니다.

마지막으로, 학습 진행속도를 위한 alpha 값을 각각 0.01, 0.05, 0.1, 0.3, 0.5를 넣어 예측해 본 결과, m=10000, n=1000, k=5000의 환경에서는 0.1이 가장 최적의 알파 값이었음을 알 수 있었습니다. 또한, alpha값이 클수록 각 단계(매 500에폭마다)의 변화가 더 크게 일어남을 볼 수 있었습니다.