HW2_Linear Regression

2016707072 신민정

빈칸 Code 채우기

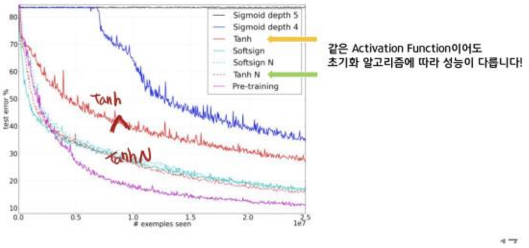
Initialize_with_zeros

Parameter초기화

Zero로 초기화하고 시작

(다양한 방식의 initializer가 있다.

Weight&bias의 initilize를 어떤방식으로 하느냐에 따라 성능에 차이가 있다)




```
# GRADED FUNCTION: initialize_with_zeros

def initialize_with_zeros(dim):
    """
    This function creates a vector of zeros of shape (dim, 1) for w and initializes b to 0.

    Argument:
    dim -- size of the w vector we want (or number of parameters in this case)

    Returns:
    w -- initialized vector of shape (dim, 1)
    b -- initialized scalar (corresponds to the bias)
    """

    ### START CODE HERE ###
    w = np.zeros([dim,1]) #numpy.zeros로 zero vector선언
    b = 0 #scalar
    ### END CODE HERE ###

    assert(w.shape == (dim, 1))
    assert(isinstance(b, float) or isinstance(b, int))

    return w, b
```

```
w = [[0.]
 [0.]]
b = 0
```

Expected Output:

w = [[0.]
[0.]]

b = 0

Propagate

cost값 및 gradient를 계산

```
# GRADED FUNCTION: propagate
from sklearn.metrics import mean_squared_error
def propagate(w, b, X, Y):

    m = X.shape[1]

    # FORWARD PROPAGATION (FROM X TO COST)
    ### START CODE HERE ###
    cost = mean_squared_error((np.dot(w.T,X) + b),Y)/2
    """
    cost = np.sum((np.dot(w.T,X) + b - Y)**2) / (2*m) #MSE사용 (1/2는 상수여서 상관X)
    """
    ### END CODE HERE ###

    # BACKWARD PROPAGATION (TO FIND GRAD)
    ### START CODE HERE ###
    dw = np.dot(X,(np.dot(w.T,X) + b - Y).T)/m
    db = np.sum(np.dot(w.T,X) + b - Y) / m
    ### END CODE HERE ###

    assert(dw.shape == w.shape)
    assert(db.dtype == float)
    cost = np.squeeze(cost)
    assert(cost.shape == ())

    grads = {"dw": dw,
             "db": db}

    return grads, cost
```

Cost fuction으로 MSE를 사용

python으로 구현할 수도 있지만, sklearn에서 import할 수 있다.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

```
dw = [[12.8       ]
 [30.82666667]]
db = 4.533333333333333
cost = 41.49333333333333
```

Expected Output:

dw = [[12.8]

[30.82666667]]

db = 4.533333333333333

cost = 41.49333333333333

Optimization

현재 parameter에 새로운 parameter를 업데이트하는 함수(parmeter : weight, bias)

*학습규칙 : θ=θ−α*dθ (α : learning rate)*

```
# GRADED FUNCTION: optimize

def optimize(w, b, X, Y, num_iterations, learning_rate, print_cost = False):

    costs = []

    for i in range(num_iterations):
        # Cost and gradient calculation
        ### START CODE HERE ###
        grads, cost = propagate(w, b, X, Y)
        ### END CODE HERE ###

        # Retrieve derivatives from grads
        dw = grads["dw"]
        db = grads["db"]

        # update rule
        ### START CODE HERE ###
        w = w - learning_rate*dw
        b = b - learning_rate*db
        ### END CODE HERE ###

        # Record the costs
        costs.append(cost)

        # Print the cost every 100 training iterations
        if i % 100 == 0:
            print ("Cost after iteration %i: %f" %(i, cost))

    params = {"w": w,
              "b": b}

    grads = {"dw": dw,
             "db": db}

    return params, grads, costs
```

```
Cost after iteration 0: 41.493333
w = [[-0.04675219]
 [-0.12676061]]
b = 1.223758731602527
dw = [[ 0.12274692]
 [-0.09406359]]
db = 0.3683397115660049
```

Expected Output:

w = [[-0.04675219]

[-0.12676061]]

b = 1.223758731602527

dw = [[0.12274692]

[-0.09406359]]

db = 0.36833971156600487

Model

```
# GRADED FUNCTION: model

def model(X, Y, num_iterations = 2000, learning_rate = 0.5, print_cost = False):
    """
    Builds the logistic regression model by calling the function you've implemented previously

    Arguments:
    X_train -- training set represented by a numpy array of shape (dim, m_train)
    Y_train -- training labels represented by a numpy array (vector) of shape (1, m_train)
    X_test -- test set represented by a numpy array of shape (dim, m_test)
    Y_test -- test labels represented by a numpy array (vector) of shape (1, m_test)
    num_iterations -- hyperparameter representing the number of iterations to optimize the parameters
    learning_rate -- hyperparameter representing the learning rate used in the update rule of optimize()
    print_cost -- Set to true to print the cost every 100 iterations

    Returns:
    d -- dictionary containing information about the model.
    """

    ### START CODE HERE ###

    # initialize parameters with zeros
    w, b = initialize_with_zeros(X.shape[0])

    # Gradient descent
    parameters, grads, costs = optimize(w, b, X, Y, num_iterations, learning_rate, print_cost )


    # Retrieve parameters w and b from dictionary "parameters"
    w = parameters["w"]
    b = parameters["b"]

    ### END CODE HERE ###

    d = {"costs": costs,
         "w" : w,
         "b" : b,
         "learning_rate" : learning_rate,
         "num_iterations": num_iterations}

    return d
```

```
Cost after iteration 0: 219.679900
Cost after iteration 100: 0.652527
Cost after iteration 200: 0.298894
Cost after iteration 300: 0.168183
Cost after iteration 400: 0.119869
Cost after iteration 500: 0.102011
Cost after iteration 600: 0.095410
Cost after iteration 700: 0.092970
Cost after iteration 800: 0.092068
Cost after iteration 900: 0.091735
```

Expected Output:

Cost after iteration 0: 218.021738

Cost after iteration 100: 1.053341

Cost after iteration 200: 0.692178

Cost after iteration 300: 0.470978

Cost after iteration 400: 0.335501

Cost after iteration 500: 0.252526

Cost after iteration 600: 0.201707

Cost after iteration 700: 0.170582

Cost after iteration 800: 0.151519

Cost after iteration 900: 0.139843

Learning rate & Epoch변경

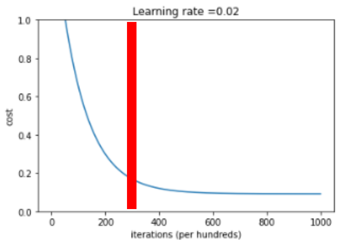
Learning rate : 학습률

Gradient가 줄어드는 감소되는 정도인 learning rate를 변경하며 비교해 보았다.

Learning rate가 작으면 학습시간이 오래걸리게 된다. 시간이 충분하다면 Learning rate를 최대한 작게하여 global minimum을 찾는 것이 최선이지 않느냐는 의문이 생길 수 있지만, learning rate가 너무 작으면 local minimum에 빠질 위험이 있다.

Epoch (number of iterations): 반복횟수

학습을 반복하며 최적의 parameter를 찾는다.



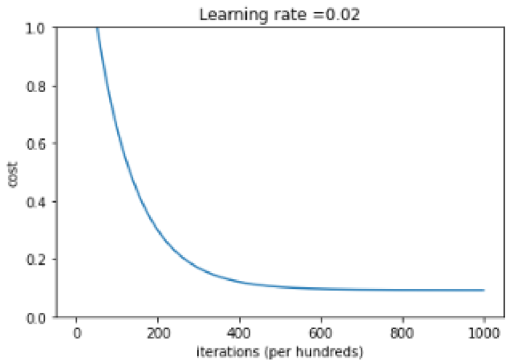
cost function이 수렴하는 구간의 epoch가 최적의 epoch이다.(elbow point)

일정 epoch이상이 되면 cost function은 수렴하게 되는데, 그 이상을 반복하다보면 trainset에 대해 overfitting에 빠질 위험이 있다.

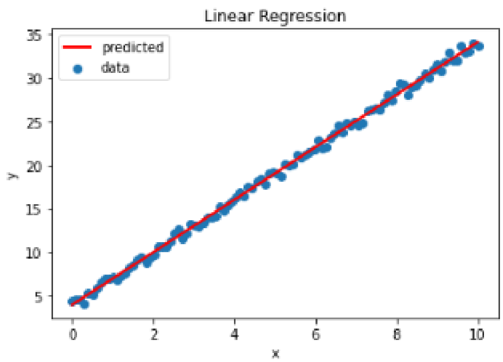
그래프를 보며 결정할 수도 있지만, keras의 callback함수들로 최적의 learning rate와 epoch를 자동으로 결정하여 학습하는 방법도 있다.

Default

```
d = model(x, y, num_iterations = 1000, learning_rate = 0.02, print_cost = True)
```

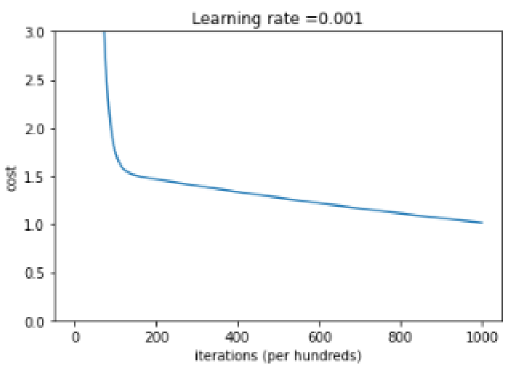



```
w = [[3.01223771]]
b = 3.964959489146403
```



Learning rate 감소 (learning rate = 0.001)

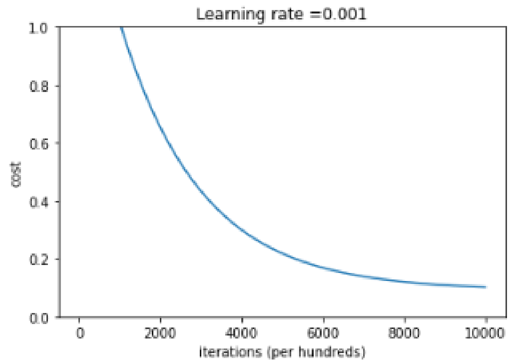
```
d_1 = model(x, y, num_iterations = 1000, learning_rate = 0.001, print_cost = True)
```



learning rate를 줄였더니, 기존의 iteration = 1000 내에서는 끝까지 수렴되지 않았다. iteration을 10000으로 늘려 확인해보자

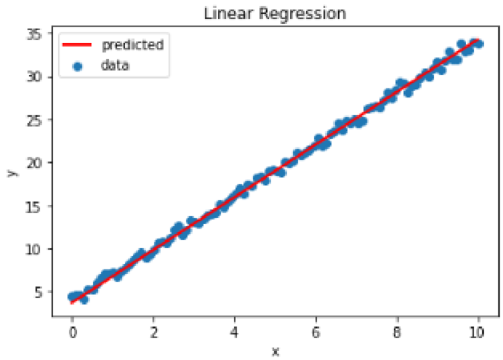
Learning rate감소&Epoch 증가 (learning rate =0.001, epoch = 10000)

```
d_2 = model(x, y, num_iterations = 10000, learning_rate = 0.001, print_cost = True)
```



약 6000정도에서 elbow point가 나타나는 것을 알 수 있다.

```
w = [[3.05209532]]
b = 3.6998785938721888
```




```
x = np.array([np.linspace(0, 10, 100)]) # 0부터 10까지 100개의 데이터를 생성합니다.
y = 3 * x + 4 + np.random.randn(*x.shape) * 0.5  # noise값을 변화하면서 다양한 데이터를 생성해 보세요.
```

Data를 설정할 때부터, linear한 data였기 때문에 최적화가 잘되었다. Learning rate와 epoch에 따라 크게 차이가 없다.

New Data Set with sklearn

Code 주석 참고

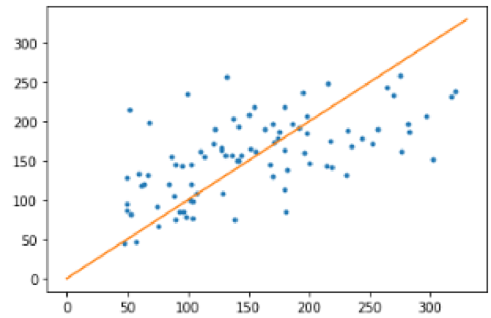
```
# 데이터 로드
from sklearn import datasets
diabetes = datasets.load_diabetes()
X = pd.DataFrame(diabetes.data)
Y = pd.Series(diabetes.target)
data = pd.concat([X,Y], axis = 1)
data.columns = ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6', 'target']
```

```
data.head()
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0

```
# 예측 vs. 실제데이터 plot
Y_pred = model_LinearRegression.predict(X_test)
plt.plot(Y_test, Y_pred, '.')

# 예측과 실제가 비슷하면, 라인상에 분포함
x = np.linspace(0, 330, 100)
y = x
plt.plot(x, y)
plt.show()
```



line상에 fit하지 않은 것으로 보아 linear regression이 적합하지 않아보인다.