3. 파이썬과 벡터화

<벡터화>

벡터화란?

- 벡터화 아닐때에는 for 문 반복해야함
- 벡터화된 구현은 w^Tx를 직접 계산한다 -> 훨씬 빠르다!

np.dot(w,x)+b

★ built in 함수를 쓰면 numpy가 병렬화의 장점을 통해 계산을 더 빠르게 할 수 있음

(CPU와 GPU 둘다 적용)

<벡터화 예제>

신경망이나 로지스틱 회귀 프로그래밍에서는 가능한 for문을 쓰지 않는 것이 좋다

⇨ 내장함수나 다른 방법 써라

U=A v ; u_i ; u=np.zeros(n,1)

<벡터화할 때>

U= np.dot(A,v)

import numpy as np

u=np.exp(v)

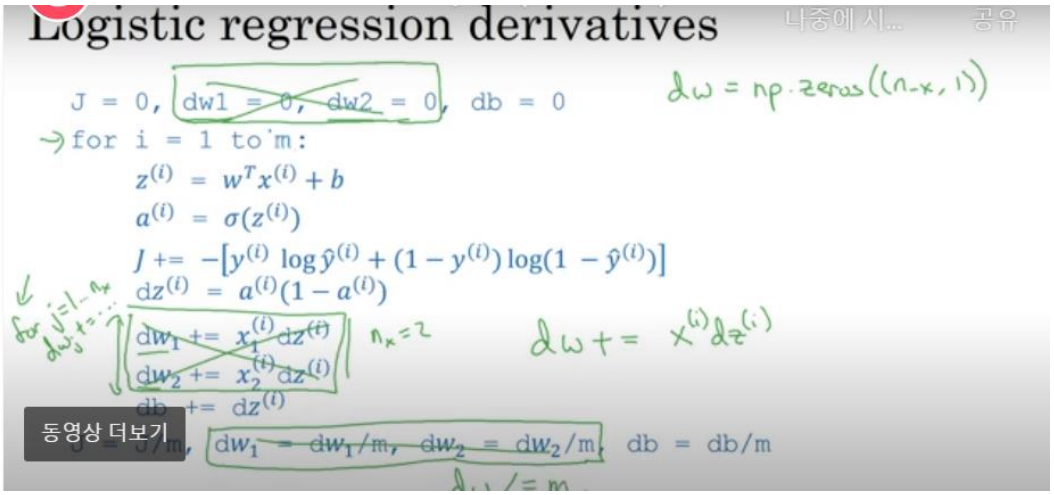
⇨ for 문을 안써도 구할 수 있다!

다양한 파이썬 내장함수 : np.log ; np.abs ; np.max(v,0) : v의 원소와 0 중 더 큰 값 반환

같은 원리로 로지스틱 회귀에서 두번째 for 문 제거할 수 있다

- dw를 벡터로 만든다 np.zeros((n_x,1))

- dw+=x^(i)*dz(i)
- dw/=m



<로지스틱 회귀의 벡터화>

■ 정방향 전파:

z=wT x + b , a=시그모이드함수(z) 를 반복 계산해야한다.

for 문 없이 계산하기 위해서는 Z는 훈련 입력을 열로 쌓은 벡터 (n_x 행 m열)

z(1), z(2), z(3) 를 계산하기 위해서는 (1,m) 벡터 먼저 만들면 b자리에 row 벡터 들어간다.

결과적으로 1* m 벡터가 나온다

Z 벡터에서 첫번째 값이 z1, 두번째값이 z2, 이렇게 한번에 구할 수 있다.

- 파이썬 코드

Z=np.dot(x.T,X)+b,

★ 파이썬에서 b는 하나의 숫자임. 자동으로 (1,m) row 벡터로 바꾸어준다

=> **브로드캐스팅** !!

a를 계산하기 위해서는 : a1,a2,a3 .. 해서 A라고 하기

시그모이드 함수에 Z를 인자로 받아서 A로 반환

m개의 훈련 샘플을 순환하는 대신 한 코드로 z를 계산하고 시그마 구현으로 a를 한줄로 구할 수 있다.

<로지스틱 회귀의 경사 계산을 벡터화>

dz1= a1-y1, dz2=a2-y2 , ... => dZ 를 (1,m) 벡터로 정의

★ dZ=A-Y

dw 는 0벡터

dw+=x1dz1, dw+=x2dz2, .. dw/=m

db=0, db+=dz1, .. db/=m (sum 하고 m으로 나누기)

db=1/m * (np.sum(dz))

dw=1/m * XdzT =

w=w-learningrate dw

b=b- learningrate db

⇨ 경사하강법의 한 반복

⇨ 이렇게 벡터화를 통해 for 문을 없앨 수 있다.

but 경사하강법을 여러 번 반복하려면 반복 횟수 만큼 for 문 필요하긴하다.

<파이썬의 브로드캐스팅>

EX) 식품별로 carbs, protein fats 나온 데이터프레임 보고 칼럼별로 데이터를 더해서 백분율을 계산하여 각 식품의 칼로리의 퍼센트 구하기

- 실제 코드

A를 3by4 데이터 프레임이라고 하자

cal= A.sum(axis=0) => 칼럼별로 합 나옴 (axis=0은 세로로 더하라는 뜻)

percentage=A/cal.reshape(1,4) <= (여기서는 이미 (1,4)이긴 함)

★ reshape 함수는 형태 확실하지 않을 때 쓰면 좋다

- 브로드캐스팅

(m,n) +-/* (1,n) 할 때는 (1,n)을 (m,n)으로 만든 후 연산

(m,n) +-/* (m,1) 하면 n번 가로로 복사해서 (m,n) 으로 만든 후 연산

<파이썬과 넘파이 벡터>

a= np.random.randn(5,1) 하면 칼럼벡터

(n,) 인 배열(랭크 1 배열) 을 사용하지 않는 것이 좋음(a.shape 하면 (5,) 나온는

대신 칼럼벡터나 로우벡터 만들기!

차원 잘 모르면 assert 함수 쓰기 TJtj 행렬과 배열 차원 확인하기 . assert(a.reshape == (5,1))

<로지스틱 회귀의 비용함수>

yhat=sigmoid(wTx+b)

y가 1일 확률= yhat , y가 0일 확률=1-yhat

★ p(y|x)=yhat^y * (1-yhat)^(1-y)

y=1 일 때 / y가 0 일 때 나눠서 생각하면 쉽다

로그함수는 단순 증가함수라서 로그 값을 최대화 하는 것은 p(y|x)를 최대화하는 것 과 같다.

log(p(y|x)) = y*log(yhat) + (1-y)log(1-yhat)

⇨ 손실함수의 음수가 된다 (확률을 높이려면 손실함수를 최소화하고 싶기 때문)

⇨ 손실함수 최소화 = 확률의 로그값 최대화

<m개 샘플에 대해서는?>

전체 확률 = 각 확률의 곱

곱에 로그 씌우면 합

로지스틱 회귀의 비용 J(w,b) = sum(loss) /m