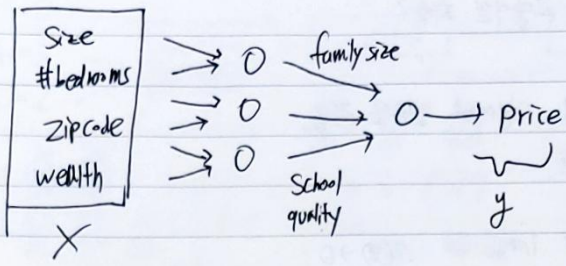
<1주차>

DATE. NO.

<Neural Network>

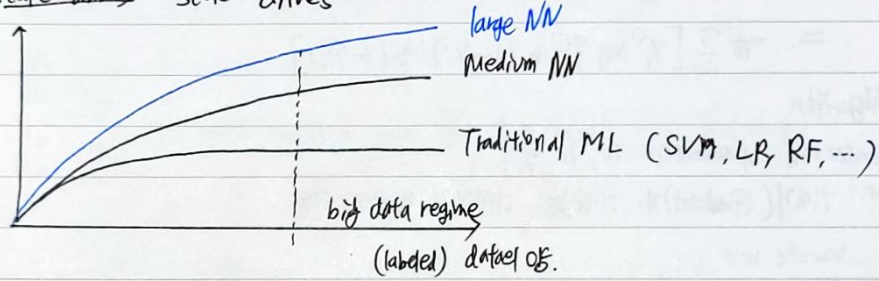


Input(x)	Output(y)	Application	
Image	Object	Photo tagging	← CNN
Audio	Text	음성인식	← t-series, RNN (for sequence data)
Image, radar info	다른차의 위치	자율주행	← Custom, Hybrid NN

▷ Structured data: 명확한 뜻이 정의됨 (size, # of bedrooms, price, ...)

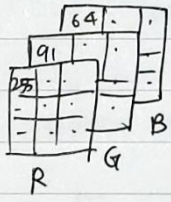
▷ Unstructured data: 명확한 뜻이 정의되지 않음. (→ pixel값, 소리의 tone 등으로 변환해야함).

▷ ~~Scale drives~~ Scale drives



<Binary cf with LR> <2주차>

cat image →



$X = \begin{bmatrix} R \\ \vdots \\ G \\ \vdots \\ B \end{bmatrix}$ (R,G,B 픽셀 채도 모두 나열)

전체 차원:

64×64×3 = 12288

↑ ↑ ↑

x축 y축 RGB

$n_x = 12288$

▷ 학습 데이터 표기

(x, y), $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

m개의 training example: $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$ ... $(x^{(m)}, y^{(m)})$

= m_{train}, m_{test} = # of test example.

$$X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ | & | & | & & | \end{bmatrix}$$ ←신경망 구현시 위와같이 표기해야 쉬움.

= $n_x \times m$ matrix

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

DATE. NO.

▷ Logistic Regression

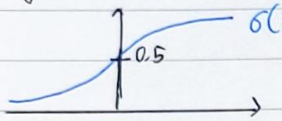
Given x, want $\hat{y} = P(y=1|x)$: 고양이일 확률?

parameter : w, $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output $\hat{y} = \sigma(w^T x + b)$ ← $(w^T x + b)$에 sigmoid 함수를 적용,

$0 \le \hat{y} \le 1$로 만듦.

- Sigmoid function



$\sigma(z) = \frac{1}{1+e^{-z}}$: z large → $\sigma(z) \approx 1$

z small → $\sigma(z) \approx 0$

▷ Cost function of LR 비용함수

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

- Loss function: $L(\hat{y}, y) = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$ ← Cross-entropy loss

① If $y=1$: $L(\hat{y}, y) = -\log \hat{y}$ ← $\log \hat{y}$이 크게, $\hat{y}$가 크게 (1에 최대한 가깝게)

② If $y=0$: $L(\hat{y}, y) = -\log(1-\hat{y})$ ← $\log(1-\hat{y})$이 크게, $1-\hat{y}$가 크게 (0에 최대한 가깝게)

- Cost function: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i, y_i)$ Loss function의 평균

최소화

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]$$

▷ Gradient Descent Algorithm

- $J(w,b)$를 minimize하는 parameter w, b 찾기.
- Initial point에서 기울기(Gradient)가 가장 높은 내려갈 방향으로 이동.

Repeat {

$$w := w - \alpha \underbrace{\frac{dJ(w,b)}{dw}}_{\text{derivative}}$$ ← α: learning rate

$$b := b - \alpha \frac{dJ(w,b)}{db}$$

}

d / ∂ ← partial derivative

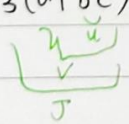
↑ Univariate case

- Chain Rules

$a \xrightarrow[\text{affect}]{} v \to J$ 일때, $\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da}$

▷ 계산 그래프 (산출 그래프, Computational Graph)

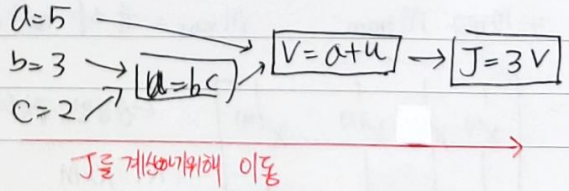
$J(a,b,c) = 3(a+bc)$



$u = bc$

$v = a + u$

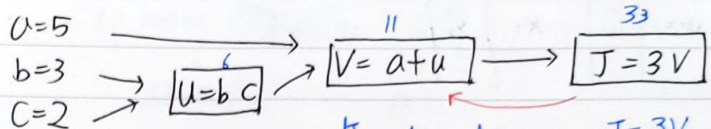
$J = 3v$



↳ J 계산, forward, left to right

DATE. NO.

· Derivative 계산 (← backwards, right to left)



$\frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da}$

$= 3 \cdot 1 = 3$

$\therefore \frac{dJ}{da} = 3 = "da"$

$J = 3V$

$V = 11 \rightarrow 11.001$

$J : 33 \rightarrow 33.003$

$\frac{dJ}{dV} = 3. = "dv"$

$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} =$

$"dc" = \frac{dJ}{dv} \cdot \frac{dv}{du} \cdot \frac{du}{db} = 3 \cdot 1 \cdot C = 3C$

✱ back propagation을 위해 코드를 쓰는 경우,

Final output (제일 선경) variable 이 있으면 (신경그래프의 마지막노드, J)

$\frac{d\,\text{Final output var}}{d\,\text{Var}} = "dvar"$, 최종 결과값 변수의 derivative

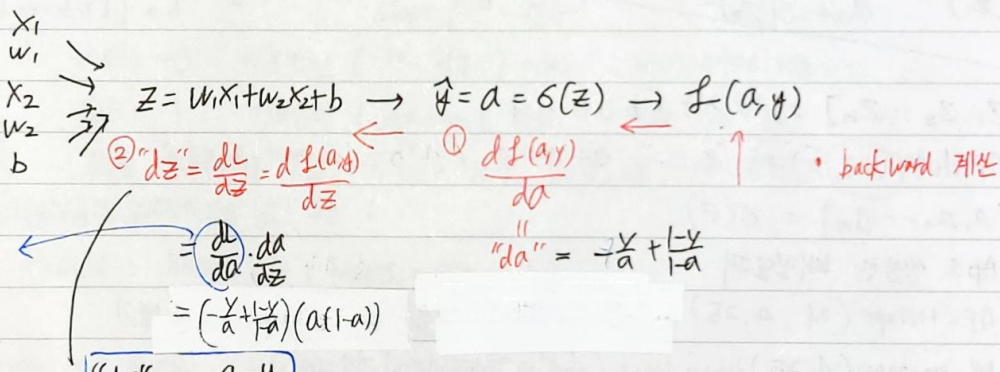
▷ Logistic Regression을 위한 GDM (Single training model)

· $z = w^T x + b$

$\hat{y} = a = \sigma(z) = \frac{1}{1+e^{-z}}$

$\mathcal{L}(a,y) = -(y \log(a) + (1-y)\log(1-a))$, $J(w,b) = \frac{1}{m} \sum \mathcal{L}(a,y)$

·



③ w, b를 얼마나 바꿔야하는지 계산.

$\frac{\partial \mathcal{L}}{\partial w_1} = "dw_1" = x_1 \cdot dz$, $\frac{\partial \mathcal{L}}{\partial w_2} = dw_2 = x_2 \cdot dz$ ···· , $\frac{d\mathcal{L}}{db} = db = dz$

④ Update

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

⋮

DATE. NO.

▷ Vectorization

$z = w^T x + b$ 에서 $w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \end{bmatrix}$, $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$ $w, x \in \mathbb{R}^{n_x}$

- non-vectorized:

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

- Vectorized - numpy 사용
  - $z = \underbrace{np.dot(w,x)}_{w^T x} + b$ $\leftrightarrow$ $np.outer(u,v)_{i,j} = uv^T$: ixj matrix
  - $u = e^v$ → `u = np.exp(v)`
  - $u = \log v$ → `u = np.log(v)`

▷ Vectorizing LR

- m 개의 training example에서

$z_1 = w^T x_1 + b$, $(\hat{y}_1 =) a_1 = \sigma(z_1)$; $z_2 = w^T x_2 + b$, $a_2 = \sigma(z_2)$; ...

$X = \begin{bmatrix} | & | & | & & | \\ x_1 & x_2 & x_3 & \cdots & x_m \\ | & | & | & & | \end{bmatrix}$, $w^T = [w_1, w_2, \cdots w_n]$, $b = [b\ b\ b \cdots b]$

→ $[z_1\ z_2 \cdots z_m] = w^T X + [b\ b\ b \cdots]$

- `Z = np.dot(w.T, X) + b` (.T : 전치 method / a*b : 'element-wise' 곱, 같은 크기의 행렬 끼리만 적용 가능)
- $A = [a_1, a_2 \cdots a_m] = \sigma(z)$
- M = np로 생성한 배열일 때

```
N = np.reshape(M, (4,25))   4x25 배열로 생성 (2차원 배열 - (행, 열))
N = M.reshape(4, 25)        / '-1' : 형태에 맞게 자동설정
```

ex) Numpy 에서 벡터 생성

```
np.array([1,2,3])      // 1차원 배열
np.array([[1,2,3]])    // 2차원 배열 - row vector
np.array([[1],[2],[3]]) // 2차원 배열, column vector
```

▷ Broadcasting → 형태를 맞추기 위해 copy됨.

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

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$

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

DATE. NO.

✳ , 벡터생성시 rank 1 array는 만들지 말것.

- np.random.rand(5) → (5,1)
  - .shape: (5,)
  - (rank 1 array)
  - (5,1)
- np.random.rand(a,b,c)
  : 입력된 숫자 차원의 random number array를 생성.

- assert(a.shape == (5,1)) ← dimension을 정확히 모를 경우.
- 3차원 배열: .reshape(높,행,열)
- np.dot(a,b), a = m×n matrix, b = n×k matrix

$m\left(\begin{matrix}\cdot & \cdot \\ \cdot & \cdot\end{matrix}\right)_{m\times n}\left(\begin{matrix}\cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot\end{matrix}\right)_{n\times k} = m\left(\begin{matrix}\cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot\end{matrix}\right)_{m\times k}$ → dot: m×k matrix

- algorithm의 input으로 image의 벡터 생성시, (length*height*3, 1)의 형태로 reshaping 필요.
- .shape method는 tuple을 반환.
  - 3차원 array에서 shape[0]=높수, shape[1]=행수, shape[2]=열수
  - 2차원: shape[0]=행수, shape[1]=열수

- axis=0 : 세로방향 (row 방향) 계산 : each column 계산
  axis=1 : 가로방향 (column 방향) 계산 : each row 계산
  (3차원 데이터에서는 axis=0,1,2는 각각 높,행,열 방향 계산)

▷ Jupyter 실습

① EDA) train_x : image of (height, width, 3)
↓ num_px ↓ num_px

- train_set_x_orig : ndarray, shape of (m_train, height, width, 3) // dimension, shape 찾기

X → X_flatten = X.reshape(X.shape[0], -1).T // reshaping
(a,b,c,d) → (b·c·d, a)
'a' ↑, b·c·d ↑, transpose ↑

- standardize
  - $\frac{x-\bar{x}}{\sigma}$
  - picture dataset에는 대신 pixel channel의 최댓값인 255로 나누는것이 편함.

② Building NN
1. model의 구조 정의
2. parameter 최적화
3. Loop : · current loss - forward propagation
  · current gradient - back propagation
  · Update parameters - GD.

DATE. NO.

- np.zeros(shape), shape = (·,·) 형태로 대입할 것.
- isinstance(x, float) - 자료형 검사, True or false 반환
  int
  str
  list
  dict
  :
- Forward / Backward Propagation
  ① Forward
  - get X
  - Compute $A = \sigma(w^T X + b)$ $(= (a_1, a_2 \cdots a_m))$
  - Calc cost function $J = -\frac{1}{m}\{\sum_{i=1}^{m} y_i \log(a_i) + (1-y_i)\log(1-a_i)\}$

  ② Backward
  - $dw = \frac{\partial J}{\partial w} = \frac{1}{m} X(A-Y)^T$ // (1/m) · np.dot(X, ((A-Y).T))
  - $db = \frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i)$ // (1/m) · np.sum(A-Y)

- Update
  - $w = w - dw *$ learning rate
    $b = b - db \cdot$ learning rate

▷ 사용된 함수

S ← · Sigmoid(z)

X_train.shape[0] = n_x

W,b ← · Initialize_with_zeros(dim) : (dim,1)의 0행렬 w, b=0로 초기화

grads, cost ← · propagate(w,b,X,Y) : cost function 계산, 이에따른 dw와 db 계산. → grads={dw,db}, cost

params, grads, costs ← · Optimize(w,b,X,Y, num_iter, learning_rate) : w와 b update
  - propagate로부터 grad, cost 받음
  - w,b update
  - params (w,b의 dict)와 grads(dw,db의 dict), costs 반환

Y_prediction ← · predict(w,b,X) : 최종 (w,b)를 사용해서 X의 클래스 예측.

- w = $n_x \times 1$ matrix
- X : $n_x \times m$
- A : $w^T X$과 차원 동일 → $1 \times m$ matrix

* Cost 함수 시각화 = 얼마나 잘 학습했는지 판단.

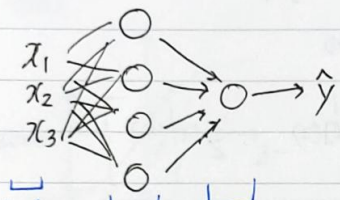
DATE. NO.

[3주차. Shallow neural network]

x, $W^{[1]}$, $b^{[1]}$ → $(z^{[1]} = W^{[1]}x + b^{[1]}) \rightarrow (a^{[1]} = \sigma(z^{[1]})) \rightarrow z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \rightarrow a^{[2]} \rightarrow L(a^{[2]}, y)$

$dW^{[2]} \leftarrow W^{[2]}$, $db^{[2]} \leftarrow b^{[2]}$; $z^{[2]} \rightarrow dz^{[2]}$; $a^{[2]} \rightarrow da^{[2]}$

▷ NN Representation



- 전체 layer수 : hidden# + Output#

Input layer = $a^{[0]}$ = 'activation'

hidden layer = $a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$

output layer = $a^{[2]}$ (1, 1)

Input = $a^{[1]}$

$W^{[2]}$, $b^{[2]}$

(1,4) (1,1)

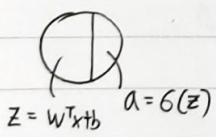
$W^{[1]}$, $b^{[1]}$

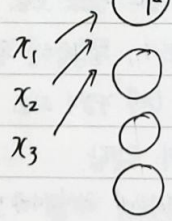
(4,3) (4,1)

nodes ← Input features

▷ Computing Output

- hidden layer의 node





$z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]}$

$a_1^{[1]} = \sigma(z_1^{[1]})$

$z_2^{[1]} = w_2^{[1]T}x + b_1^{[1]}$

$a_2^{[1]} = \sigma(z_2^{[1]})$

$a_i^{[l]}$ ← l: # layer, i: ith node

- Vectorizing

$$z^{[1]} = \begin{bmatrix} -w_1^{[1]T}- \\ -w_2^{[1]T}- \\ -w_3^{[1]T}- \\ -w_4^{[1]T}- \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T}x + b_1^{[1]} \\ \vdots \\ w_4^{[1]T}x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ \vdots \\ z_4^{[1]} \end{bmatrix} \xrightarrow{\sigma} \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$

(4,3) = $W^{[1]}$ (3,1) (4,1) = $b^{[1]}$ (4,1) (4,1)

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \xrightarrow{\sigma} a^{[2]} = \hat{y}$$

(1,1) (1,4) (4,1) (1,1) (1,1)

DATE. NO.

- multiple training examples 경우에는

$X \rightarrow a^{[2]} = \hat{y}$

↳ $x^{(1)} \rightarrow a^{[2](1)}$

$\vdots \rightarrow \vdots$

$x^{(m)} \rightarrow a^{[2](m)}$

for i=1 to m:

$$\begin{cases} z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]} \\ a^{[1](i)} = \sigma(z^{[1](i)}) \end{cases}$$

$$\begin{cases} z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]} \\ a^{[2](i)} = \sigma(z^{[2](i)}) \end{cases}$$

→ $z^{[l]} = W^{[l]}X + b^{[l]}$

$A^{[l]} = \sigma(z^{[l]})$

$W^{[l]} = \begin{bmatrix} \equiv \end{bmatrix}$, l이다 다른 값을 가짐.

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

$\downarrow z = wx + b$

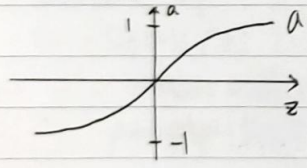
$Z^{[l]} = \begin{bmatrix} | & | & & | \\ z^{[l](1)} & z^{[l](2)} & \cdots & z^{[l](m)} \\ | & | & & | \end{bmatrix}$ nodes index, training set의 index

$A^{[l]} = \begin{bmatrix} | & | & & | \\ a^{[l](1)} & a^{[l](2)} & \cdots & a^{[l](m)} \\ | & | & & | \end{bmatrix}$ # of hidden unit

▷ Activation Function

$a = g(z)$, g는 non-linear 함수

① tanh

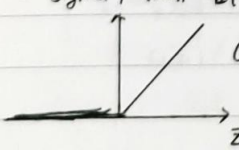


$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

- 항상 Sigmoid Function보다 좋은 성능을 보임.
  : 값이 -1~1 사이에 위치하면서, 활성함수의 평균값이 hidden layer에서 계산될 때 0에 가깝게 때문.
- 데이터를 0으로 centering 할 때 유용함.
- 하지만 출력층의 경우, $\hat{y}$가 0~1사이에 위치해야하기 때문에 binary classification의 output layer에는 Sigmoid 사용.

② Relu

- Sigmoid / tanh 함수에서 z가 매우 크거나 작은경우에 기울기가 0에 가까워질 보완.



$a = \max(0, z)$

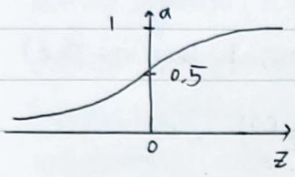
* hidden layer에서는 Relu, Output layer에서는 Sigmoid 사용 (binary clf)

DATE. NO.

▷ Derivatives (slope) of Activation Functions

· Sigmoid

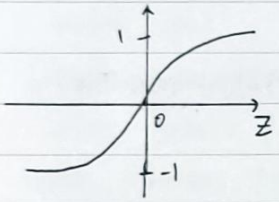
$g(z) = \frac{1}{1+e^{-z}}$ → $\frac{d}{dz} g(z)$: slope of $g(x)$ at z $= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$



$= g(z)(1-g(z))$

$= a(1-a)$ where $a = \hat{y}$

◦ tanh



$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \tanh(z)$

$\frac{d}{dz} g(z) = 1 - (\tanh(z))^2$

$z = 10$ $\tanh(z) \simeq 1$, $g'(z) \simeq 0$

$z = -10$ $\tanh(z) \simeq -1$, $g'(z) \simeq 0$

$z = 0$ $\tanh(z) = 0$, $g'(z) = 1$

◦ Relu and Leaky Relu

$g(z) = \max(0, z)$

$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

(Leaky)

$g(z) = \max(0.01z, z)$

$g'(z) = \begin{cases} 0.01 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

▷ Gradient Descent of Neural Network

· 1개의 hidden layer의 경우

- parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$ → $W^{[1]} = (n^{[1]}, n^{[0]})$ vector

$n_x = n^{[0]}, n^{[1]}, n^{[2]}$

$b^{[1]} = n^{[1]} \times 1$

$W^{[2]} = n^{[2]} \times n^{[1]}$

$b^{[2]} = n^{[2]} \times 1$

- Cost functions: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$

$\hat{y} = a^{[2]}$

- Gradient Descent: Repeat {

Compute prediction $(\hat{y}^{(i)}, i = 1 \sim m)$

$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, $dW^{[2]} = \frac{\partial J}{\partial W^{[2]}}$, ...

Update $W^{[1]} = W^{[1]} - \alpha \cdot dW^{[1]}$

$b^{[1]} = b^{[1]} - \alpha \cdot db^{[1]}$

$W^{[2]}$

⋮

}

DATE. NO.

· $dw = dz \cdot \frac{\partial z}{\partial w}$]
· $db = dz \cdot \frac{\partial z}{\partial b}$

· Formula ☆

forward propagation)

$z^{[1]} = w^{[1]}x + b^{[1]}$
$A^{[1]} = g^{[1]}(z^{[1]})$
$z^{[2]} = w^{[2]}A^{[1]} + b^{[2]}$
$A^{[2]} = g^{[2]}(z^{[2]})$

$z^{[l]} = w^{[l]}A^{[l-1]} + b^{[l]}$
$A^{[l]} = g^{[l]}(z^{[l]})$

Back propagation)

$dz^{[2]} = A^{[2]} - Y$
$dw^{[2]} = \frac{1}{m} \cdot dz^{[2]} A^{[1]T}$
$db^{[2]} = \frac{1}{m} \cdot np.sum(dz^{[2]}, axis=1, keepdims=True)$ ($n^{[2]}$,1)로 dim 고정.
$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$ //element-wise product
($n^{[1]}$, m) ($n^{[1]}$, m)
$dz = da \cdot g'(z)$
$dw^{[1]} = \frac{1}{m} dz^{[1]} x^T$
$db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims=True)$

· 신경망 그래프 (Single example)

x, $w^{[1]}$, $b^{[1]}$ → $z^{[1]} = w^{[1]}x + b^{[1]}$ → $a^{[1]} = \sigma(z^{[1]})$ → $z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$ → $a^{[2]} = \sigma(z^{[2]})$ → $\mathcal{L}(a^{[2]}, y)$
($w^{[2]}$, $b^{[2]}$ ↘)

$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$
$dz^{[2]} = a^{[2]} - y$
$dw^{[2]} = dz^{[2]} a^{[1]T}$
$db^{[2]} = dz^{[2]}$
← /m은 multi-example 일때.

dim
$w^{[2]} = (n^{[2]}, n^{[1]})$
$z^{[2]}, dz^{[2]} = (n^{[2]}, m)$
$z^{[1]}, dz^{[1]} = (n^{[1]}, m)$
$w^{[1]} = (n^{[1]}, n^{[0]})$

$b^{[2]} = (n^{[2]}, 1)$
$b^{[1]} = (n^{[1]}, 1)$

$(n^{[1]}, m) = (n^{[1]}, n^{[2]}) \cdot (n^{[2]}, m) * (n^{[1]}, m)$

$dw^{[1]} = dz^{[1]} \cdot x^T$
$db^{[1]} = dz^{[1]}$

▷ Random Initialization ← 'break symmetry'로 뉴런이 같은 것을 배우지 않도록 설정.

· Weight - 난수생성, 아주 작은 값으로 initialize / 0일경우 모든 hidden Unit이 symmetric 이 되므로 오류.

Weight $w^{[1]} = np.random.randn((n^{[1]}, n^{[0]})) \times 0.01$, $n^{[0]} = n_x$
bias $b^{[1]} = np.zeros((n^{[1]}, 1))$ ↳ layer의 개수에 따라 달라짐.

$w^{[2]}$, $b^{[2]}$ ... 도 동일하게 Initialize.

* $a_4^{[2]}$: 2nd layer의 4th neuron의 activation output

$a^{[2](4)}$: 2nd layer의 4th training example의 activation output

DATE. NO.

▷ 실습

- def layer_sizes(X, Y)
  return: n_x, n_h, n_y
- initialize_parameters(n_x, n_h, n_y) : W1 ~ b2 계산
  return: parameters = { }
- forward_propagation(X, parameters) : A2와 cache(Z, A의 dictionary)
  return: A2, cache = { }
  [hidden: tanh
  output: sigmoid
- Compute_cost(A2, Y, parameters)
  return: cost
- backward_propagation(parameters, cache, X, Y) : dW, db의 dict, grad 계산.
  return: grads = { }
- Update_parameters(parameters, grads, learning_rate) : parameter dict를 update.
  return: parameters = { }
- nn_model(X, Y, n_h, num_iterations)
  - Initialize parameters
  - loop
    - Forward prop
    - Cost computing
    - back propagation
    - Update params with GD

  return : parameters = { }
- predict(parameters, X) : 산출된 parameter를 forward propagation하여 A2, cache 산출 후 $\hat{Y}$ 계산.

DATE. NO.

<4주차, Deep Neural Network>

▷ Notation

$L = 4$ (# layers)

$n^{[l]}$ = # units in layer l

$a^{[l]} = g^{[l]}(z^{[l]})$, $a^{[0]} = X$, $a^{[L]} = \hat{y}$

- layer가 깊어질수록 더 복잡한 feature의 연산 수행.

▷ Forward propagation

- 1부터 L-th layer까지 for loop 사용,

$$z^{[l]} = w^{[l]} A^{[l-1]} + b^{[l]} \quad \rightarrow \text{'cache'로 저장, 나중의 backward step에 쓰임}$$
$$A^{[l]} = g^{[l]}(z^{[l]})$$

▷ Parameter의 dimension

$X : (n^{[0]}, 1)$

$z^{[1]} = w^{[1]} x + b^{[1]}$

$(n^{[1]}, 1) \quad (n^{[1]}, n^{[0]}) \quad (n^{[0]}, 1) \quad (n^{[1]}, 1)$

$z^{[l]} = (n^{[l]}, m) \rightarrow a^{[l]} = (n^{[l]}, m)$

$w^{[2]} = (n^{[2]}, n^{[1]})$

$w^{[l]} = (n^{[l]}, n^{[l-1]})$ $\quad b^{[l]} = (n^{[l]}, 1)$ broadcasted

dZ, dA : Same dimension as Z, A

* Shallow, many hidden nodes < Deep, less hidden nodes.

▷ Forward & Backward functions

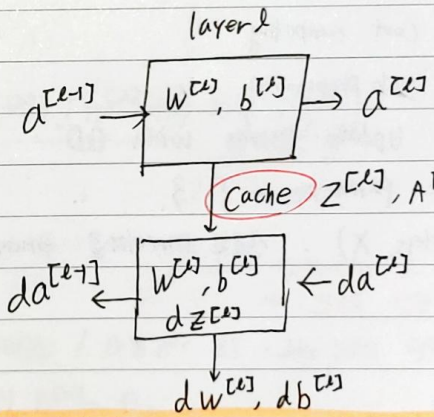
Layer l : $w^{[l]}, b^{[l]}$

Forward :

- Input : $a^{[l-1]}$
- Output : $a^{[l]}$, 'cache' $z^{[l]}$

Backward :

- Input : $da^{[l]}$, cache($z^{[l]}$) $+ w^{[l]}, b^{[l]}$
- Output : $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$



$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$
$$db^{[l]} = dz^{[l]}$$
$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

(vectorized)

$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]}) = W^{[l+1]T} \cdot dZ^{[l+1]} * \sim$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} \cdot np.sum(dZ^{[l]}, axis=1, keepdims=True)$$
$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

▷ Parameters : W, b

Hyper parameters : α
- # iter
- # hidden layers, # hidden units
- activation function

DATE. NO.

▷실습

#1. · initial_parameters_deep(layer_dims) : W, b 에 대한 dict, parameter return.

· model : [Linear part (L-1 layer) + Relu] + Sigmoid part (Output layer)

· linear_forward (A, W, b) : return Z, cache = (A, W, b) L-1개, 0~L-1

· linear_activation_forward (A_prev, W, b, activation)

- helper func: sigmoid, relu : input Z, return A, activation_cache

- return A, cache (= linear_cache, activation_cache)

· L_model_forward (X, parameters) : linear_activation forward with relu n-1 번 진행

→ with sigmoid 1번 진행.

- $l=1$ to $l=L-1$ 까지 A_prev = A, linear_activ_forward (A_prev, parameters['Wl'], P~['bl'], act=relu)

$l=L$ 에서 linear_activ_forward (A, ~, activ= sigmoid) 진행 → AL, caches 반환

· linear_backward (dZ, cache) : return dA_prev, dW, db

↳ (A_prev, W, b)

· linear_activation_backward (dA, cache, activation) : return dA_prev, dW, db

* cache를 linear_cache와 activation_cache로 나누어서 대입

* linear, activation 부분 한꺼번에 backward 하는 함수.

· L_model_backward (AL, Y, caches)

· dAL계산

· L → L-1 layer : sigmoid

L-2 → 0 : relu

· $dA^{[l]}$ input → Output $dA^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$

· update_parameters (parameters, grads, learning_rate) : return parameters