

Binary classification



→ C (cat)
VS D (non cat)
y -

유튭 같이 고양이 사진이 있다고 하자!

이미지 인식을 위해 고양이 일 때는 1, 아닐 때는 0으로 예제를 출격하고자 함.

출격 레이블을 나타내기 위해 y를 사용.

이미지를 나타내기 위해 컴퓨터는 R, G, B 채널에 해석하는 세 가지의 보이드 행렬을 사용.

		→ Blue		
		→ Green		
→ Red		255	134	93
64		255	134	202
		255	231	42
		123	94	83
		34	44	187
		34	76	232
		67	83	194
64		22	2	22
		4	30	124
		34	142	94
		67	83	202

이미지의 size가 64×64 이고 했을 때,
열의 이미지와 같이 64×64 행렬이 각 R, G, B.
3개의 채널을 표현된다.
 $64 \times 64 \times 3 = 12,288$.

이진 분류 (Binary Classification)에서 목표는 특징 벡터 x 로 표현된 이미지를 정답할 수 있는
분류기를 통해 y 가 1인지 0인지 (고양이인지, 아님지)를 예측하는 것!

Notation

n 개의 training examples : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(n)} \\ | & | & & | \end{bmatrix}^T \quad n_x \quad Y = [y^{(1)}, y^{(2)}, \dots, y^{(n)}]$$

$\underbrace{\hspace{1cm}}_m$

$$X.\text{shape} = (n_x, m)$$

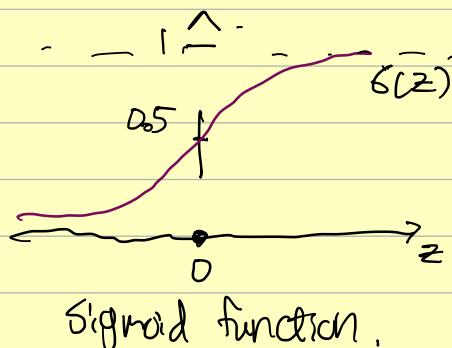
$$Y.\text{shape} = (1, m).$$

Logistic Regression

- 지도학습에서 예측값 y 가 0이거나 1인 경우 사용하는 알고리즘.
- Given $x \in \mathbb{R}^n$; we want to estimate $\hat{y} = P(y=1 | x)$
 $0 \leq \hat{y} \leq 1$

Parameters : $w \in \mathbb{R}^m$, $b \in \mathbb{R}$.

Output $\hat{y} = \sigma(w^T x + b)$.



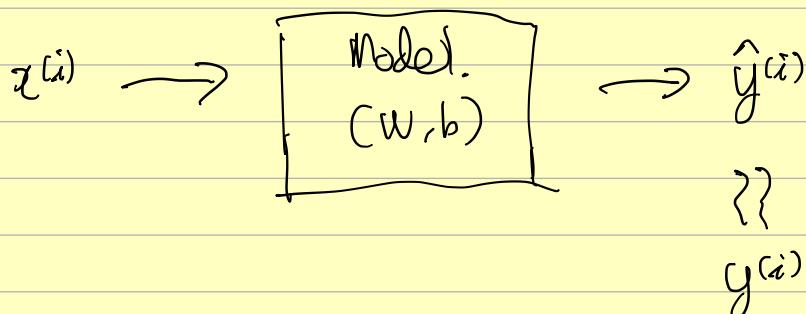
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

z 가 크면 $\sigma(z) \approx \frac{1}{1+0} = 1$.

z 가 작으면 $\sigma(z) \approx \frac{1}{1+\infty} \approx 0$.

Sigmoid function.

- x 를 통해 주어진 확률 y hat (\hat{y})을 estimate하고 싶다.
- \hat{y} 을 estimate하기 위해 필요한 parameter는 w 와 b 이다.



Logistic regression cost function

Loss function. : $L(\hat{y}, y)$

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

binary-cross entropy loss

• If $y=1 \Rightarrow L(\hat{y}, y) = -\log \hat{y} \Leftarrow$ want $\log \hat{y}$ large,
want \hat{y} large.

• If $y=0 \Rightarrow L(\hat{y}, y) = -\log(1-\hat{y}) \Leftarrow$ want $\log(1-\hat{y})$ large.
want \hat{y} small.

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

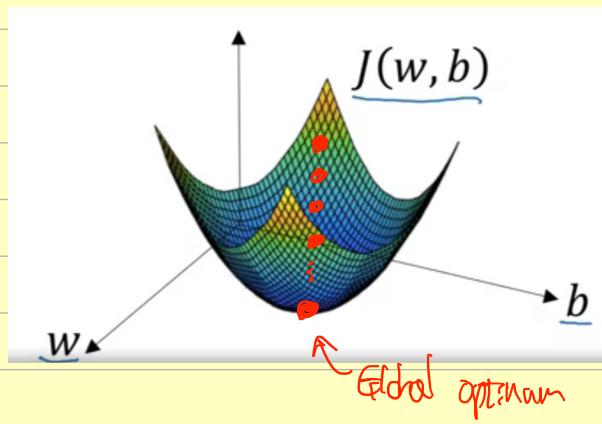
$$= -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$$

Gradient Descent

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

cost function을 $J(w, b)$ 라 할 때, $J(w, b)$ 를 최소화하는 w 와 b 를 찾는 방법인데,

⇒ 미지의 training data가 존재할 때, w 가 너무 크면 전체를 다 미분하기
너무 힘들기 때문에 gradient descent method를 사용한다.



cost function J 는 볼록함수.

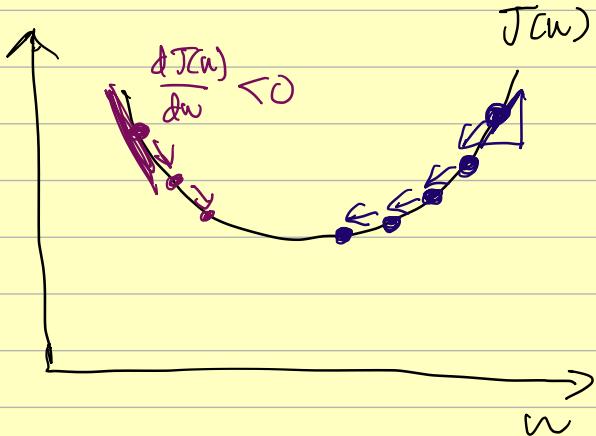
한 점에서 미분해서 기울기를 찾고, 기울기
방향으로 이동. ⇒ 그 점에서 미전보다 기울기가
줄어들고, 그 점에서 미분하고 기울기를 찾아
이동 → 위 과정을 반복
⇒ 최종값에 도달.

Repeat ⌈

$$w \leftarrow w - \alpha \frac{dJ(w, b)}{dw}$$

$$b \leftarrow b - \alpha \frac{dJ(w, b)}{dw}$$

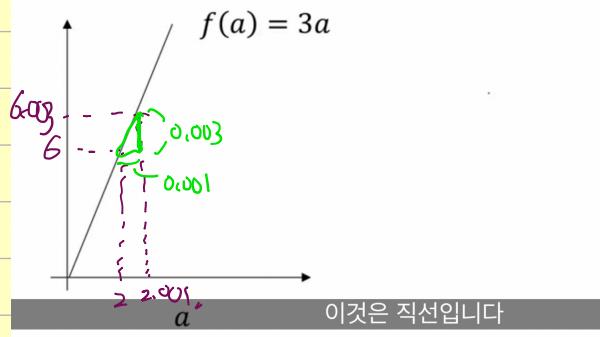
⌉



⇒ J 가 w 에 따른 함수라고만 생각했을
때 (2차원이라고 생각하고)
그려본 그림.

Derivatives

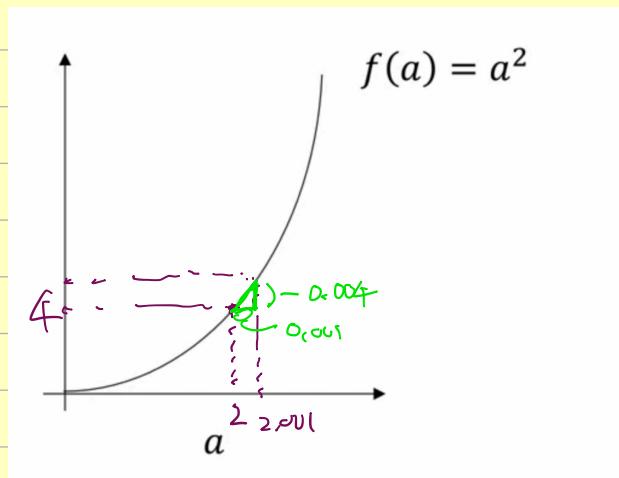
Intuition about derivatives



$$a=2 \Rightarrow f(a)=6.$$

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

$$\text{slope} = \frac{\text{height}}{\text{width}} = \frac{0.003}{0.001} = 3.$$



$$a=2 \Rightarrow f(a)=4$$

$$a=2.001 \Rightarrow f(a) \approx 4.004$$

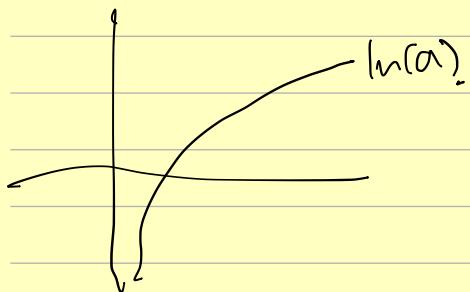
$$\text{slope} \Rightarrow a=2 \text{ 일 때 } \frac{d}{da} f(a) = 4$$

$$\text{if } a=5 \Rightarrow \frac{d}{da} f(a) = 10$$

$$\frac{d}{da} f(a) = \frac{d}{da} a^2 = 2a.$$

$$f(a) = \ln(a)$$

$$\frac{d}{da} f(a) = \frac{d}{da} \ln(a) = \frac{1}{a}$$



Computation graph

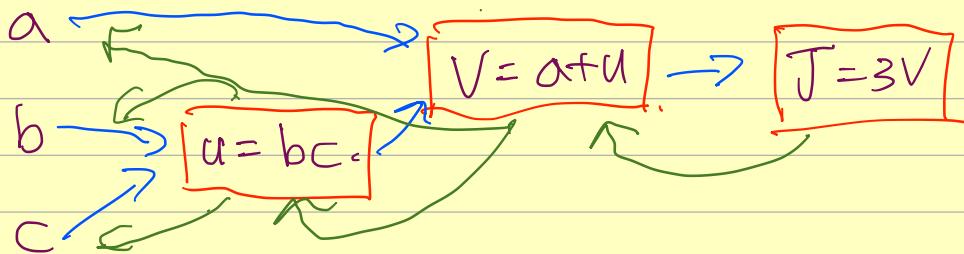
$$J(a, b, c) = 3(a + bc)$$

\underbrace{bc}_{v}

$u = bc$

$$v = a + u$$

$$J = 3v$$



$a=5, b=3, c=2$ 를 입력하면 $u=6, v=11, J=33$.

gradient method는 결국 output인 J 를 a, b, c 에 대해 미분한 값을 구하는 것.

따라서 최종적으로 $\frac{dJ}{da}, \frac{dJ}{db}, \frac{dJ}{dc}$ 를 구해야 한다.

미분값을 위한 backward pass는 두번짼 시작한다.

$$J = 3v \text{에서 } \frac{dJ}{dv} = 3 \text{이다.}$$

이를 통해 $v = a + u$ 에서 $\frac{dJ}{da}$ 를 Chain-rule을 이용하여 구할 수 있다.

$$\boxed{\frac{dJ}{da} = \frac{dJ}{dv} \times \frac{dv}{da} = 3 \times 1 = 3}$$

이전 stage에서 구한 gradient

재활용해 새로운 gradient를 구하는 방식으로 이를

이와 같은 방식으로

$$\frac{dJ}{du} = \frac{dJ}{dv} \times \frac{dv}{du} = 3 \times 1 = 3$$

$$\frac{dJ}{db} = \frac{dJ}{du} \times \frac{du}{db} = 3 \times c = 3 \times 2 = 6,$$

backpropagation이다.

$$\frac{dJ}{dc} = \frac{dJ}{du} \times \frac{du}{dc} = 3 \times b = 3 \times 3 = 9, \text{ 를 계산할 수 있다.}$$

Logistic regression gradient descent

$$z = w^T x + b \rightarrow \text{linear / affine function}$$

$$\hat{y} = a = \sigma(z) \rightarrow \text{nonlinear function}$$

$$L(a, y) = -[y \log(a) + (1-y) \log(1-a)] \rightarrow \text{loss for one example}$$

We want to modify the parameters w and b in order to reduce the loss.

$$x_1, x_2, w_1, w_2, b$$

$$z = w_1 x_1 + w_2 x_2 + b$$

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

$$L(a, y)$$

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

$$\frac{da}{dz} = \frac{dL(a,y)}{da}$$

$$= -\frac{y}{a} + \frac{1-y}{1-a}$$

$$= a - y$$

$$= \left[\frac{dL}{da} \right] \cdot \left[\frac{da}{dz} \right] \rightarrow \text{partial}$$

$$J = - \{ y \ln(\sigma(z)) + (1-y) \ln(1-\sigma(z)) \}$$

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

$$\frac{\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$

$$= a(1-a)$$

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

$$J = - \{ y [\ln a + (1-y) \ln(1-a)] \}$$

$$\frac{dJ}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{dJ}{dz} = \left(-\frac{y}{a} + \frac{1-y}{1-a} \right) \cdot a(1-a)$$

$$= \frac{-y(1-a) + a(1-y)}{a(1-a)} \cdot a(1-a)$$

$$= -y + ay + a - ay = a - y.$$

$$\frac{dJ}{dz} = a - y \text{ 이므로 두번가 알고자하는 } \frac{dL}{dw_1}, \frac{dL}{dw_2}, \frac{dL}{db}$$

계산할 수 있다.

$$\frac{dJ}{dz} = \frac{dL(a, y)}{dz} = a - y, z = w_1x_1 + w_2x_2 + b,$$

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

$$\frac{dL(a, y)}{dw_2} = \frac{dL(a, y)}{dz} \cdot \frac{dz}{dw_2} = (a - y) x_2$$

$$\frac{dL(a, y)}{db} = \frac{dL(a, y)}{dz} \cdot \frac{dz}{db} = (a - y) \cdot []$$

이를 gradient method의 적용할 때는 w_1, w_2, b 각각에对自己的

$\frac{dL}{dw_1}, \frac{dL}{dw_2}, \frac{dL}{db}$ 및 학습률 learning rate를 곱한 값을 더해서 update

$\Rightarrow m$ 개의 example을 처리할 때, m 개 example에 대한 미분값의 평균

Python vectorization

- Vectorization은 통계 for loop을 제거할 수 있다.

```
[6]: import time

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

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

print(c)
print(f"Vectorized version : {1000 * (toc - tic)}ms")

c = 0
tic = time.time()

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

print(c)
print(f'For loop : {1000*(toc-tic)}ms')
executed in 276ms, finished 17:45:09 2023-09-10
249734.7759955873
Vectorized version : 1.3530254364013672ms
249734.77599558095
For loop : 244.0798282623291ms
```

For Loop을 사용했을 때와 Vectorization 코드를 사용했을 때,
시간 차이가 약 200배 이상 나는 것을 볼 수 있다.

* 가능하면 대량 계산을 통해 Vectorization 코드를 사용하는 것이 좋다.

$$u = Av$$

$$u_i = \sum_j A_{ij} v_j$$

$$u = np.zeros([n, 1])$$

for i in range(n):

 for j in range(n):

$$u[i] += A[i][j] * v[j]$$

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

numpy의 dot 함수를

사용하여 for loop을

사용할 수 있다.

Logistic regression derivatives

```
J = 0, dw1 = 0, dw2 = 0, db = 0
→ for i = 1 to m:
    z(i) = wTx(i) + b
    a(i) = σ(z(i))
    J += -[y(i) log a(i)) + (1 - y(i)) log(1 - a(i))]
    dz(i) = a(i) - y(i)
    dw1 += x1(i) dz(i)
    dw2 += x2(i) dz(i)
    db += dz(i)
J = J/m, dw1 = dw1/m, dw2 = dw2/m, db = db/m
```

$$dw = np.zeros((n+1, 1))$$

$$dw \leftarrow x⁽ⁱ⁾ dz⁽ⁱ⁾$$

$$dw / = m$$

↳ 기존 logistic regression의 디자인 행렬을 구하는 공식.

python numpy의 내장함수 np.zeros 를 활용하여

기존 2개의 for loop을 대체할 수 있다.

Vectorizing Logistic Regression.

- Vectorization은 행렬을 세로 벡터로 바꾸는 선형 변환.

$m \times n$ 행렬 A의 선형변환 $v = (A)$ 로 표기하고, 행렬 A의 열을 다음 블록에 쓰여야 할 수 있다.

이러한 Vector화의 장점은 for-loop과 동일한 결과를 얻는.

알고리즘을 내장 함수를 통해 빠르고 효율적으로 구현할 수 있다.

Vectorizing Logistic Regression

$$\rightarrow z^{(1)} = w^T x^{(1)} + b \quad z^{(2)} = w^T x^{(2)} + b \quad z^{(3)} = w^T x^{(3)} + b$$

$$\rightarrow a^{(1)} = \sigma(z^{(1)}) \quad a^{(2)} = \sigma(z^{(2)}) \quad a^{(3)} = \sigma(z^{(3)})$$

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_m^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_m^{(n)} \end{bmatrix}$$

$$w^T \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_m^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_m^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_m^{(n)} \end{bmatrix}$$

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

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

$$A = [a^{(1)}, a^{(2)}, \dots, a^{(n)}] = \sigma(Z)$$

Vectorizing logistic regression's gradient output

Vectorizing Logistic Regression

$$\begin{aligned}
 dz^{(1)} &= a^{(1)} - y^{(1)} & dz^{(2)} &= a^{(2)} - y^{(2)} & \dots \\
 \underline{\underline{d\bar{z}}} &= \left[\underline{dz^{(1)}} \ \underline{dz^{(2)}} \ \dots \underline{dz^{(m)}} \right]_{1 \times m} & \leftarrow & & \\
 A &= [a^{(1)} \ \dots \ a^{(m)}] & Y &= [y^{(1)} \ \dots \ y^{(m)}] \\
 \rightarrow d\bar{z} &= A - Y = [a^{(1)} - y^{(1)} \ \ a^{(2)} - y^{(2)} \ \ \dots]
 \end{aligned}$$

$\frac{\partial w}{\partial z^{(1)}} = \frac{\partial w}{\partial a^{(1)}} = \frac{\partial w}{\partial x^{(1)}} = \frac{\partial w}{\partial z^{(2)}} = \dots = \frac{\partial w}{\partial z^{(m)}} = \frac{\partial w}{\partial a^{(m)}} = \frac{\partial w}{\partial x^{(m)}}$
 $\frac{\partial b}{\partial z^{(1)}} = \frac{\partial b}{\partial a^{(1)}} = \frac{\partial b}{\partial x^{(1)}} = \frac{\partial b}{\partial z^{(2)}} = \dots = \frac{\partial b}{\partial z^{(m)}} = \frac{\partial b}{\partial a^{(m)}} = \frac{\partial b}{\partial x^{(m)}}$
 $\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial w_i} = \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial a^{(i)}} = \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial x^{(i)}} = \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial z^{(i)}}$
 $\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial b_i} = \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial a^{(i)}} = \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial x^{(i)}} = \frac{1}{m} \sum_{i=1}^m \frac{\partial J}{\partial z^{(i)}}$

$$\begin{aligned}
 db &= \frac{1}{m} \sum_{i=1}^m dz^{(i)} \\
 &= \frac{1}{m} \text{np.sum}(d\bar{z}) \\
 d\bar{w} &= \frac{1}{m} X d\bar{z}^\top \\
 &= \frac{1}{m} \left[\begin{array}{c|c} x^{(1)} & \vdots \\ \hline 1 & 1 \end{array} \right] \left[\begin{array}{c} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{array} \right] \\
 &= \frac{1}{m} \left[\begin{array}{c} x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)} \end{array} \right]_{n \times 1}
 \end{aligned}$$

An

Implementing Logistic Regression

```

J = 0, dw1 = 0, dw2 = 0, db = 0
for i = 1 to m:
    z(i) = wTx(i) + b
    a(i) = σ(z(i))
    J += -[y(i) log a(i) + (1 - y(i)) log(1 - a(i))]
    dz(i) = a(i) - y(i)
    dw1 += x1(i) dz(i)
    dw2 += x2(i) dz(i)
    db += dz(i)
J = J/m, dw1 = dw1/m, dw2 = dw2/m
db = db/m

```

```

for iter in range(1000):
    z = wTX + b
    = np.dot(w.T, X) + b
    A = σ(z)
    d\bar{z} = A - Y
    d\bar{w} = \frac{1}{m} X d\bar{z}^\top
    db = \frac{1}{m} np.sum(d\bar{z})
    w := w - α d\bar{w}
    b := b - α db

```

End

Broadcasting in python

Broadcasting

- 특정 조건을 만족하지 않는 모양의 다른 배열끼리 연산도 가능하게 해주며, 모양이 부족한 부분은 확장하여 연산을 수행하게 되는,

예시) • $\text{np.arange}(3) + 5$

$$[0, 1, 2] + 5 = [5, 6, 7]$$

• $\text{np.ones}(3, 3) + \text{np.arange}(3)$

$$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} + [0, 1, 2] = \begin{bmatrix} 1, 2, 3 \\ 1, 2, 3 \\ 1, 2, 3 \end{bmatrix}$$

• $\text{np.arange}(3).reshape(3, 1) + \text{np.arange}(3)$

$$\begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} + [0, 1, 2] = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

• 차원의 크기가 1일 때 브로드캐스팅 가능.

- 두 배열 간의 연산에서 최소한 하나의 배열의 차원이 1이어야.

• 차원의 짝이 맞을 때 가능.

- 차원의 짝의 길이가 동일하면 브로드캐스팅 가능.