HW_3

2016707072 신민정

이하 나오는 사진과 코드는 과제 설명과 본인의 코드,그림을 첨부하였습니다.

1. Perceptron Learning Algorithm

Data

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots \\ x_1^{(N)} & x_2^{(N)} \end{bmatrix}$$

# of Column = # of data feature / # of Row = # of data sample

Perceptron에서 bias weight(w0)와 곱해질 x_0 = 1로 설정

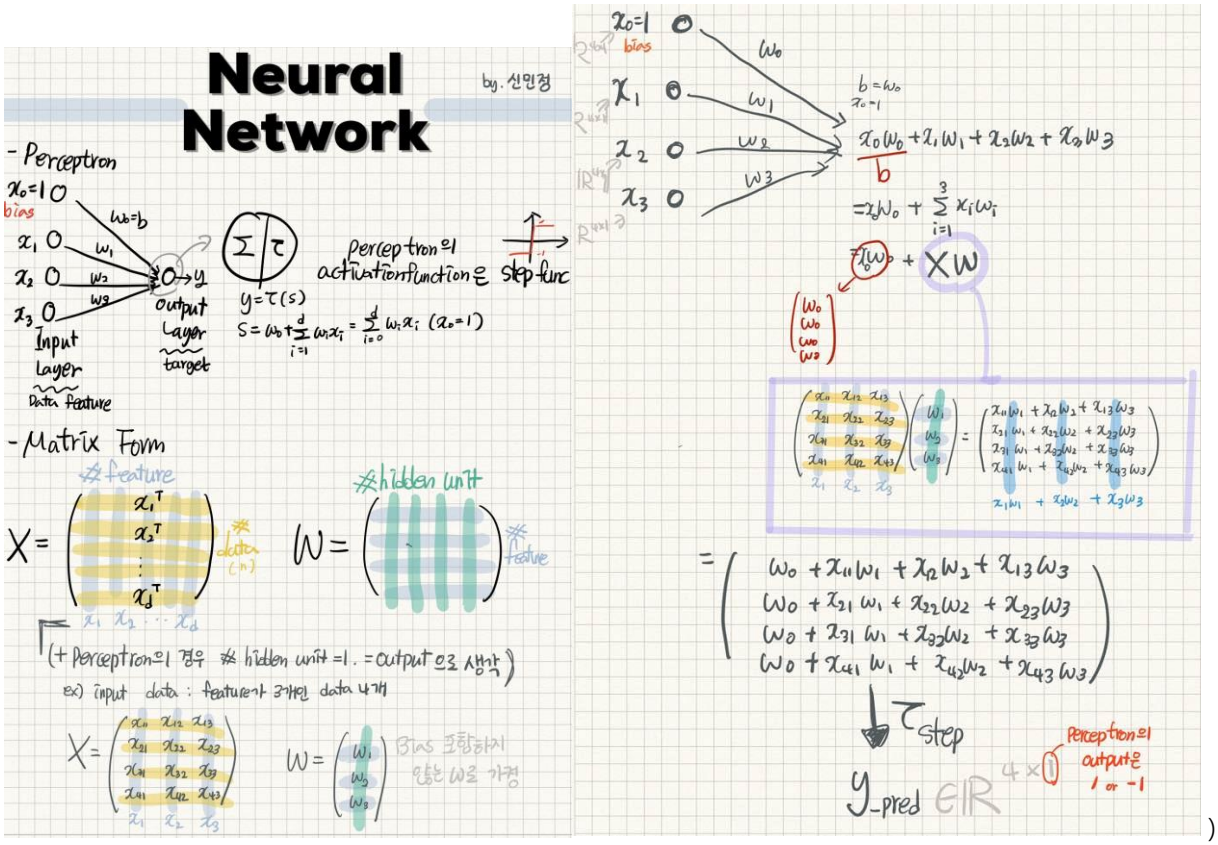
이번 과제에서 **X**의 shape은 `X.shape==(100,2)`. 즉 data sample은 100개, feature는 2개라는 뜻이다.

Bias에 곱해질 x0가 추가된, **X_ones**의 shape은 `shape of X_ones : (100, 3)` 이다.

.($z^{(i)} = \mathbf{w}^T\mathbf{Xones}$ 이 식의 경우) Perceptron에 들어갈 때에 X는 # of Row= # of data feature / # of Column = # of data sample 이어야 하므로 transpose를 취해주어야 한다

(다른 방식 : Perceptron에 들어가는 X를 data from과 일치시키는 경우

Z = XW



)

가중치 matrix인 W의 # of Column = # of data feature / # of Row = # of hidden unit

그러므로 단일 Perceptron을 구현하는 이번 과제에서, **W**의 shape은 `shape of w : (3,)` 이다.

델타학습규칙(퍼셉트론 학습규칙)은 가중치를 갱신하는 학습규칙이다. $w_i = w_i + \rho \sum y_k x_{ki}$ (ρ : learning rate)

본 과제에서는 $\mathbf{w} = \mathbf{w} + \mathbf{X}\mathbf{ones}^{(i)} * y^{(i)}$ 으로 사용하였다.

Epoch는 w를 최적화하도록 학습하는 횟수, iteration은 data sample로 설정하여 1epoch에 전체 data set에 대해 예측값과 비교한다.

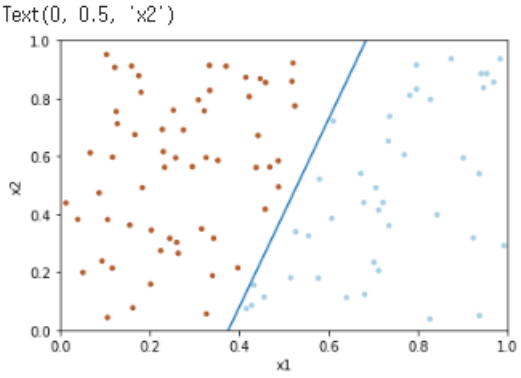
```python
def perceptron(X,Y,epochs):
  X_ones=np.concatenate((np.ones((100,1)),X), axis=1)
  w= np.zeros(X_ones.shape[1])
  X_ones = np.transpose(X_ones)
  print("shape of w :",w.shape)
  print(type(w))
  print("shape of X_ones :", X_ones.shape)
  print("w:",w)
  iteration = 100
  for ep in range(epochs):
   for i in range(iteration): # datasample수 만큼 iteration
    ### START CODE HERE ###
      z = np.dot(w.T,X_ones[:,i])
      """
      shape of w : (3,)
      shape of X_ones : (100, 3)  -> (3,100)

      z(i) = w.T*X_ones[i번째 sample]
      z(i) = scalar
      """
      if z*Y[i] <= 0 :
        w += Y[i]*X_ones[:,i]


    ### END CODE HERE ###

    #print("shape of z : ",z.shape)
  return w
```

주어진 X,Y 뿐만 아니라 epoch을 input으로 지정하여 epoch을 변화시키며 그래프를 비교하였다.

1. epoch =100

```python
w=perceptron(X,Y,100)
assert w.shape[0]==3
print(w)
```

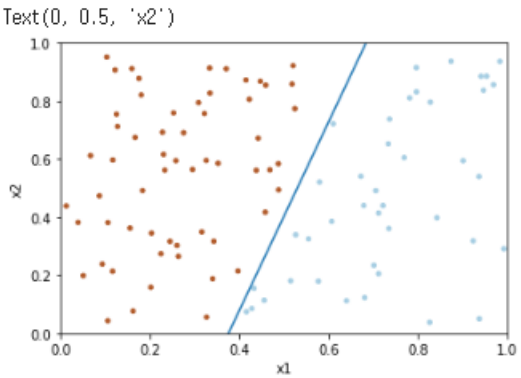
```
shape of w : (3,)
<class 'numpy.ndarray'>
shape of X_ones : (3, 100)
[ 2.         -5.31406015  1.63538617]
```



2. epoch =50

```python
w=perceptron(X,Y,50)
assert w.shape[0]==3
print(w)
```

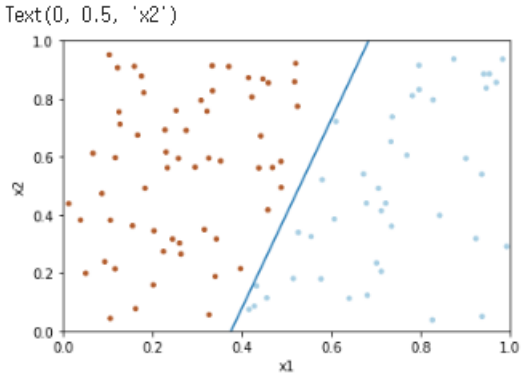
```
shape of w : (3,)
<class 'numpy.ndarray'>
shape of X_ones : (3, 100)
[ 2.         -5.31406015  1.63538617]
```



3. Epoch = 5

```
w=perceptron(X,Y,5)
assert w.shape[0]==3
print(w)
```

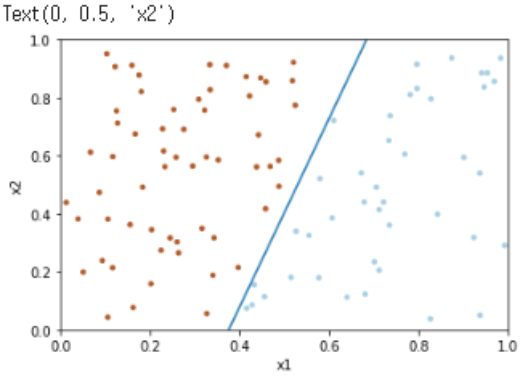
```
shape of w : (3,)
<class 'numpy.ndarray'>
shape of X_ones : (3, 100)
[ 2.         -5.31406015  1.63538617]
```



4. Epoch = 3

```
w=perceptron(X,Y,3)
assert w.shape[0]==3
print(w)
```

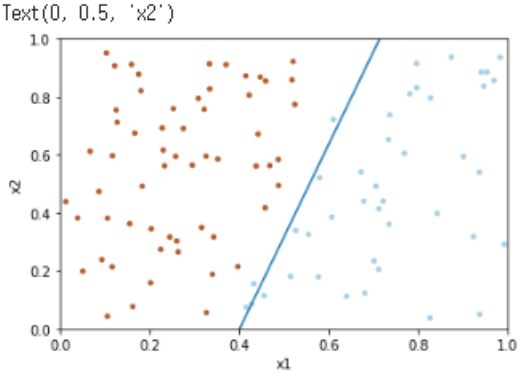
```
shape of w : (3,)
<class 'numpy.ndarray'>
shape of X_ones : (3, 100)
[ 2.         -5.31406015  1.63538617]
```



5. Epoch = 2

```
w=perceptron(X,Y,2)
assert w.shape[0]==3
print(w)
```

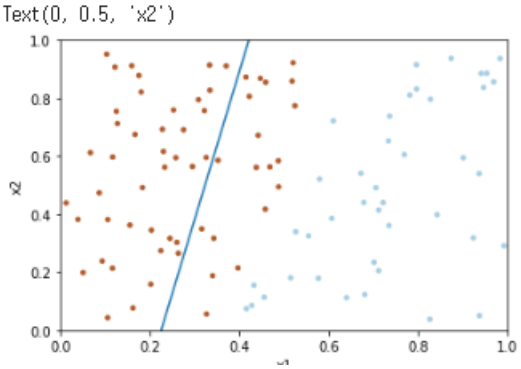
```
shape of w : (3,)
<class 'numpy.ndarray'>
shape of X_ones : (3, 100)
[ 2.         -4.98131465  1.55873623]
```



6. Epoch = 1

```
w=perceptron(X,Y,1)
assert w.shape[0]==3
print(w)
```

```
shape of w : (3,)
<class 'numpy.ndarray'>
shape of X_ones : (3, 100)
[ 1.         -4.43201865  0.86956052]
```

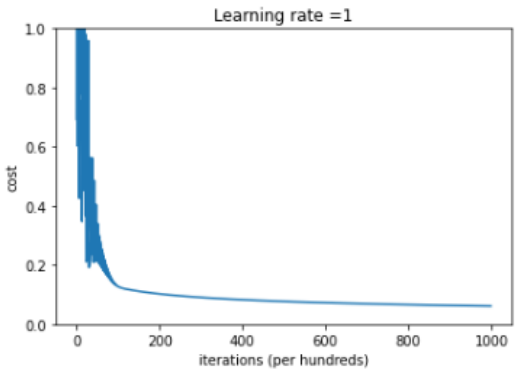


이번 데이터셋에서는 epoch =3일 때가 elbow point이다. Epoch가 3이상으로 올라가면 w값은 수렴하여 거의 변화가 없다. 이번과제에서는 data를 test, train으로 split하지 않았기 때문에 상관이 없지만, train set에 대해 w가 수렴했음에도 epoch을 너무 크게하면 overfitting이 일어날 수 있으므로 elbow point부근에서 학습을 종료하는 것이 바람직하다.

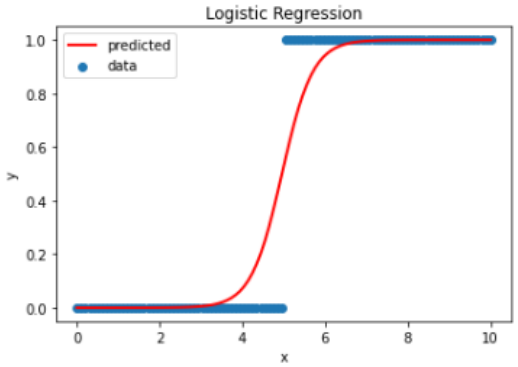
이번과제에서 Perceptron 코드를 구성할 때 data sample하나하나 z값을 구하여 sigma를 코딩으로 구현하였다. Perceptron의 수학식을 완전히 이해할 수 있는 뜻 깊은 과제였다.

2. Logistic Regression

1.
```
d = model(x, y, num_iterations = 1000, learning_rate = 1, print_cost = True)
```




```
w = [[2.67577818]]
b = -13.259904221073695
```



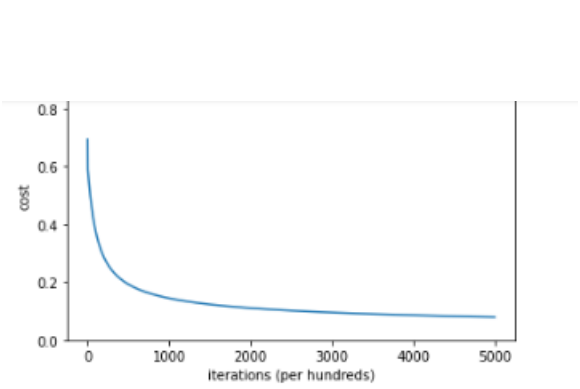
약 200번 iteration하였을 때 cost가 수렴하는 것을 알 수 있다.

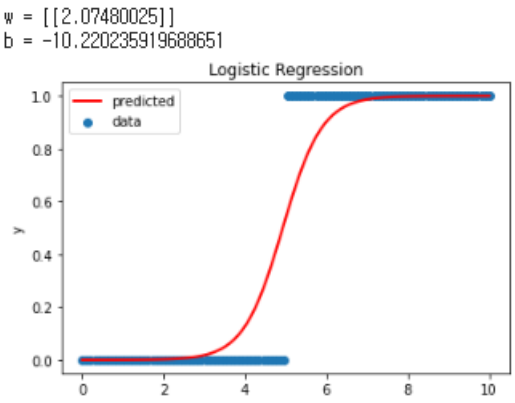
Predicted 함수는 거의 data에 fit하였지만 약간의 오차가 발생하였다.

2.
```
d_1 = model(x, y, num_iterations = 5000, learning_rate = 0.1, print_cost = True)
```

default값에서 iteration 수와 learning_rate를 변경하였다.

Learning rate를 1에서 0.1로 줄였고, learning rate를 줄였으니 num_iteration은 늘려 수렴값을 확인하였다.

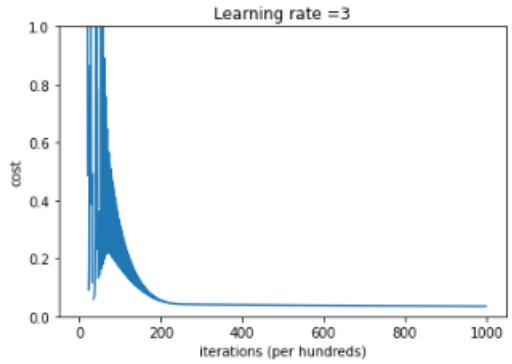


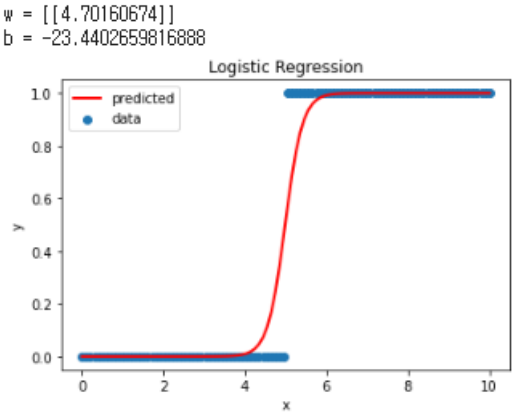


약 1300번의 iteration이상일 때 cost가 수렴하고, default model보다 정확도는 떨어졌다.

3.
```
d_1 = model(x, y, num_iterations = 1000, learning_rate = 3, print_cost = True)
```

Default model에서 learning rate만 으로 늘렸다.





Learning rate가 큰 만큼 빠르게 수렴한다.

이번 과제에서는 정확하게 fit된 model을 만들었지만, learning rate가 너무 크다면 objective function의 minimum을 찾지 못하고 발산할 위험이 있어 주의해야한다.

주관적인 최선의 방법은 0.1 or 0.01정도의 learning rate, epoch =1000으로 크게 설정한 뒤 keras의 callback함수 를 사용하여 local minimum에 빠지지 않으게 global minimum을 찾아으며 overfitting을 방지하는 방법이 좋은 것 같다.

keras의 callback함수:

ModelCheckpoint : 모델이 학습하면서 정의한 조건을 만족했을 때 Model의 weight 값을 중간 저장. 학습시간이 오래걸린다면, 모델이 개선된 validation score를 도출해낼 때마다 weight를 중간 저장함으로써, 혹시 중간에 memory overflow나 crash가 나더라도 다시 weight를 불러와서 학습을 이어나갈 수 있기 때문에, 시간을 save해 줄 수 있다.

EarlyStopping : model의 성능 지표가 설정한 epoch동안 개선되지 않을 때 조기 종료

ReduceLROnPlateau : Local Minima에 빠져 더이상 학습률이 개선되지 않고 정체되었을 때, learning rate를 늘리거나 줄여주는 방법으로 빠져나온다