[2-1. Logistic Regression as a Neural Network]

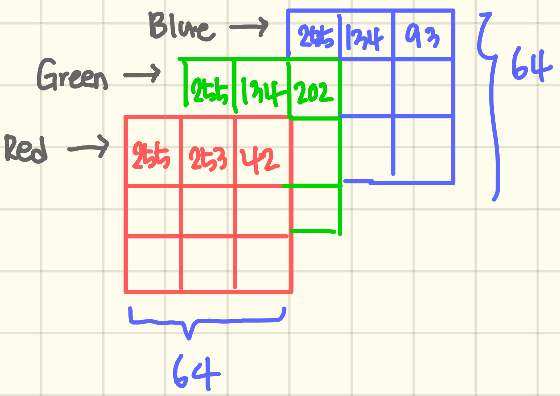
<Binary Classification>

*Binary Classification : 그렇다 / 아니다 2개로 분류하는 것으로, 결과가 '그렇다' 이면 1으로, '아니다' 이면 0으로 표현

ex)

고양이 사진 → $\underbrace{1 \text{ (cat) vs } 0 \text{ (non cat)}}_{y}$

x



$\Rightarrow X = \begin{bmatrix} 255 \\ 253 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \end{bmatrix}$ 이므로, 전체차원 : 64×64×3 = 12,288

$\therefore n = n_x = 12288$

*notation

- (x, y) : $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$
- m training example : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots (x^{(m)}, y^{(m)})\}$

  $M_{train} = m$ / M_{test} = #test example
- $X = \underbrace{\begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}}_{m} \Big\} n_x \Rightarrow X \in \mathbb{R}^{n_x \times m}$

  X.shape = (n_x, m)
- $Y = [y^{(1)} \ y^{(2)} \ \cdots \ y^{(m)}] \Rightarrow Y \in \mathbb{R}^{1 \times m}$, Y.shape = $(1, m)$

<Logistic Regression>

* Logistic Regression : y가 0 또는 1로 정해져 있는 이진 분류 문제에 사용되는 알고리즘

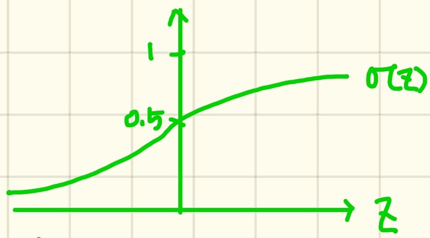
X: 입력특성 , y: 주어진 x에 해당하는 실제 값 , $\hat{y}$: y의 예측값 = $P(y=1|x)$

이때, $x \in \mathbb{R}^{n_x}$ 이고 parameter : $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$ 이므로

⇒ Output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z}) = \sigma(z)$

└ $\hat{y} = w^T x + b$로 계산시, $0 \le \hat{y} \le 1$의 조건을 만족하지 않을 수 있으므로 시그모이드 함수를 통하여 0과 1의 범위로 맞춰준다.

※ 시그모이드 함수 : $\sigma(z) = \frac{1}{1+e^{-z}}$



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

given $\{(x^{(1)}, y^{(1)}), \dots (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$

<Logistic Regression Cost Function>

* 손실함수(Loss Function): 하나의 입력특성(x)에 대한 실제 값(y)과 예측값($\hat{y}$)의 오차를 계산하는 함수

- 일반적으로 손실함수는 보통 $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ 으로 사용하나, 로지스틱 회귀는 지역 최소값에 빠질 수 있으므로 다른 손실함수를 사용한다.
- 로지스틱 회귀의 손실함수 : $L(\hat{y}, y) = -\{y \log \hat{y} + (1-y) \log(1-\hat{y})\}$

i) if $y=1$: $L(\hat{y}, y) = -\log \hat{y}$ ⇒ want $\log \hat{y}$ large ∴ want $\hat{y}$ large ≈ 1

ii) if $y=0$: $L(\hat{y}, y) = -\log(1-\hat{y})$ ⇒ want $\log(1-\hat{y})$ large ∴ want $\hat{y}$ small ≈ 0

* 비용함수(Cost Function): 하나의 입력에 대한 오차를 계산하는 함수가 손실함수이며, 모든 입력에 대한 오차를 계산하는 함수를 비용 함수라고 한다. 즉, 비용함수는 모든 입력에 대한 손실함수의 평균

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y) = -\frac{1}{m}\sum_{i=1}^{m}\{y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\}$$

✩ 중요한 점: 비용함수의 값이 작아지도록 w와 b를 찾는것이 목표! ✩

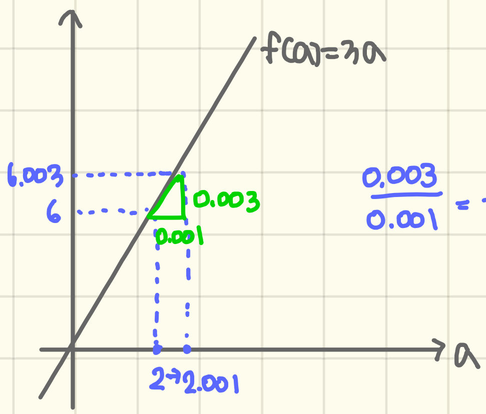
< Gradient Descent >

* 경사하강법: 비용함수를 최소로 만드는 w와 b를 찾는 것으로, 임의의 점을 골라서 기울기를 따라 최적의 값으로 업데이트 함

learning rate

- $w : w - \alpha \boxed{\frac{dJ(w,b)}{dw}}$ 도함수 dw라고도 함
- $b : b - \alpha \frac{dJ(w,b)}{db}$

< Derivatives (미분) >



i) $a=2 \rightarrow f(a)=6$

$a=2.001 \rightarrow f(a)=6.003$

$\frac{0.003}{0.001} = \frac{\text{height}}{\text{width}} =$ $\boxed{\text{slope}}$ (derivative) of f(a) at a=2 : 3

ii) $a=5 \rightarrow f(a)=15$

$a=5.001 \rightarrow f(a)=15.003$

slope at a=5 : 3

* 도함수(=slope): 변수 a를 조금 변화시켰을 때, f(a)가 얼마나 변화하는지 측정하는 것

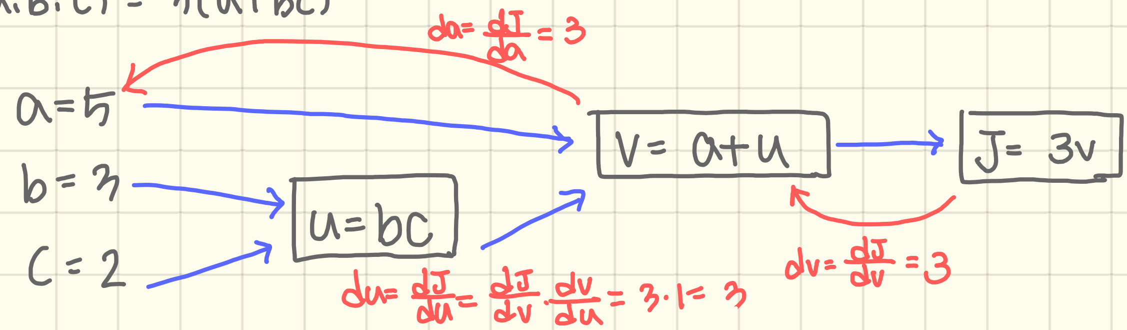
$\frac{d}{da}f(a) = \frac{df(a)}{da}$ 로 표기

< More Derivatives Examples >

함수 : $f(a)$	a^2	a^3	$\ln(a)$
도함수 : $\frac{d}{da}f(a)$	$2a$	$3a^2$	$\frac{1}{a}$

< Computation Graph >

ex) $J(a,b,c) = 3(a+bc)$



< Derivatives with a Computer Graph >

* 미분의 연쇄법칙 : 합성함수의 도함수에 대한 공식

- 위의 표에서 $\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da} = 3 \cdot 1 = 3$ 으로 구할 수 있다.
- $\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} = 3 \cdot c = 3\underbrace{c}_{=2} = 6$ / $\frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{dc} = 3 \cdot \underbrace{b}_{=3} = 9$
- 표기법 : $d\,var = \frac{d\,\text{final output var}}{d\,var}$

< Logistic Regression Gradient Descent >

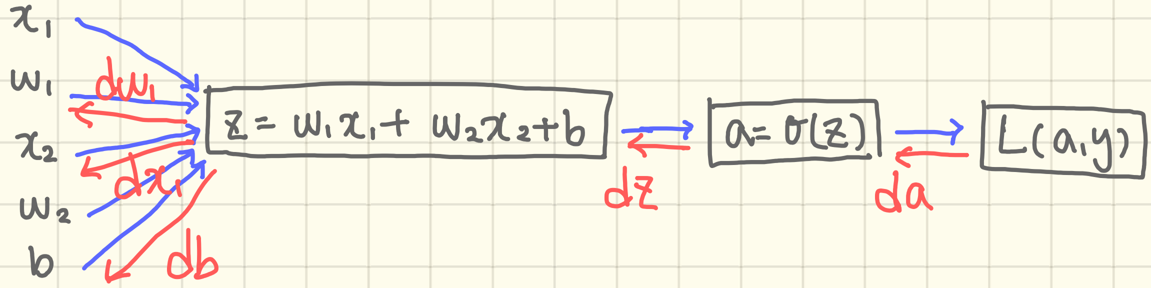
$z = w^T + b$

$\hat{y} = a = \sigma(z)$

$L(a,y) = -\{y\log a + (1-y)\log(1-a)\}$

즉 computer graph로 정리해보면, (편의상 x_1, x_2만 있다고 생각)

⇒



i) $da = \frac{dL(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$

ii) $dz = \frac{dL(a,y)}{dz} = \frac{dL(a,y)}{da} \cdot \frac{da}{dz} = \left\{-\frac{y}{a} + \frac{1-y}{1-a}\right\}\{a(1-a)\}$

$= a - y$

iii) $dw_1 = \frac{dL(a,y)}{dw_1} = x_1 dz$

$dw_2 = x_2 dz$

$db = dz$

$\Rightarrow$ $w_1 := w_1 - \alpha dw_1$
$w_2 := w_2 - \alpha dw_2$
$b := b - \alpha db$
로 업데이트 된다.

<Gradient Descent on m Examples>

$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(a^{(i)}, y^{(i)})$,

$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$

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

For $i=1$ to m,

$z^{(i)} = w^T x^{(i)} + b$
$a^{(i)} = \sigma(z^{(i)})$
$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log (1-a^{(i)})]$
$dz^{(i)} = a^{(i)} - y^{(i)}$
$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$
$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$
$db \mathrel{+}= dz^{(i)}$

특성 개수$(n)=2$라고 가정

[2-2. Python and Vectorization]

<Vectorization>

*벡터화: 코딩에서 for문을 제거하는 기술

ex) $z = w^T x + b$

(Non-vectorization)

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

(Vectorization)

```
z = np.dot(w, x) + b
```

*SIMD(Single Instruction Multiple Data): 병렬 프로세스의 한 종류로, 하나의 명령어로 여러개의 값을 동시에 처리하는 방식으로 벡터 연산을 가능하게 함

ex2) $v = \begin{bmatrix} v_1 \\ \cdots \\ v_n \end{bmatrix}$ $u = \begin{bmatrix} e^{v_1} \\ \cdots \\ e^{v_n} \end{bmatrix}$

(non-vectorization)

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

(vectorization)

```
u= np.exp(v)
```

<Vectorizing Logistic Regression>

$z^{(1)} = w^T x^{(1)} + b$

$a^{(1)} = \sigma(z^{(1)})$

$z^{(2)} = w^T x^{(2)} + b$

$a^{(2)} = \sigma(z^{(2)})$

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

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

...

$\Rightarrow X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$

$[z^{(1)}\ z^{(2)} \cdots z^{(m)}] = w^T X + [b\ b \cdots b]$

$= [\underbrace{w^T x^{(1)} + b}_{= z^{(1)}}\ \ \underbrace{w^T x^{(2)} + b}_{= z^{(2)}}\ \cdots\ \underbrace{w^T x^{(m)} + b}_{= z^{(m)}}]$

(vectorization coding)

```
z= np.dot(np.transpose(w), x) + b
```

< Vectorizing Logistic Regression's Gradient Descent >

i) dz

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

$\Rightarrow dz = [dz^{(1)} \ dz^{(2)} \ \cdots \ dz^{(m)}]$

$A = [a^{(1)} \ a^{(2)} \ \cdots \ a^{(m)}]$

$Y = [y^{(1)} \ y^{(2)} \ \cdots \ y^{(m)}]$

$\Rightarrow dz = A - Y$

$= [a^{(1)} - y^{(1)} \ \ a^{(2)} - y^{(2)} \ \cdots \ a^{(m)} - y^{(m)}]$

ii) dw

$dw = 0$

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

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

$\cdots$

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

$dw \mathrel{/}= m$

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

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

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

iii) db

$db = 0$

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

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

$\cdots$

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

$db \mathrel{/}= m$

$$\Rightarrow db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$$

$$= \frac{1}{m} \text{np.sum}(dz)$$

$$= \text{np.sum}(dz)$$

non - vectorizing

```
J = 0, dw_1 = 0, dw_2 = 0, db = 0
for i = 1 to m:
```

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$\left.\begin{aligned} dw_1 &\mathrel{+}= x_1^{(i)} dz^{(i)} \\ dw_2 &\mathrel{+}= x_2^{(i)} dz^{(i)} \end{aligned}\right\} \Rightarrow dw \mathrel{+}= x^{(i)} * dz^{(i)}$$
$$db \mathrel{+}= dz^{(i)}$$

```
J = J/m, dw_1 = dw_1/m, dw_2 = dw_2/m
db = db/m
```

Vectorizing $\rightarrow$ $Z = w^T X + b = np.dot(w^T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

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

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

$w = w - \alpha dw$

$b = b - \alpha d$

<Broadcasting in Python>

ex1)

$$A = \begin{bmatrix} 56.0 & 0.0 & 4.4 & 68.0 \\ 1.2 & 104.0 & 52.0 & 8.0 \\ 1.8 & 135.0 & 99.0 & 0.9 \end{bmatrix} \rightarrow$$

cal = A.sum(axis=0)

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

$$\underset{(3,4)}{A} \quad \underset{(1,4)}{\text{cal.reshape(1,4)}}$$

⇒ (3,4) 행렬을 (1,4)로 나눔

ex2)

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

ex3)

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

$$(1,n) \xrightarrow[\text{확장}]{} (m,n) \quad \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix}$$

ex4)

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

$$(m,1) \xrightarrow[\text{확장}]{} (m,n) \quad \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix}$$

∴ (m, n) matrix $\begin{pmatrix} + \\ - \\ * \\ \div \end{pmatrix}$ (1,n) ⇝ (m,n), (m,1) ⇝ (m,n) 로 확장하여 계산

Made with Goodnotes

<Explanation of Logistic Regression Cost Function>

Logistic regression cost function

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

$$\rightarrow \hat{y} = P(y=1 \mid x)$$

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

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

$\Rightarrow P(y|x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}$

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

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

로그함수는 단조증가 함수이므로,

$$\underbrace{\log P(y|x)}_{\text{최대화}} = \log\left(\hat{y}^{y}(1-\hat{y})^{(1-y)}\right) = y\log\hat{y} + (1-y)\log(1-\hat{y})$$

$= -L(y,\hat{y})$: $-$손실함수

최소화

$\therefore$ 손실함수 $L(\hat{y}, y) = -\log P(y|x) = -(\hat{y}^{y}(1-\hat{y})^{(1-y)})$

m개의 훈련세트로 확장시키면,

$$P(\text{labels of training set}) = \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)})$$

$\xrightarrow{\text{로그}} \log P(\text{labes of training set}) = \log \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)}) = -\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

$\therefore$ 비용함수 $J(w,b) = -\log P(\text{labels of traing set})$

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