1.

# What is vectorization?

for문 없애는 예술! 큰 데이터 셋에서 벡터화 중요!

$z = w^T x + b$

column 벡터.

$w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \qquad w \in \mathbb{R}^{n_x}, \; x \in \mathbb{R}^{n_x}$

① Non-vectorized:

```
z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b
```

② Vectorized

직접 계산!

$z = \underbrace{np.dot(w, x)}_{w^T x} + b$ 훨씬 빠르다!

파이썬 numpy

→ GPU
→ CPU

SIMD — single instruction multiple data

병렬 명령어

for 필요없는 함수 or np.dot 활용.

→ 파이썬 NumPy가 병렬화 장점 통해 계산 훨씬 빠르게 해줌.

```
import numpy as np
a = np.array([1, 2, 3, 4])
print a
```

코드 벡터화시 훨씬 빠른 결과

```
In [*]: import time

a = np.random.rand(1000000)
b = np.random.rand(1000000)

tic = time.time()   시작
c = np.dot(a,b)
toc = time.time()   끝

print(c)
print("Vectorized version:" + str(1000*(toc-tic))

c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]
toc = time.time()

print(c)
print("For loop:" + str(1000*(toc-tic)) + "ms")
```

```
250286.989866
Vectorized version:1.5027523040771484ms
```

300배 단축!

2.

# Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$u = Av$

$u_i = \sum_j A_{ij} v_j$

```
u = np.zeros((n,1))
for i ...
    for j ...
        u[i] += A[i][j] * v[j]
```

```
u = np.dot(A,v)
```

# Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector.

모든 원소에

지수계산

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

```
u = np.zeros((n,1))
for i in range(n):
    u[i]=math.exp(v[i])
```

```
import numpy as np
u = np.exp(v)
```

np.log(v) → 로그값

np.abs(v) → 절댓값

np.maximum(v,0) ← 0과 v중 큰 값

v**2 제곱   1/v 역수

내장함수 변경 확인!

# Logistic regression derivatives

로지스틱 회귀의 도함수를 구하는 코드

dw = np.zeros((n_x,1))

J = 0, $\boxed{\text{dw1} = \cancel{0,\ \text{dw2} = 0}}$, db = 0

for i = 1 to m: ← 훈련세트 순환 for.

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$

$dz^{(i)} = a^{(i)}(1 - a^{(i)})$

for j=1…$n_x$

$dw_j$ += …

개선? $dw_j$??

$\boxed{\cancel{dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}}}$

$\boxed{\cancel{dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}}}$

$n_x = 2$

$dw \mathrel{+}= \underline{x^{(i)} dz^{(i)}}$

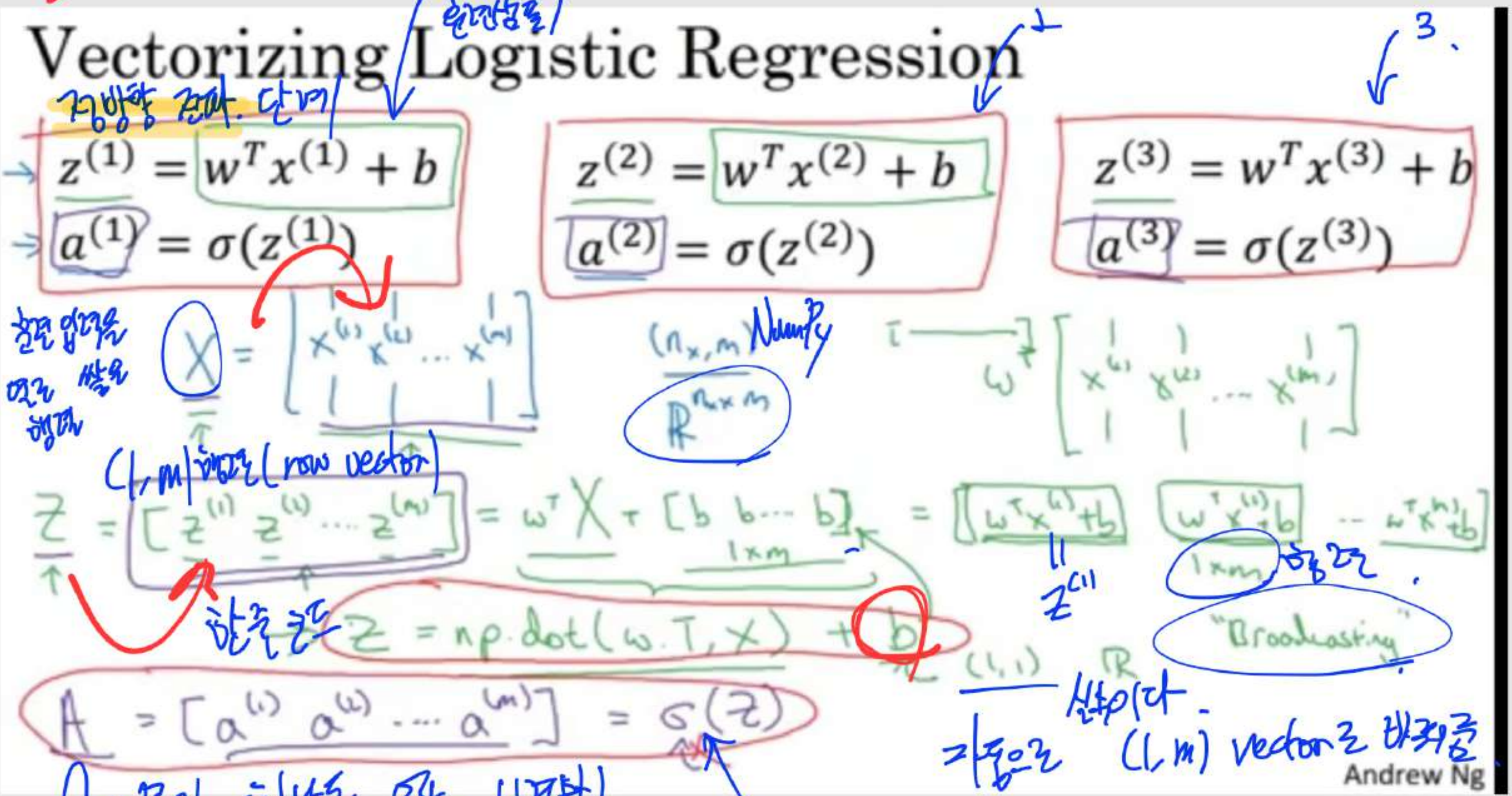
$db \mathrel{+}= dz^{(i)}$

J = J/m, $\boxed{dw_1 \cancel{= dw_1/m,\ dw_2 = dw_2/m}}$, db = db/m

dw /= m

!

3.




for 문이 하나도 없는 방정식!

적절한 σ 구현으로 모든 a 동시 계산

소문자 x, z 쌓아서 대문자 X, Z 얻음!

벡터 값을 가지는 시그모이드 함수 구현 Z(대문자) input → A 반환.

- 아래의 식은 for문을 이용해 i의 값을 변화시키며 계산해야 합니다.
- $z^{(i)}=W^Tx^{(i)}+b$
- $a^{(i)}=\sigma(z^{(i)})$

- 하지만 계산의 효율성을 증가시키기 위해 벡터를 이용하면 다음과 같이 계산할 수 있습니다.
- Z = np.dot(np.transpose(W), X) + b
- 위의 코드에서 (1,m) 크기의 행렬과 상수 b를 더하기에 오류가 날 것 같지만, 파이썬이 자동적으로 상수를 (1,m) 크기의 행렬로 브로드캐스팅 해주기에 오류가 발생하지 않습니다.

→ 역방향 전파의 도함수도 벡터화를 통해 구해보자!

4.

m : 경사하강 동시에 하는 법

# Vectorizing Logistic Regression

$dz^{(1)} = a^{(1)} - y^{(1)}$ $\quad dz^{(2)} = a^{(2)} - y^{(2)}$ ....

$\boxed{dZ = [dz^{(1)} \; dz^{(2)} \cdots dz^{(m)}]}$ ← $\quad 1 \times m$

$A = [a^{(1)} \cdots a^{(m)}] \quad Y = [y^{(1)} \cdots y^{(m)}]$

→ $dZ = A - Y = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \cdots]$

$dw = 0$

이거 넣음

$dw \mathrel{+}= x^{(1)} dz^{(1)}$

$dw \mathrel{+}= x^{(2)} dz^{(2)}$

$\vdots$

$dw \mathrel{/}= m$

$\cancel{dw_1}$, $\cancel{dw_2}$

for문 제거함!

$db = 0$ 초기화

$db \mathrel{+}= dz^{(1)}$

$db \mathrel{+}= dz^{(2)}$

$\vdots$

$db \mathrel{+}= dz^{(m)}$

$db \mathrel{/}= m$

db 벡터화

$$\boxed{db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} = \frac{1}{m} np.sum(dZ)}$$

$$\boxed{dw = \frac{1}{m} X dZ^T}$$

$$= \frac{1}{m} \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \left[ x^{(1)} dz^{(1)} + \cdots + x^{(m)} dz^{(m)} \right]$$

$n \times 1$

for문 없이 변수의 갱신값 계산가능

# Implementing Logistic Regression

벡터화하지 않은 구현

J = 0, $dw_1$ = 0, $dw_2$ = 0, db = 0

for i = 1 to m:

$\quad z^{(i)} = w^T x^{(i)} + b$

$\quad a^{(i)} = \sigma(z^{(i)})$

$\quad J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$\quad dz^{(i)} = a^{(i)} - y^{(i)}$

$\quad dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$\quad dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$ $\quad$ $dw \mathrel{+}= x^{(i)} * dz^{(i)}$

$\quad db \mathrel{+}= dz^{(i)}$

J = J/m, $dw_1 = dw_1/m$, $dw_2 = dw_2/m$

db = db/m

for iter in range(1000): ←

$Z = w^T X + b$

$= np.dot(w.T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X dZ^T$

$db = \frac{1}{m} np.sum(dZ)$

순방향/역방향 모두 했다!

$w := w - \alpha dw$

$b := b - \alpha db$

m개의 모든 훈련 샘플에 대해 예측값 & 도함수 계산

경사하강법을 여러 번 반복하고 싶으면 for문 필요!

5.

# Broadcasting example

Calories from Carbs, Proteins, Fats in 100g of different foods:

$$
\begin{array}{r|cccc}
 & \text{Apples} & \text{Beef} & \text{Eggs} & \text{Potatoes} \\
\hline
\text{Carb} & 56.0 & 0.0 & 4.4 & 68.0 \\
\text{Protein} & 1.2 & 104.0 & 52.0 & 8.0 \\
\text{Fat} & 1.8 & 135.0 & 99.0 & 0.9
\end{array} = A
$$

$\underbrace{(3,4)}$

axis

0 열

1 행

59cal $\quad \frac{56}{59} = 94.9\%$

Calculate % of calories from Carb, Protein, Fat. Can you do this without explicit for-loop?

```
cal = A.sum(axis = 0)
percentage = 100*A/(cal.reshape(1,4))
```

$(3,4) \;/\; (1,4)$

→ 차원이 확실하지 않다면 적절하게 호출해서 쓰자!

```
import numpy as np

A = np.array([[56.0, 0.0, 4.4, 68.0],
              [1.2,104.0,52.0,8.0],
              [1.8,135.0,99.0,0.9]])

print(A)
```

```
[[  56.     0.     4.4   68. ]
 [   1.2  104.    52.     8. ]
 [   1.8  135.    99.     0.9]]
```

```
cal = A.sum(axis=0)
print(cal)
```

```
[  59.   239.   155.4   76.9]
```

```
percentage = 100*A/cal.reshape(1,4)
print(percentage)
```

reshape!

# Broadcasting example

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \cancel{100} \begin{bmatrix}100\\100\\100\\100\end{bmatrix} = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$$

$$\underset{(m,n)\ (2,3)}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{(1,n) \rightsquigarrow (m,n)\quad (2,3)}{\begin{bmatrix}100 & 200 & 300\\ 100 & 200 & 300\end{bmatrix}} = \begin{bmatrix}101 & 202 & 303\\104 & 205 & 306\end{bmatrix}$$

$$\underset{(m,n)}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{(m,1) \to (m,n)}{\begin{bmatrix}100 & 100 & 100\\200 & 200 & 200\end{bmatrix}} = \begin{bmatrix}101 & 102 & 103\\204 & 205 & 206\end{bmatrix}$$

# General Principle

$(m,n)$ matrix $\quad \left( \begin{array}{c} + \\ - \\ * \\ / \end{array} \right) \quad$ copy (element-wise 연산)

$(1,n) \rightsquigarrow (m,n)$

$(m,1) \rightsquigarrow (m,n)$

$(m,1) + \mathbb{R}$ 실수 m 복사

$\begin{bmatrix}1\\2\\3\end{bmatrix} + 100 = \begin{bmatrix}101\\102\\103\end{bmatrix}$

$[1\ 2\ 3] + 100 = [101\ \ 102\ \ 103]$

NumPy 브로드캐스팅 문서 읽어보기

Matlab/Octave: bsxfun

NumPy

6. 브로드캐스팅 내용과 작동방법 모르면 알아차리기 힘든 오류 발생

## Python/numpy vectors

```
a = np.random.randn(5)
```
a.shape = (5, )
"rank 1 array"

행·열 벡터가 아니라서 직관적이지 못함
Don't use

배열 항상 벡터로! → 동작 쉽게 이해 가능

```
a = np.random.randn(5,1)
```
→ a.shape = (5,1) column vector ✓

```
a = np.random.randn(1,5)
```
→ a.shape = (1,5) row vector ✓

```
assert(a.shape == (5,1))
```

a = a.reshape((5,1)) rank 1을 바꾸어주기

문서의 역할을 한다 자주 사용을 추천!

행렬과 배열의 차원을 확인!

7. 가이드

shift+enter / to run cell

코드 블럭은 커널에서 실행. 서버에서 실행되는 코드
커널이 죽으면 restart!!

NumPy 불러오고 변수 정의 코드들.. 실행해주자!
submit assignment 확인

# Logistic regression cost function

8.

$\hat{y}$ y 예측값

$\hat{y} = \sigma(w^T x + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$

Interpret $\hat{y} = p(y=1|x)$ given $x$

특징 x 주어지면 y 1 확률($\hat{y}$) 반환

If $y=1$ : $P(y|x) = \hat{y}$

If $y=0$ : $P(y|x) = 1 - \hat{y}$

# Logistic regression cost function

이진 분류

$$\boxed{\text{If } y = 1: \quad p(y|x) = \hat{y}}$$
$$\boxed{\text{If } y = 0: \quad p(y|x) = 1 - \hat{y}}$$

$p(y|x)$

$$\boxed{p(y|x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}}$$

If $y=1$: $p(y|x) = \hat{y} \underbrace{(1-\hat{y})^0}_{=1}$

If $y=0$: $p(y|x) = \underbrace{\hat{y}^0}_{=1} (1-\hat{y})^{(1-y)} = 1 \times (1-\hat{y}) = 1-\hat{y}$

$\uparrow \log p(y|x) = \log \hat{y}^{y} (1-\hat{y})^{(1-y)} = y \log \hat{y} + (1-y)\log(1-\hat{y})$

$= -\mathcal{L}(\hat{y}, y) \downarrow$

단조증가함수.

전에 정의한 손실함수 -

최소화하려고 남음!

4'49"

Andrew N

# Cost on $m$ examples

$$\log p(\text{labels in training set}) = \log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)})$$

$$\log p(\ldots\ldots) = \sum_{i=1}^{m} \underbrace{\log p(y^{(i)}|x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)}, y^{(i)})}$$

$$= -\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Maximum likelihood estimation

Cost: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

(minimize)

### 학습내용

- 앞서 로지스틱 회귀에서 배운 손실함수를 상기해봅시다.
  - y값이 1 이 될 확률 : $P(y=1|x) = \hat{y}$
  - y값이 0 이 된 확률 : $P(y=0|x) = 1 - \hat{y}$
- 위 두가지 경우를 하나의 수식으로 나타내면 아래와 같습니다.
  - $P(y|x) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$
  - 만약에 y = 1 일 경우, $P(y|x) = \hat{y}^{1}(1-\hat{y})^{0} = \hat{y}$
  - 만약에 y = 0 일 경우, $P(y|x) = \hat{y}^{0}(1-\hat{y})^{1} = 1 - \hat{y}$
- 또한, 로그함수의 단조적인 성격 때문에 위 식은 아래와 동일 합니다.
  - $\log P(y|x) = \log(\hat{y}^{y}(1-\hat{y})^{(1-y)}) = y\log\hat{y} + (1-y)\log(1-\hat{y})$
- 우리의 목적은 확률( $\log P(y|x)$ )을 최대화 시키는 것이기 때문에 이와 등치인 -1 을 곱한 확률 ( $-\log P(y|x)$ )를 최소화를 목적으로 손실함수를 정의 합니다.
- 따라서, 훈련 샘플 하나의 손실함수는 아래와 같이 정의 됩니다.
  - $L(\hat{y}, y) = -\log P(y|x) = -(\hat{y}^{y}(1-\hat{y})^{(1-y)})$
- 비용함수는 m개 훈련 세트 중, 각 샘플 ( $x^{(i)}$ )이 주어졌을 때, 샘플에 해당하는 라벨( $y^{(i)}$ )값이 1 혹은 0 이 될 확률의 곱으로 구할 수 있습니다.
  - $P(labels\ in\ training\ set) = \prod_{i=1}^{m} P(y^{(i)}|x^{(i)})$
- 양변에 로그를 취하고, 손실함수 부분을 치환 해주면, 위의 식은 아래와 같습니다.
  - $\log P(labels\ in\ training\ set) = \log\prod_{i=1}^{m} P(y^{(i)}|x^{(i)}) = -\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$
- 따라서 비용함수는 손실함수들의 평균을 최소화 하는 것으로 정의하고 아래와 같습니다.
  - $J(w,b) = -\log P(labels\ in\ training\ set) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$