

Mathematics for Artificial Intelligence

5강: 딥러닝 학습방법 이해하기

임성빈

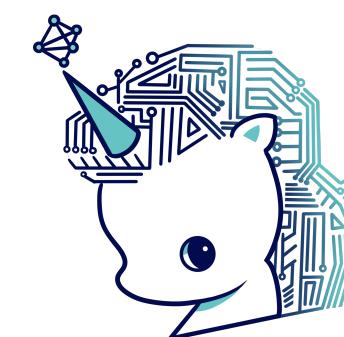


인공지능대학원 & 산업공학과
Learning Intelligent Machine Lab

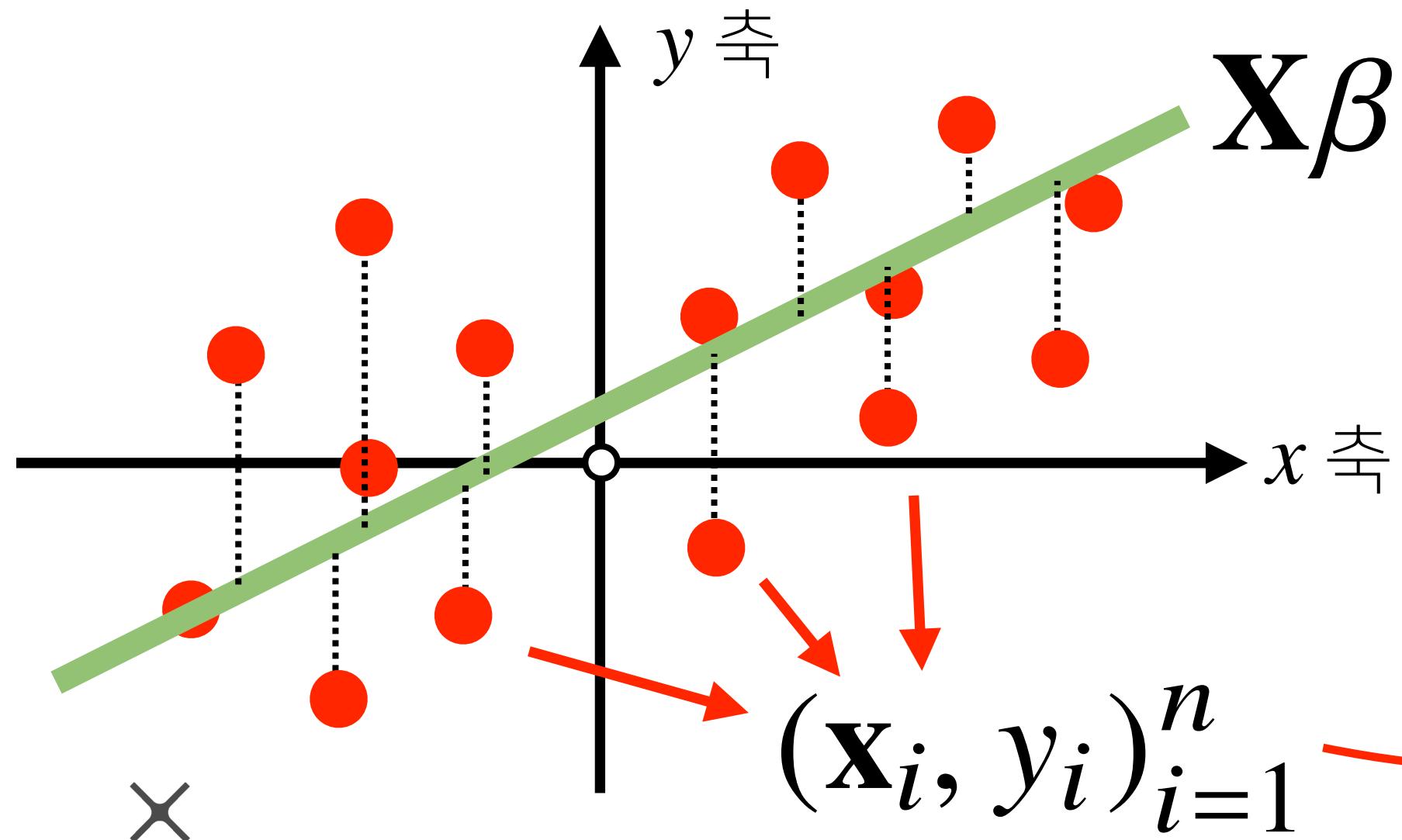


신경망을 수식으로 분해해보자

- 지난 시간까지 데이터를 선형모델로 해석하는 방법을 배웠다면 이제부턴 비선형모델인 신경망(neural network)을 배워보겠습니다



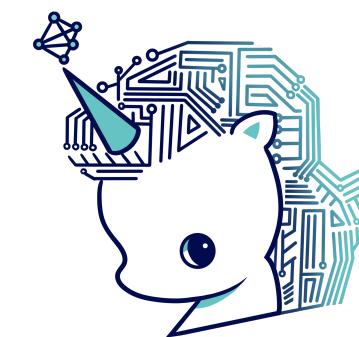
신경망을 수식으로 분해하려면 우선 선형모델을 먼저 이해해야 합니다



$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} \neq \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$
$$\mathbf{X}\beta = \hat{\mathbf{y}} \approx \mathbf{y} \rightarrow \min_{\beta} E \|\mathbf{y} - \hat{\mathbf{y}}\|_2$$

신경망을 수식으로 분해해보자

- 지난 시간까지 데이터를 선형모델로 해석하는 방법을 배웠다면 이제부턴 비선형모델인 신경망(neural network)을 배워보겠습니다



각 행벡터 \mathbf{o}_i 는 데이터 \mathbf{x}_i 와 가중치 행렬 \mathbf{W} 사이의 행렬곱과 절편 \mathbf{b} 벡터의 합으로 표현된다고 가정해봅시다

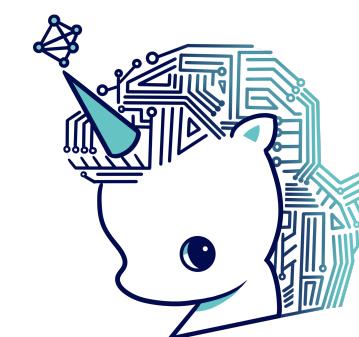
$$\begin{bmatrix} \mathbf{o}_1 \\ \mathbf{o}_2 \\ \vdots \\ \mathbf{o}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dp} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \cdots & b_p \end{bmatrix}$$

\mathbf{O} \mathbf{X} \mathbf{W} \mathbf{b}
 $(n \times p)$ $(n \times d)$ $(d \times p)$ $(n \times p)$

\times

신경망을 수식으로 분해해보자

- 지난 시간까지 데이터를 선형모델로 해석하는 방법을 배웠다면 이제부턴 비선형모델인 신경망(neural network)을 배워보겠습니다



데이터가 바뀌면 결과값도 바뀌게 됩니다. 이 때 출력 벡터의 차원은 d 에서 p 로 바뀌게 됩니다

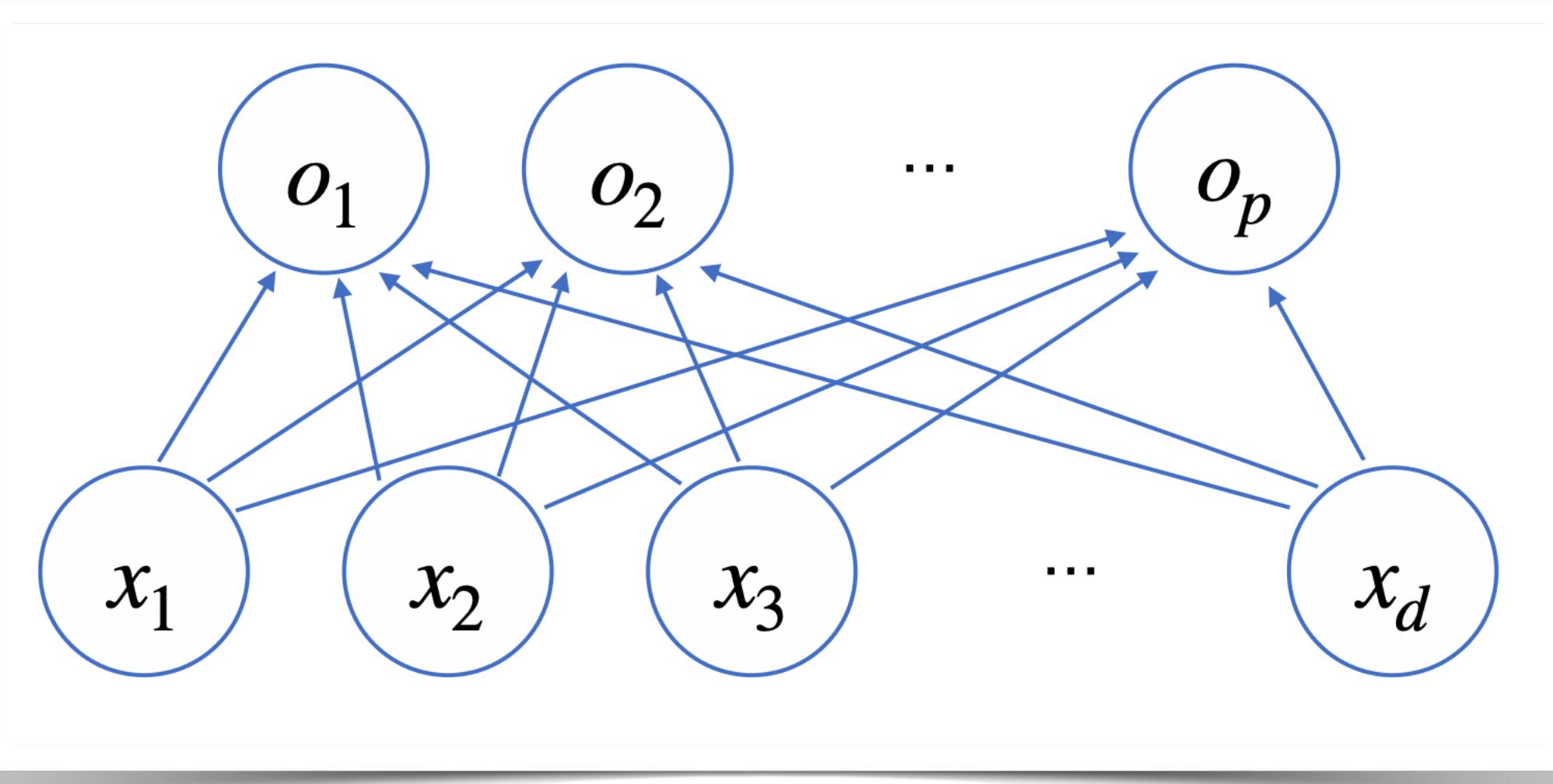
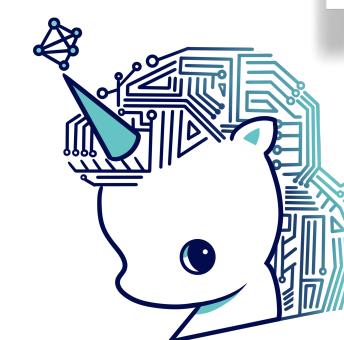
$$\begin{bmatrix} - \mathbf{o}_1 - \\ - \mathbf{o}_2 - \\ \vdots \\ - \mathbf{o}_n - \end{bmatrix} = \begin{bmatrix} - \mathbf{x}_1 - \\ - \mathbf{x}_2 - \\ \vdots \\ - \mathbf{x}_n - \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dp} \end{bmatrix} + \begin{bmatrix} | & | & & \cdots & | \\ b_1 & b_2 & \cdots & b_p \\ | & | & \cdots & | \end{bmatrix}$$

\mathbf{O} \mathbf{X} \mathbf{W} \mathbf{b}
 $(n \times p)$ $(n \times d)$ $(d \times p)$ $(n \times p)$

\times

신경망을 수식으로

- 지난 시간까지 데이터를 선
비선형모델인 신경망(neu)



면 이제부턴

d 개의 변수로 p 개의 선형모델을 만들어서 p 개의 잠재변수를 설명하는 모델을 상상해볼 수 있습니다

$$\begin{bmatrix} - \mathbf{o}_1 - \\ - \mathbf{o}_2 - \\ \vdots \\ - \mathbf{o}_n - \end{bmatrix} = \begin{bmatrix} - \mathbf{x}_1 - \\ - \mathbf{x}_2 - \\ \vdots \\ - \mathbf{x}_n - \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dp} \end{bmatrix} + \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \cdots & b_p \\ | & | & \cdots & | \end{bmatrix}$$

O
 $(n \times p)$

X
 $(n \times d)$

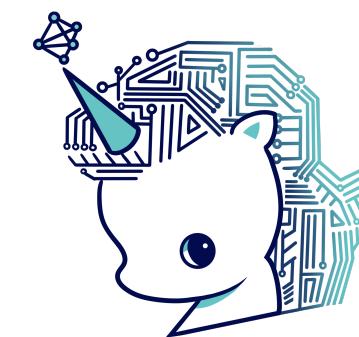
W
 $(d \times p)$

b
 $(n \times p)$

×

신경망을 수식으로 분해해보자

- 지난 시간까지 데이터를 선형모델로 해석하는 방법을 배웠다면 이제부턴 비선형모델인 신경망(neural network)을 배워보겠습니다



출력 벡터 \mathbf{o} 에 softmax 함수를 합성하면 확률벡터가 되므로 특정 클래스 k 에 속할 확률로 해석할 수 있다

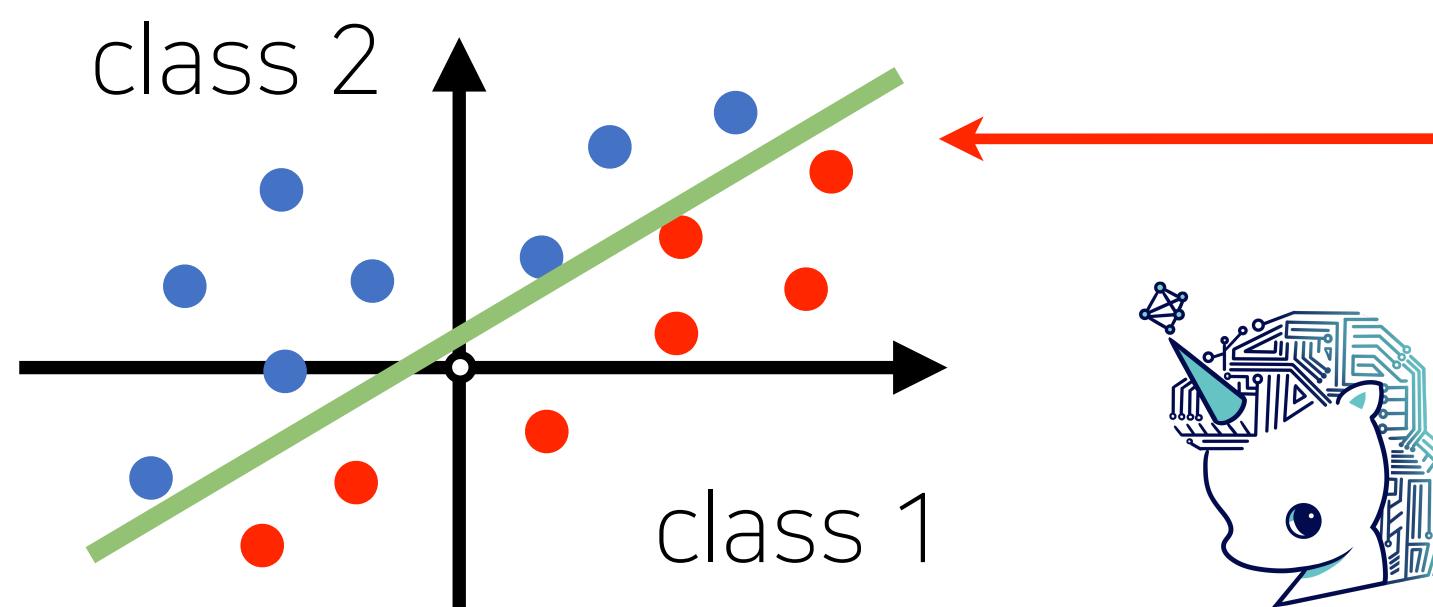
$$\text{softmax}(\mathbf{o}) = \left(\frac{\exp(o_1)}{\sum_{k=1}^p \exp(o_k)}, \dots, \frac{\exp(o_p)}{\sum_{k=1}^p \exp(o_k)} \right)$$

$$\begin{bmatrix} \mathbf{o}_1 \\ \mathbf{o}_2 \\ \vdots \\ \mathbf{o}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dp} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}$$

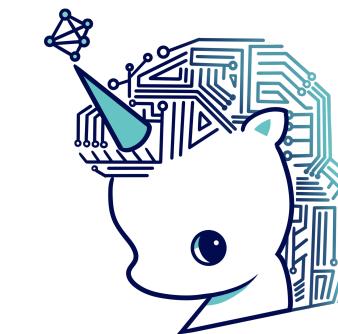
X

소프트맥스 연산

- 소프트맥스(softmax) 함수는 모델의 출력을 확률로 해석할 수 있게 변환해 주는 연산입니다
- 분류 문제를 풀 때 선형모델과 소프트맥스 함수를 결합하여 예측합니다



$$\text{softmax}(\mathbf{o}) = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$



softmax 함수를 통해 \mathbb{R}^p 에 있는 벡터를 확률벡터로 변환할 수 있습니다
(예: [1, 2, 0] → [0.24, 0.67, 0.09])

```
def softmax(vec):  
    denumerator = np.exp(vec - np.max(vec, axis=-1, keepdims=True))  
    numerator = np.sum(denumerator, axis=-1, keepdims=True)  
    val = denumerator / numerator  
    return val
```

```
1 vec = np.array([[1, 2, 0], [-1, 0, 1], [-10, 0, 10]])  
2 softmax(vec)  
  
array([[2.44728471e-01, 6.65240956e-01, 9.00305732e-02],  
       [9.00305732e-02, 2.44728471e-01, 6.65240956e-01],  
       [2.06106005e-09, 4.53978686e-05, 9.99954600e-01]])
```

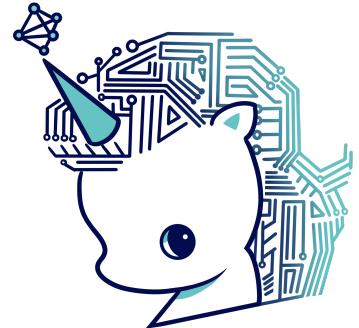
소프트맥스 연산

- 소프트맥스(softmax) 함수는 모델의 출력을 확률로 해석할 수 있게 변환해 주는 연산입니다
- 분류 문제를 풀 때 선형모델과 소프트맥스 함수를 결합하여 예측합니다

```
def one_hot(val, dim):
    return [np.eye(dim)[_] for _ in val]

def one_hot_encoding(vec):
    vec_dim = vec.shape[1]
    vec_argmax = np.argmax(vec, axis=-1)
    return one_hot(vec_argmax, vec_dim)

def softmax(vec):
    denumerator = np.exp(vec - np.max(vec, axis=-1, keepdims=True))
    numerator = np.sum(denumerator, axis=-1, keepdims=True)
    val = denumerator / numerator
    return val
```



one_hot(~~softmax(0)~~) one_hot(0)

그러나 추론을 할 때는 원-핫(one-hot) 벡터로 최대값을 가진 주소만 1로 출력하는 연산을 사용해서 softmax 를 사용하지 않습니다

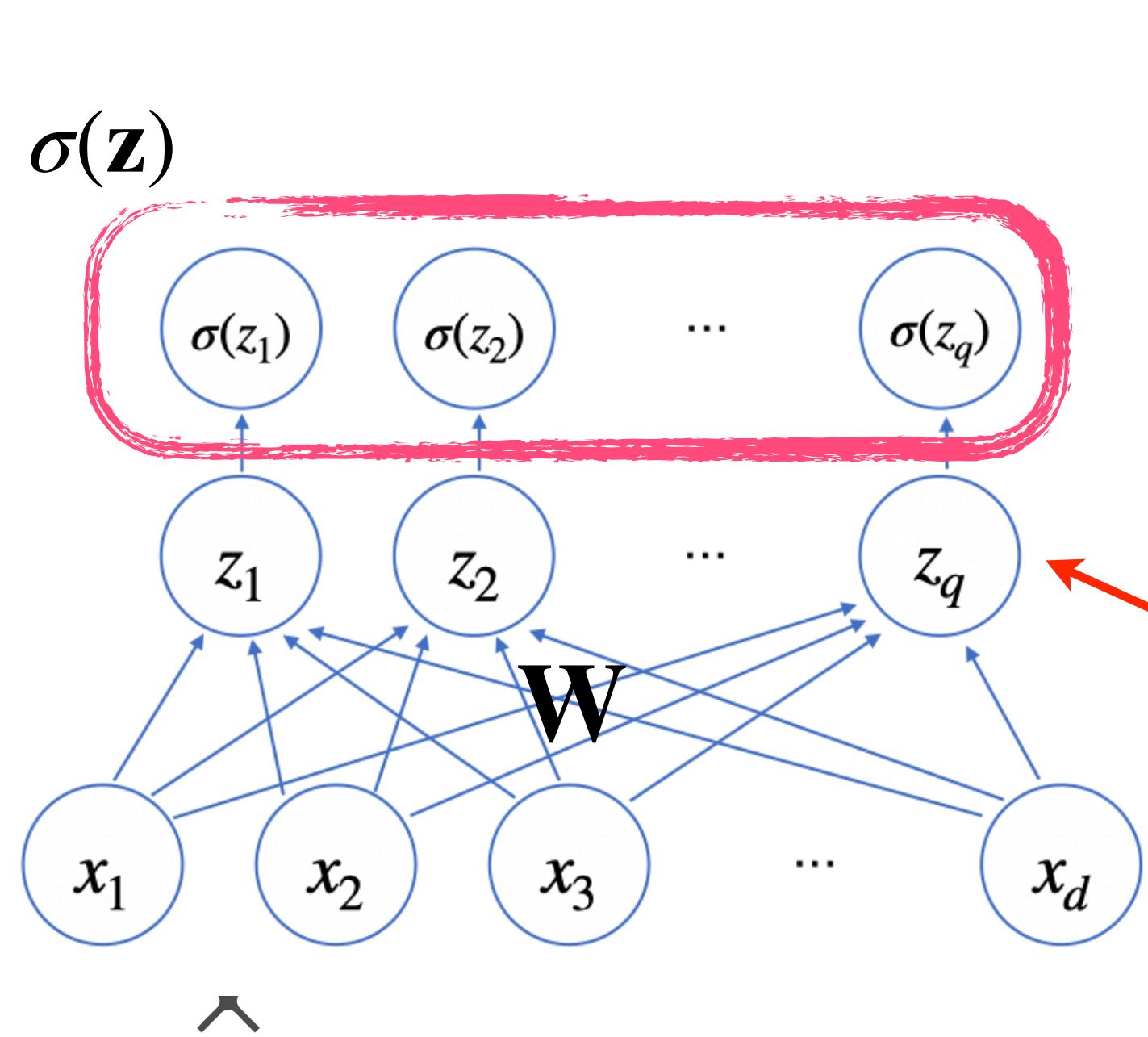
```
1 vec = np.array([[1, 2, 0], [-1, 0, 1], [-10, 0, 10]])
2 print(one_hot_encoding(vec))
3 print(one_hot_encoding(softmax(vec)))

[array([0., 1., 0.]), array([0., 0., 1.]), array([0., 0., 1.])]
[array([0., 1., 0.]), array([0., 0., 1.]), array([0., 0., 1.])]
```

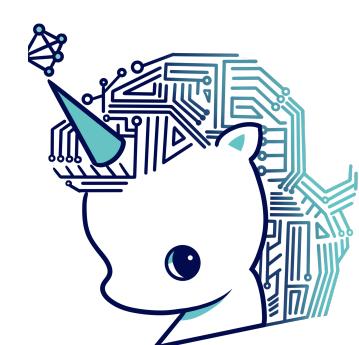
X

신경망을 수식으로 분해해보자

- 신경망은 선형모델과 활성함수(activation function)를 합성한 함수입니다



$$\mathbf{H} = (\sigma(\mathbf{z}_1), \dots, \sigma(\mathbf{z}_n)) \quad \sigma(\mathbf{z}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

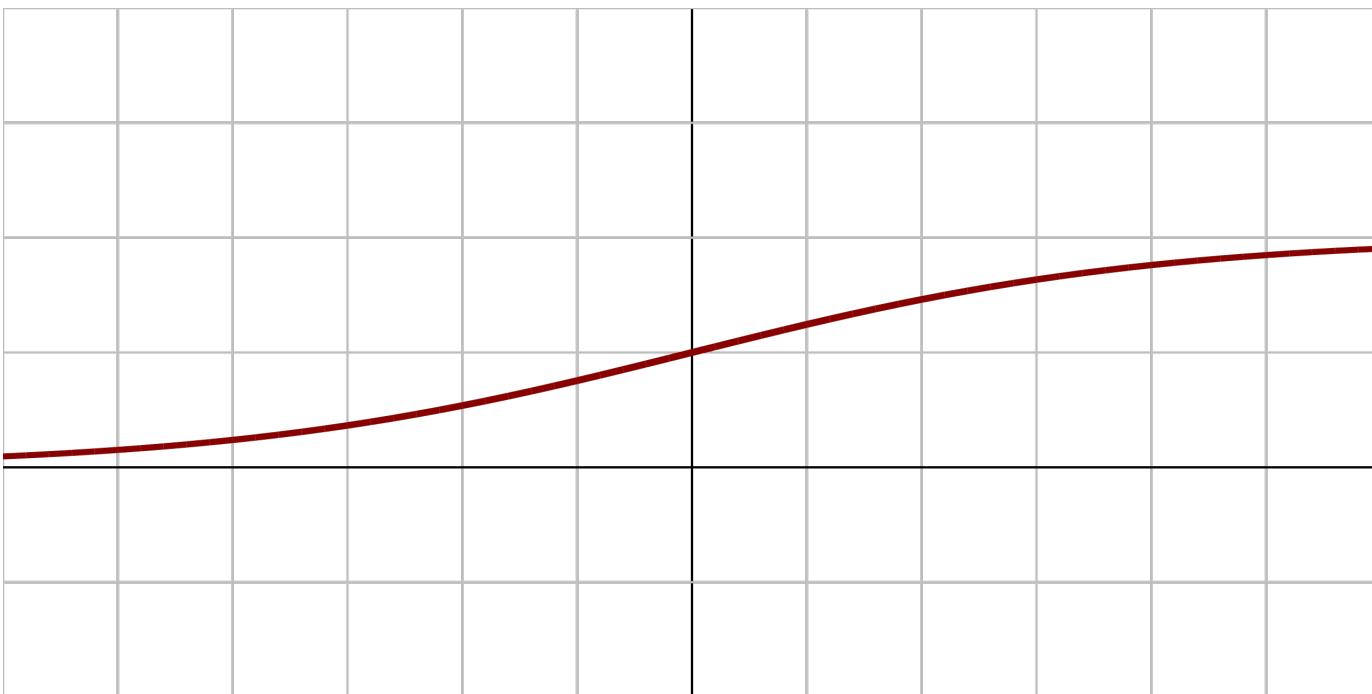


활성함수 σ 는 비선형함수로 잠재벡터 $\mathbf{z} = (z_1, \dots, z_q)$ 의 각 노드에 개별적으로 적용하여 새로운 잠재벡터 $\mathbf{H} = (\sigma(\mathbf{z}_1), \dots, \sigma(\mathbf{z}_n))$ 를 만든다

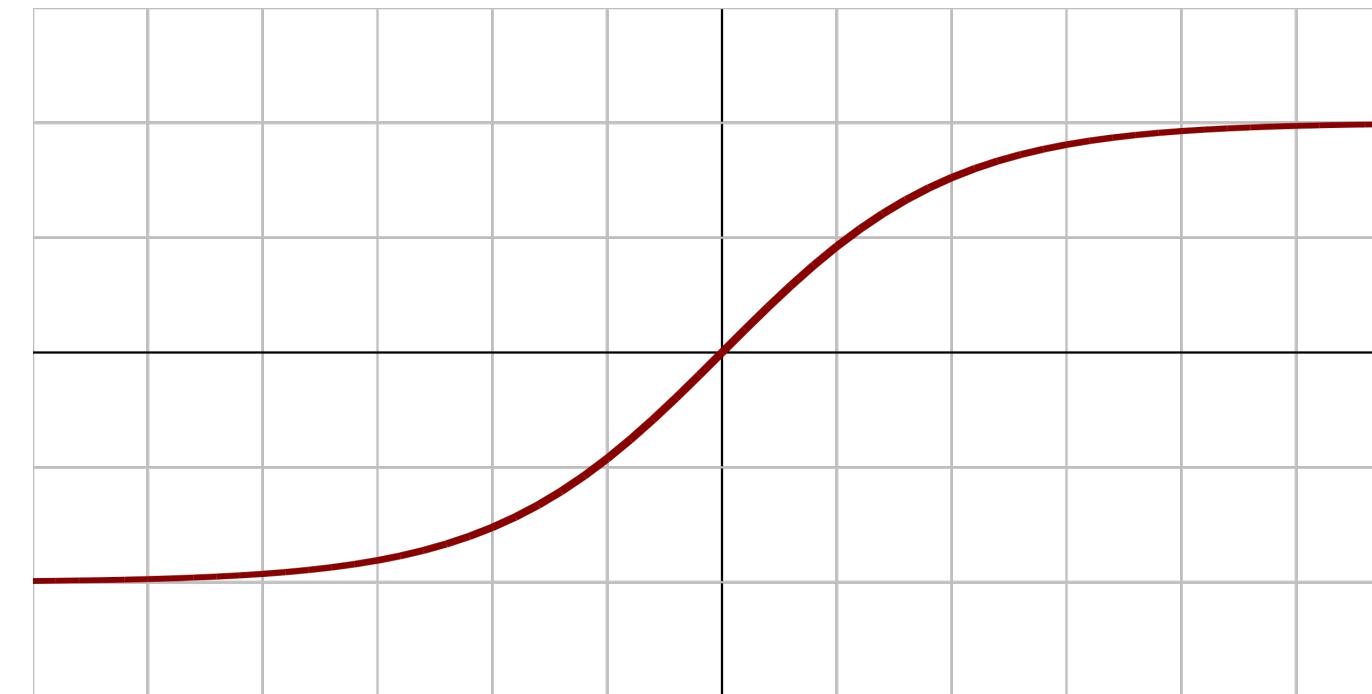
$$\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dp} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}$$

활성함수가 뭐에요?

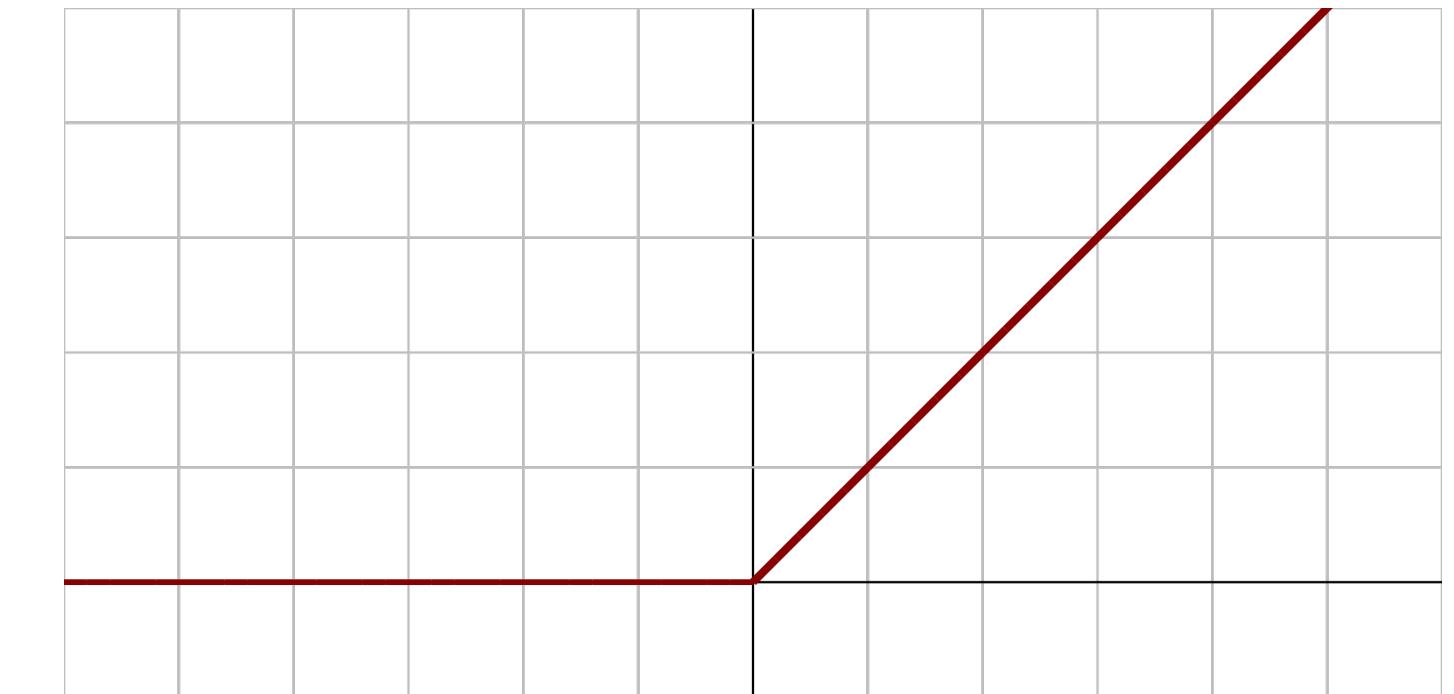
- 활성함수(activation function)는 \mathbb{R} 위에 정의된 비선형(nonlinear) 함수로서 딥러닝에서 매우 중요한 개념입니다
- 활성함수를 쓰지 않으면 딥러닝은 선형모형과 차이가 없습니다
- 시그모이드(sigmoid) 함수나 \tanh 함수는 전통적으로 많이 쓰이던 활성함수지만 딥러닝에선 ReLU 함수를 많이 쓰고 있다



$$\times \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$



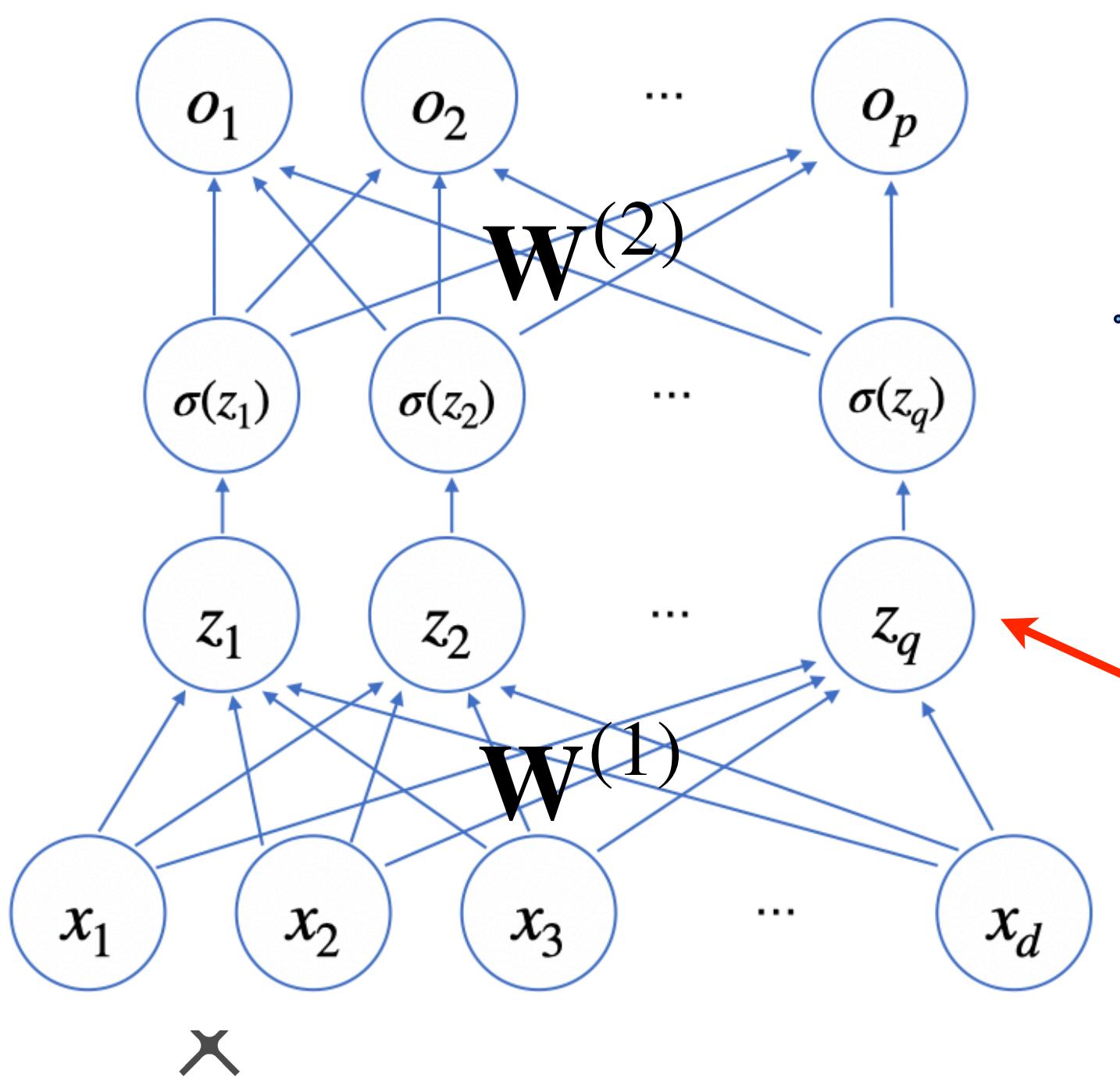
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



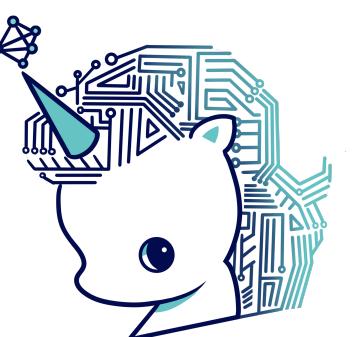
$$\text{ReLU}(x) = \max\{0, x\}$$

신경망을 수식으로 분해해보자

- 신경망은 선형모델과 활성함수(activation function)를 합성한 함수입니다



$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$



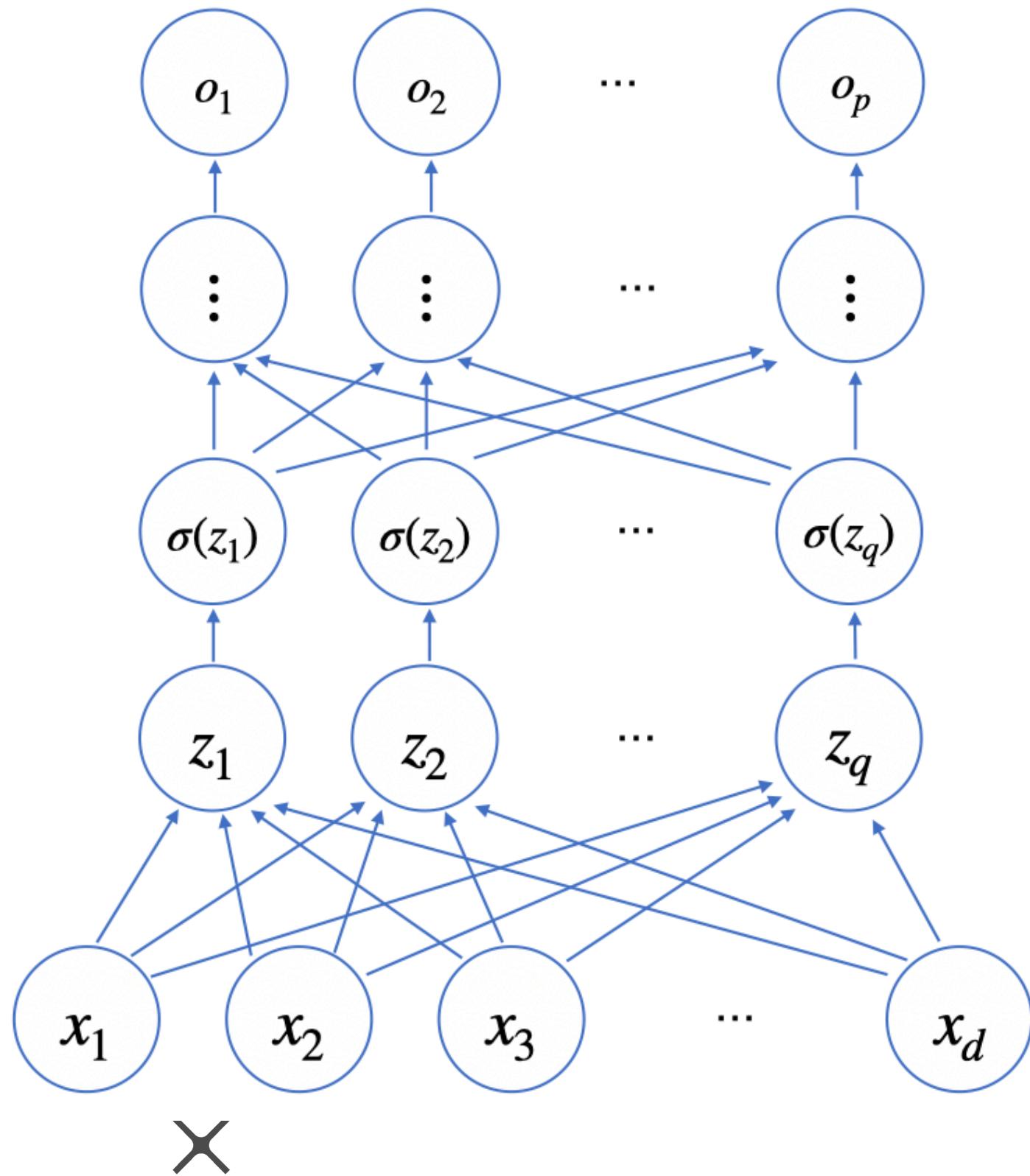
$$\mathbf{H} = (\sigma(\mathbf{z}_1), \dots, \sigma(\mathbf{z}_n))$$
$$\sigma(\mathbf{z}) = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

잠재벡터 \mathbf{H} 에서 가중치 행렬 $\mathbf{W}^{(2)}$ 와 $\mathbf{b}^{(2)}$ 를 통해 다시 한 번 선형변환해서 출력하게 되면 $(\mathbf{W}^{(2)}, \mathbf{W}^{(1)})$ 를 패러미터로 가진 2층(2-layers) 신경망이다

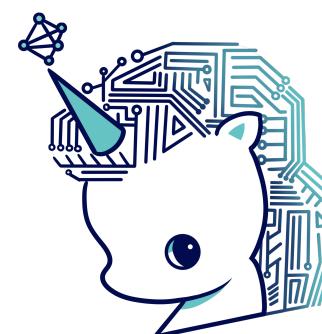
$$\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dp} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}$$

신경망을 수식으로 분해해보자

- 신경망은 선형모델과 활성함수(activation function)를 합성한 함수입니다
- 다층(multi-layer) 퍼셉트론(MLP)은 신경망이 여러층 합성된 함수입니다



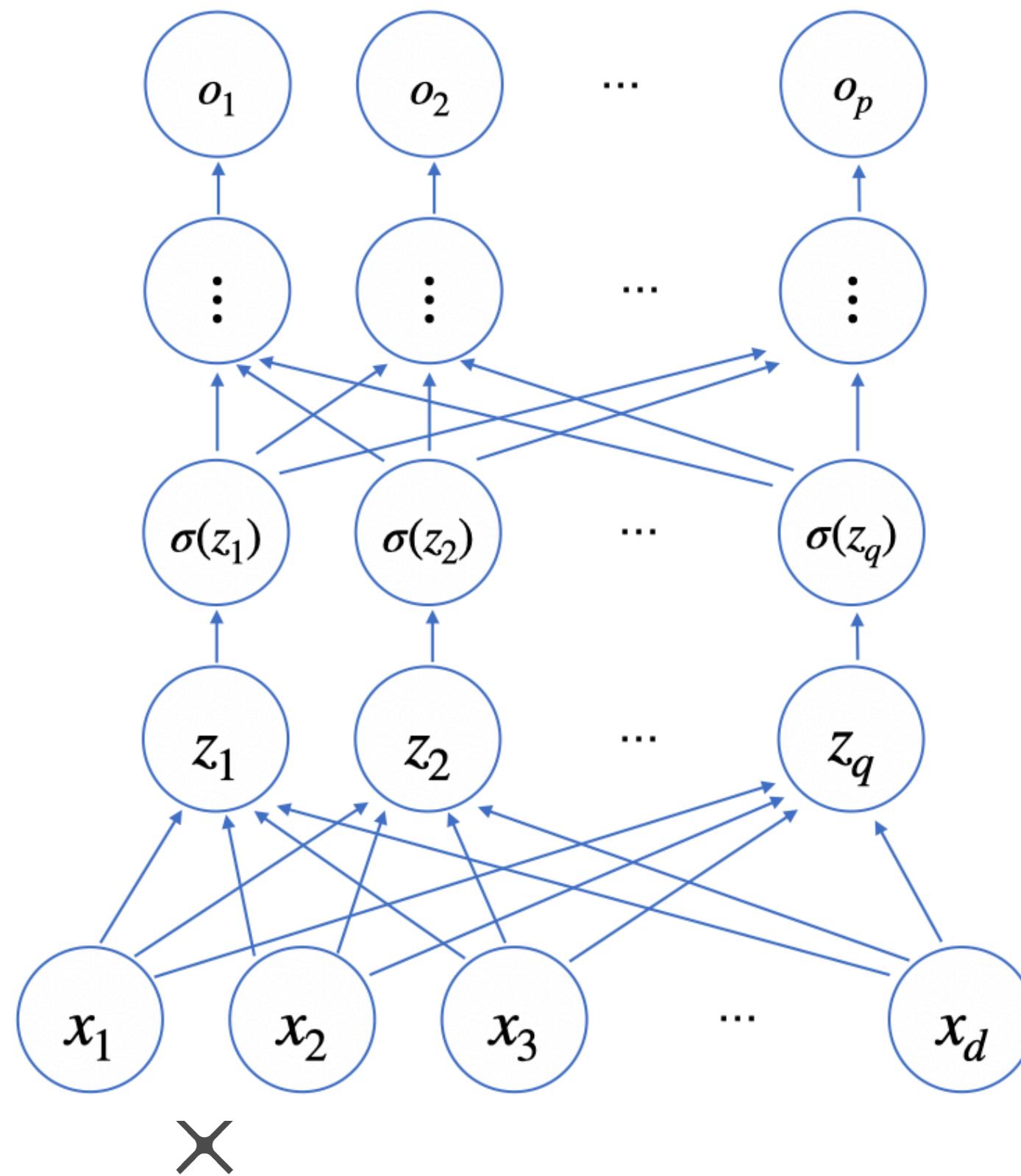
$$\begin{aligned} \mathbf{H}^{(1)} &= \sigma(\mathbf{Z}^{(1)}) \\ \mathbf{Z}^{(1)} &= \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)} \end{aligned}$$



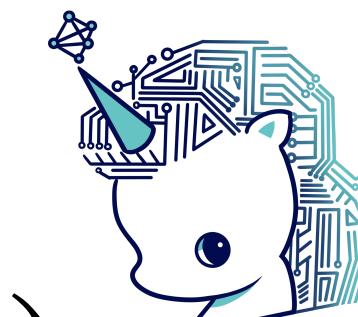
$\sigma(\mathbf{Z})$ 는 $(\sigma(\mathbf{z}_1), \dots, \sigma(\mathbf{z}_n))$ 으로
이루어진 행렬입니다

신경망을 수식으로 분해해보자

- 신경망은 선형모델과 활성함수(activation function)를 합성한 함수입니다
- 다층(multi-layer) 퍼셉트론(MLP)은 신경망이 여러층 합성된 함수입니다



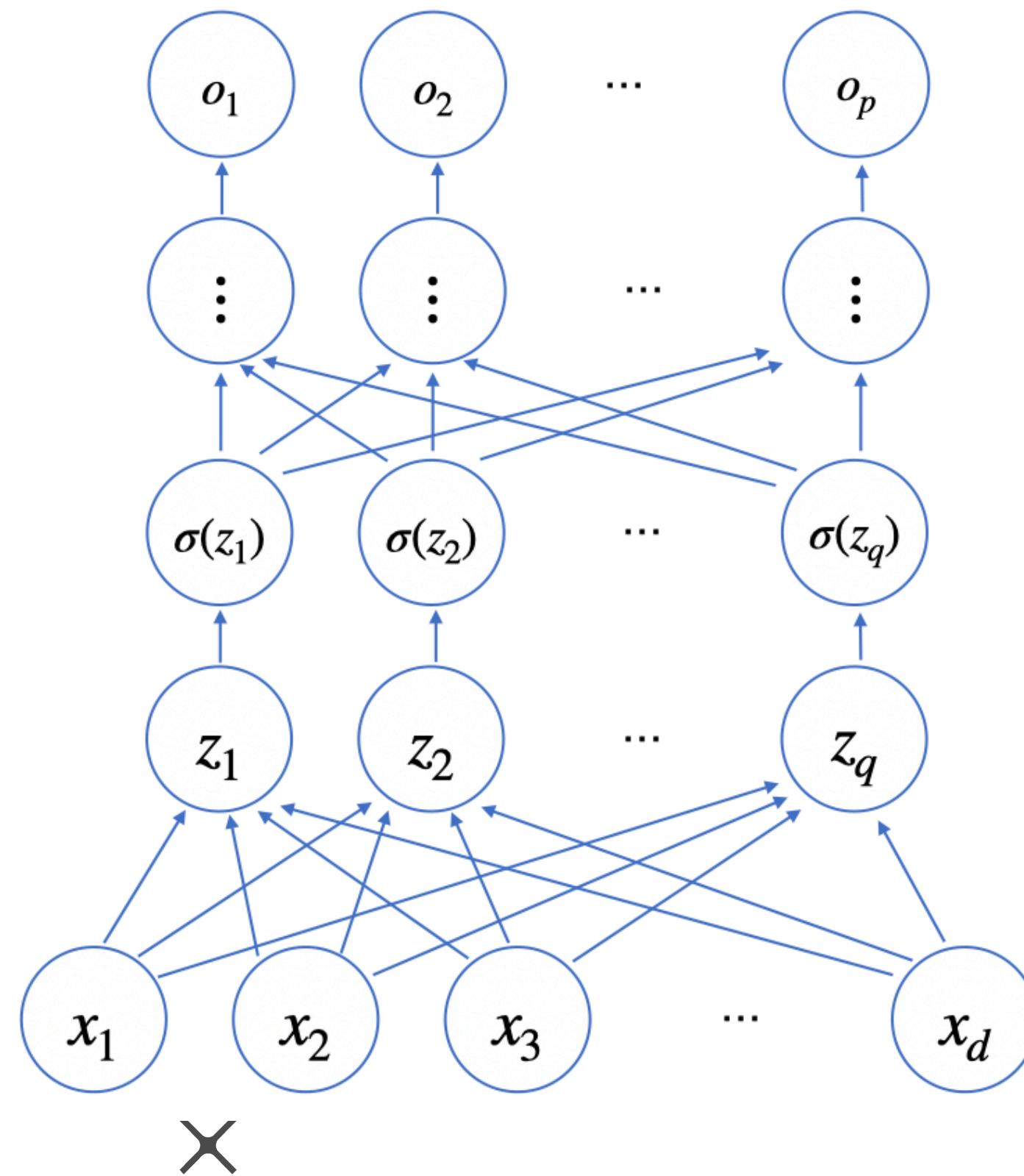
$$\begin{aligned} \mathbf{H}^{(\ell)} &= \sigma(\mathbf{Z}^{(\ell)}) \\ \mathbf{Z}^{(\ell)} &= \mathbf{H}^{(\ell-1)} \mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)} \\ \mathbf{H}^{(1)} &= \sigma(\mathbf{Z}^{(1)}) \\ \mathbf{Z}^{(1)} &= \mathbf{X} \mathbf{W}^{(1)} + \mathbf{b}^{(1)} \end{aligned}$$



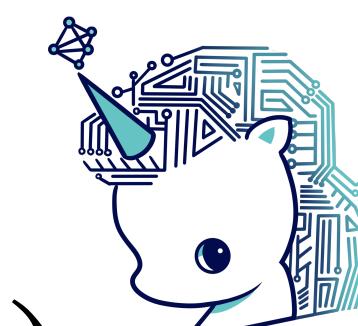
MLP의 패러미터는 L 개의 가중치 행렬 $\mathbf{W}^{(L)}, \dots, \mathbf{W}^{(1)}$ 과로 이루어져 있다

신경망을 수식으로 분해해보자

- 신경망은 선형모델과 활성함수(activation function)를 합성한 함수입니다
- 다층(multi-layer) 퍼셉트론(MLP)은 신경망이 여러층 합성된 함수입니다



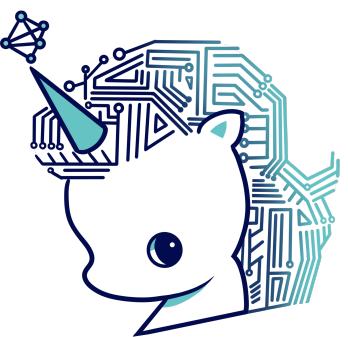
$$\begin{aligned}\mathbf{O} &= \mathbf{Z}^{(L)} \\ \mathbf{H}^{(\ell)} &= \sigma(\mathbf{Z}^{(\ell)}) \\ \mathbf{Z}^{(\ell)} &= \mathbf{H}^{(\ell-1)} \mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)} \\ \vdots \\ \mathbf{H}^{(1)} &= \sigma(\mathbf{Z}^{(1)}) \\ \mathbf{Z}^{(1)} &= \mathbf{X} \mathbf{W}^{(1)} + \mathbf{b}^{(1)}\end{aligned}$$



$\ell = 1, \dots, L$ 까지 순차적인 신경망 계산을
순전파(forward propagation)라 부른다

왜 층을 여러개를 쌓나요?

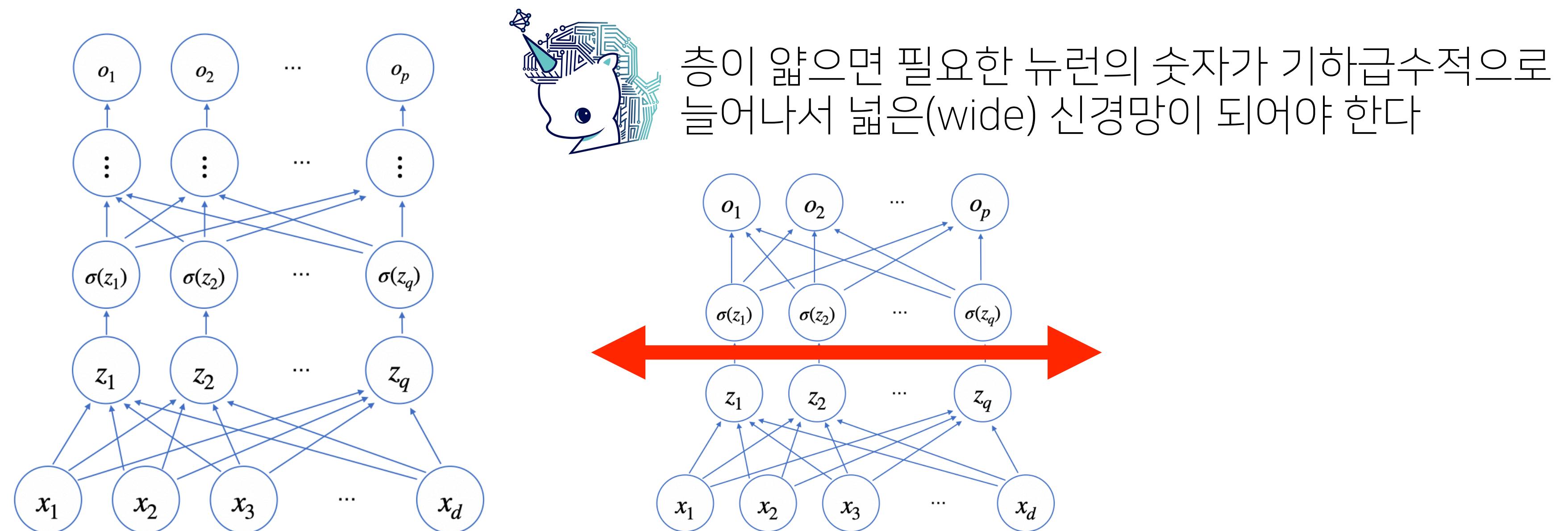
- 이론적으로는 2층 신경망으로도 임의의 연속함수를 근사할 수 있습니다



이를 universal approximation theorem 이라 부릅니다

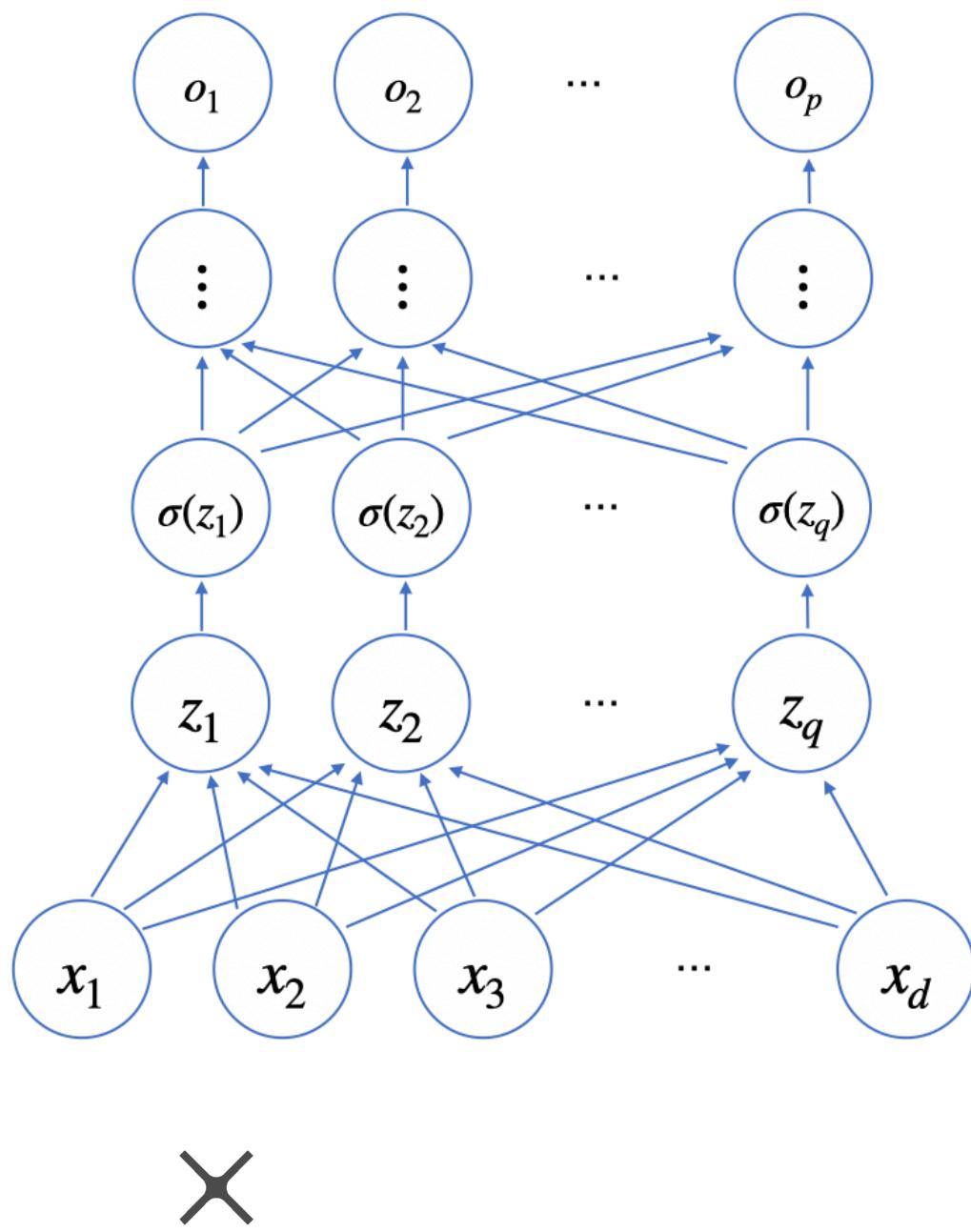
왜 층을 여러개를 쌓나요?

- 이론적으로는 2층 신경망으로도 임의의 연속함수를 근사할 수 있습니다
- 그러나 층이 깊을수록 목적함수를 근사하는데 필요한 뉴런(노드)의 숫자가 훨씬 빨리 줄어들어 좀 더 효율적으로 학습이 가능합니다

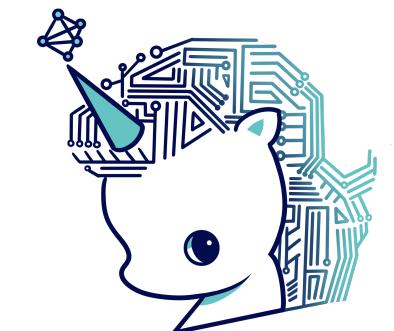


딥러닝 학습원리: 역전파 알고리즘

- 딥러닝은 역전파(backpropagation) 알고리즘을 이용하여 각 층에 사용된 패러미터 $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^L$ 를 학습합니다



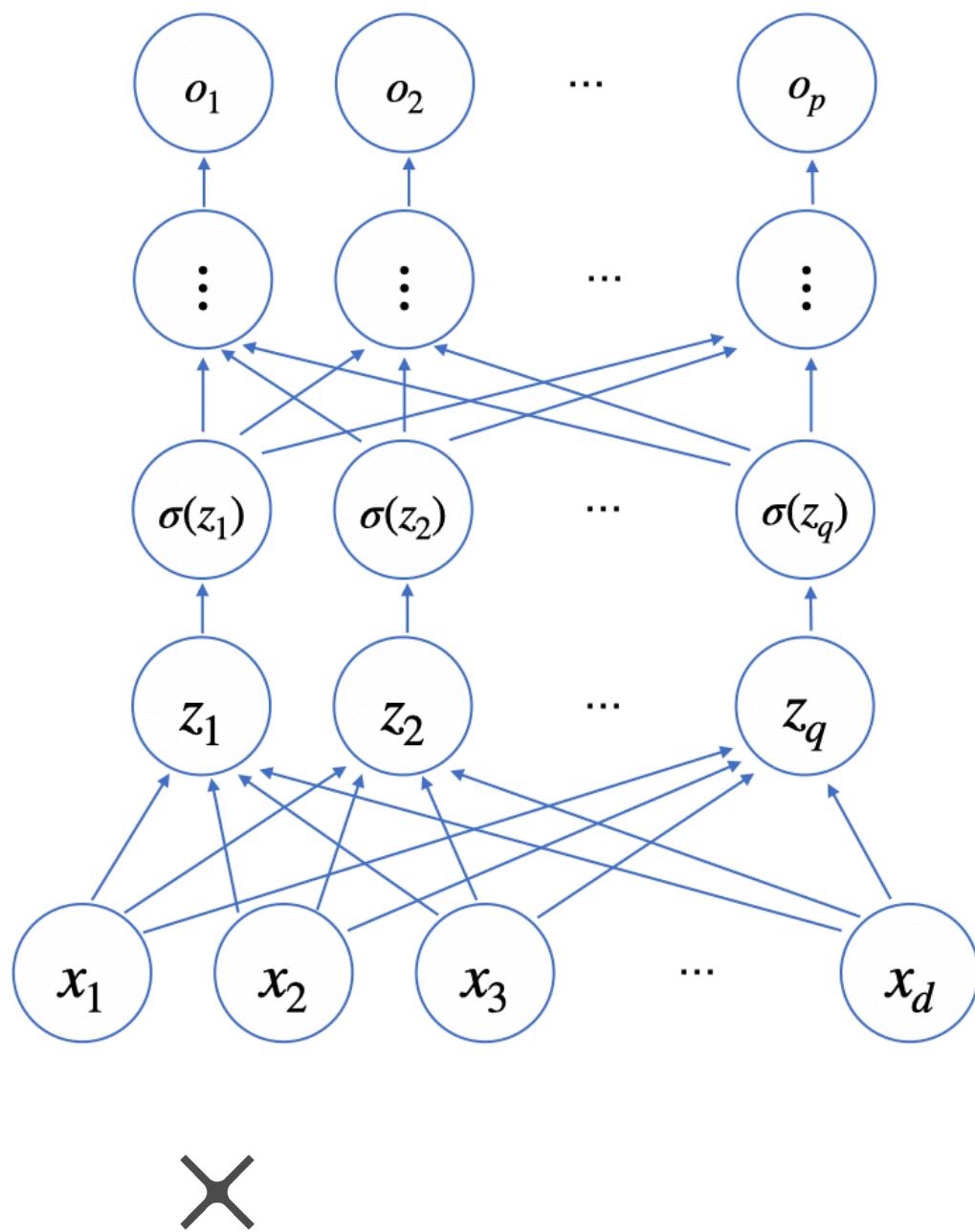
$$\begin{aligned}\mathbf{O} &= \mathbf{Z}^{(L)} \\ \vdots \\ \mathbf{H}^{(\ell)} &= \sigma(\mathbf{Z}^{(\ell)}) \\ \mathbf{Z}^{(\ell)} &= \mathbf{H}^{(\ell-1)} \mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)} \\ \vdots \\ \mathbf{H}^{(1)} &= \sigma(\mathbf{Z}^{(1)}) \\ \mathbf{Z}^{(1)} &= \mathbf{X} \mathbf{W}^{(1)} + \mathbf{b}^{(1)}\end{aligned}$$



손실함수를 \mathcal{L} 이라 했을 때 역전파는 $\partial \mathcal{L} / \partial \mathbf{W}^{(\ell)}$ 정보를 계산할 때 사용된다

딥러닝 학습원리: 역전파 알고리즘

- 딥러닝은 역전파(backpropagation) 알고리즘을 이용하여 각 층에 사용된 패러미터 $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^L$ 를 학습합니다
- 각 층 패러미터의 그레디언트 벡터는 윗층부터 역순으로 계산하게 됩니다



$$\begin{aligned}\mathbf{O} &= \mathbf{Z}^{(L)} \\ \mathbf{H}^{(\ell)} &= \sigma(\mathbf{Z}^{(\ell)}) \\ \mathbf{Z}^{(\ell)} &= \mathbf{H}^{(\ell-1)} \mathbf{W}^{(\ell)} + \mathbf{b}^{(\ell)} \\ \mathbf{H}^{(1)} &= \sigma(\mathbf{Z}^{(1)}) \\ \mathbf{Z}^{(1)} &= \mathbf{X} \mathbf{W}^{(1)} + \mathbf{b}^{(1)}\end{aligned}$$

× $\frac{\partial \mathbf{H}^{(\ell)}}{\partial \mathbf{Z}^{(\ell)}}$ $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{O}} \times \cdots \times \frac{\partial \mathbf{Z}^{(\ell)}}{\partial \mathbf{W}^{(\ell)}}$

× $\frac{\partial \mathbf{H}^{(1)}}{\partial \mathbf{Z}^{(1)}}$ $\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{O}} \times \cdots \times \frac{\partial \mathbf{Z}^{(\ell)}}{\partial \mathbf{b}^{(\ell)}}$

역전파는 $\ell = L, \dots, 1$ 순서로 연쇄법칙을 통해 그레디언트 벡터를 전달한다

역전파 알고리즘 원리 이해하기

- 역전파 알고리즘은 합성함수 미분법인 **연쇄법칙(chain-rule)** 기반 자동미분 (auto-differentiation)을 사용합니다

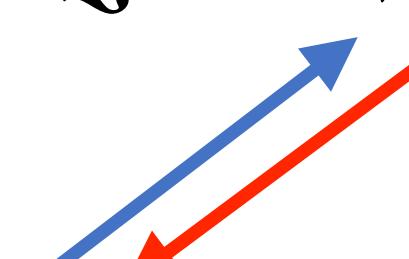
$$z = (x + y)^2$$

$$\frac{\partial z}{\partial x} = ?$$

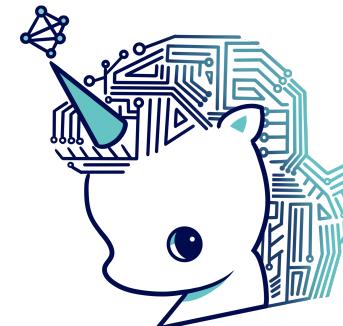
역전파 알고리즘 원리 이해하기

- 역전파 알고리즘은 합성함수 미분법인 **연쇄법칙(chain-rule)** 기반 자동미분 (auto-differentiation)을 사용합니다

$$z = (x + y)^2$$

$$\begin{array}{c} z = w^2 \\ w = x + y \end{array}$$


$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial w} \frac{\partial w}{\partial x}$$



연쇄법칙을 통해 합성함수의 미분을 계산할 수 있다

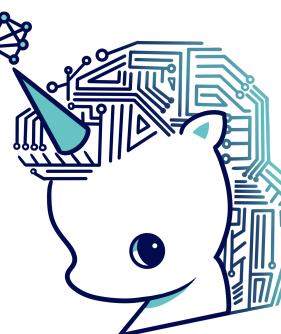
역전파 알고리즘 원리 이해하기

- 역전파 알고리즘은 합성함수 미분법인 **연쇄법칙(chain-rule)** 기반 자동미분 (auto-differentiation)을 사용합니다

$$z = (x + y)^2$$

$$\begin{aligned} z &= w^2 & \longrightarrow & \frac{\partial z}{\partial w} = 2w \\ w &= x + y & \longrightarrow & \frac{\partial w}{\partial x} = 1 \quad \frac{\partial w}{\partial y} = 1 \end{aligned}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial w} \frac{\partial w}{\partial x} = 2w \cdot 1 = 2(x + y)$$

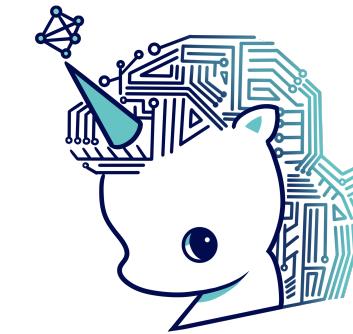


각 노드의 텐서 값을 컴퓨터가 기억해야 미분 계산이 가능하다

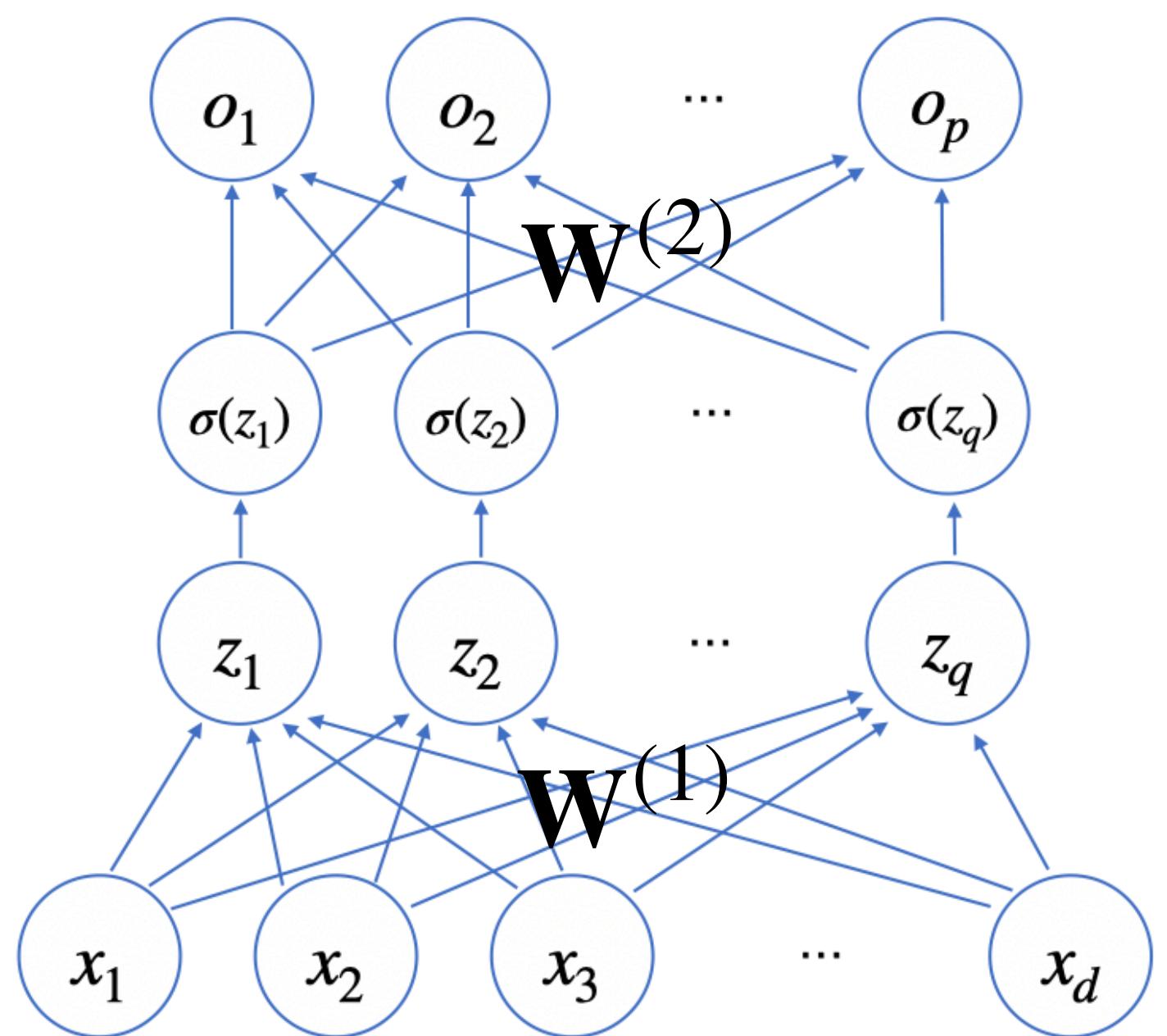
예제: 2층 신경망

- 2층 신경망의 역전파 알고리즘

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}} = ?$$



$\mathbf{W}^{(1)}$ 은 행렬이므로 각 성분에 대한 편미분을 구해야 한다



$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

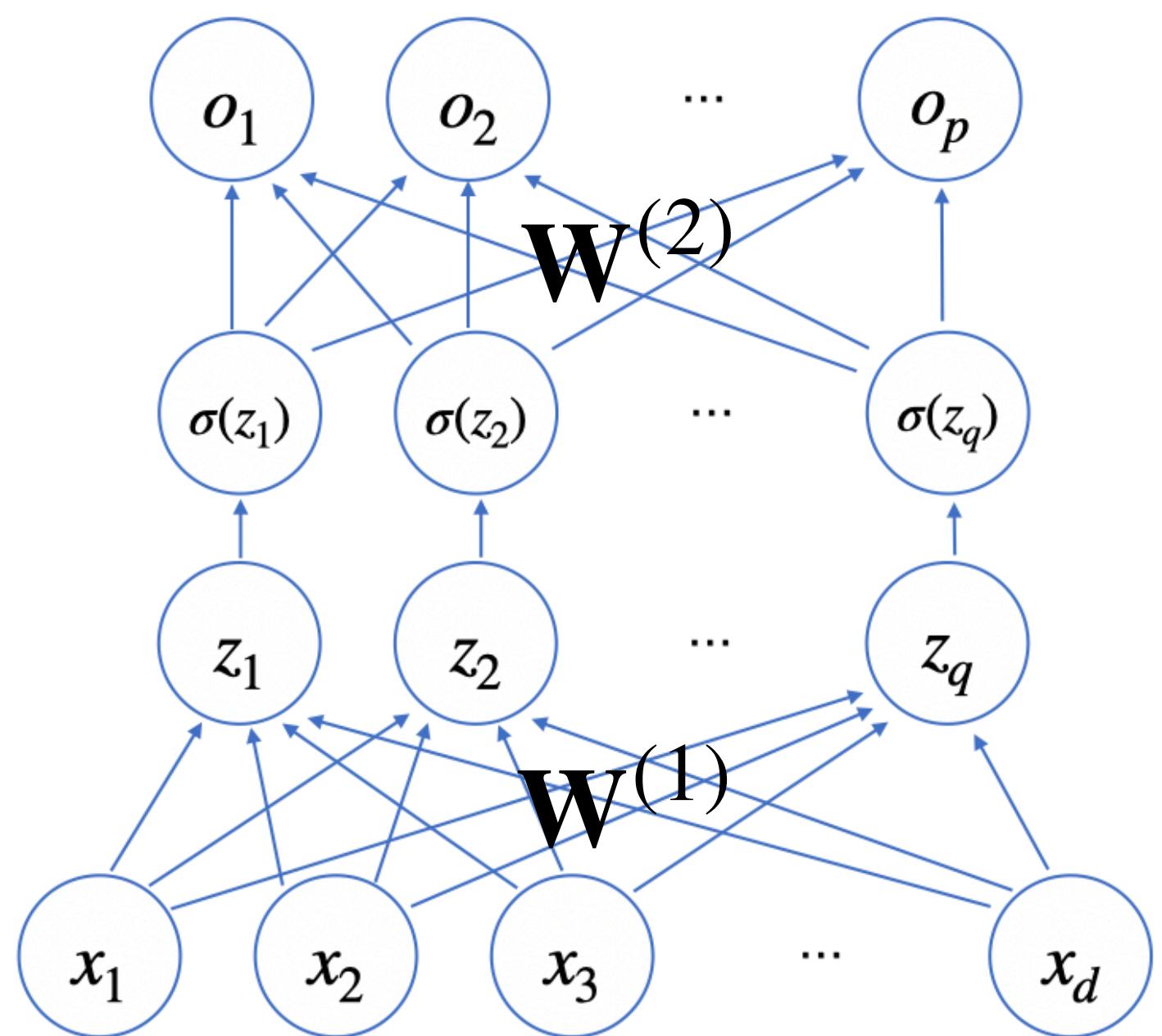
$$\mathbf{h} = \sigma(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

X

예제: 2층 신경망

$$\nabla_{\mathbf{W}^{(1)}} \mathcal{L} = (\nabla_{\mathbf{W}^{(1)}} \mathbf{z})(\nabla_{\mathbf{z}} \mathbf{h})(\nabla_{\mathbf{h}} \mathbf{o})(\nabla_{\mathbf{o}} \mathcal{L}) \longleftrightarrow \frac{\partial \mathcal{L}}{\partial W_{ij}^{(1)}} = \sum_{l,r,k} \frac{\partial \mathcal{L}}{\partial o_l} \frac{\partial o_l}{\partial h_r} \frac{\partial h_r}{\partial z_k} \frac{\partial z_k}{\partial W_{ij}^{(1)}}$$



$$\begin{aligned} \mathbf{o} &= \mathbf{W}^{(2)} \mathbf{h} + \mathbf{b}^{(2)} \\ \mathbf{h} &= \sigma(\mathbf{z}) \\ \mathbf{z} &= \mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \end{aligned}$$

$$\frac{\partial o_l}{\partial h_r} = W_{rl}^{(2)}$$

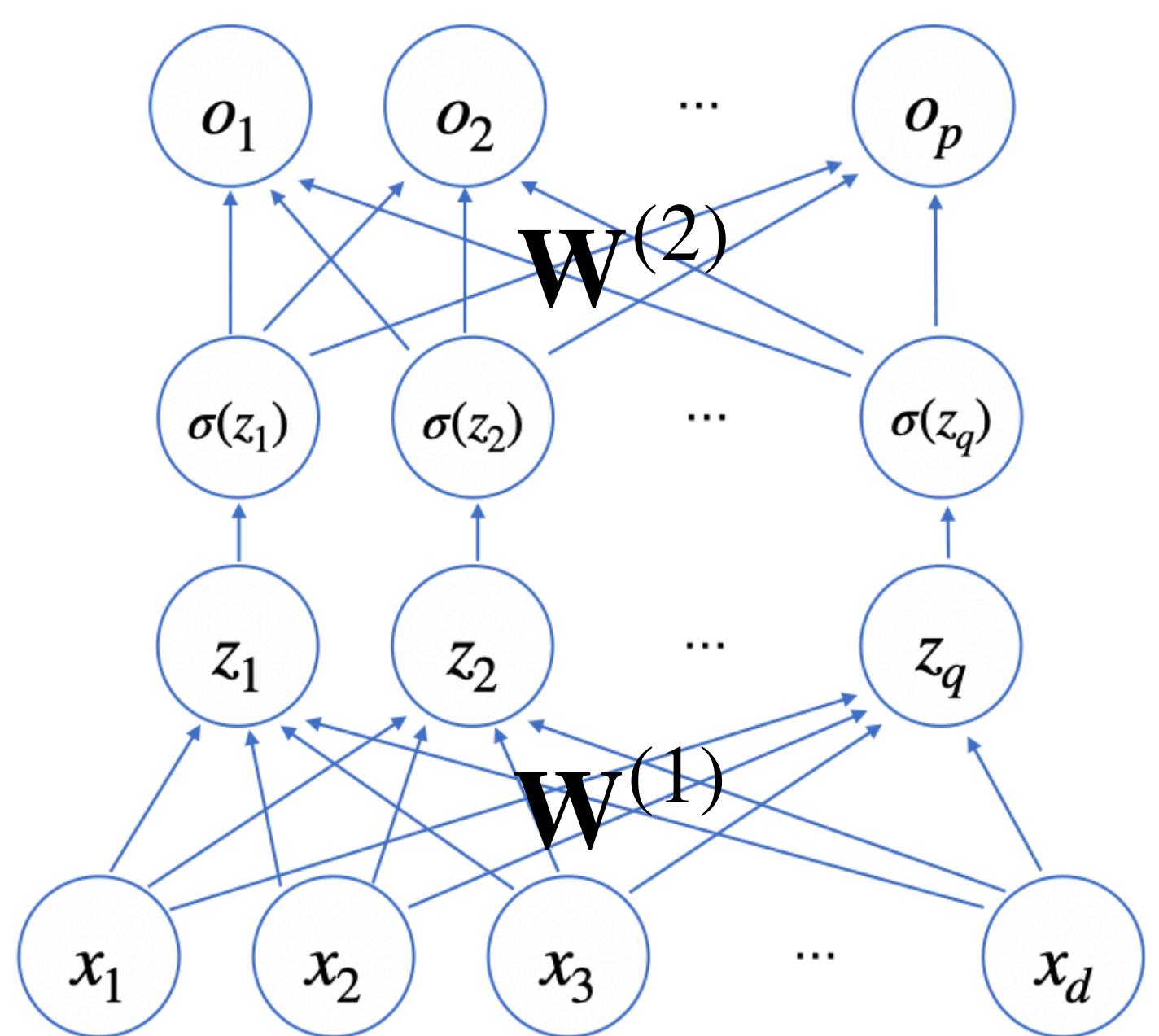
$$\frac{\partial h_r}{\partial z_k} = \sigma'(z_k) \delta_{rk}$$

$$\frac{\partial z_k}{\partial W_{ij}^{(1)}} = \frac{\partial}{\partial W_{ij}^{(1)}} \sum_{i'} x_{i'} W_{ik}^{(1)} = x_i \delta_{jk}$$

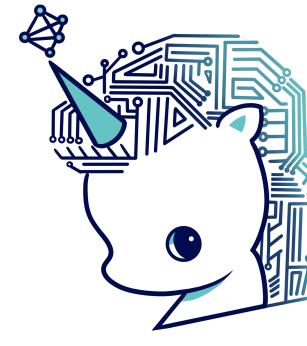
\times

예제: 2층 신경망

$$\nabla_{\mathbf{W}^{(1)}} \mathcal{L} = (\nabla_{\mathbf{W}^{(1)}} \mathbf{z})(\nabla_{\mathbf{z}} \mathbf{h})(\nabla_{\mathbf{h}} \mathbf{o})(\nabla_{\mathbf{o}} \mathcal{L}) \longleftrightarrow \frac{\partial \mathcal{L}}{\partial W_{ij}^{(1)}} = \sum_l \frac{\partial \mathcal{L}}{\partial o_l} W_{jl}^{(2)} \sigma'(z_j) x_i$$



$$\begin{aligned} \mathbf{o} &= \mathbf{W}^{(2)} \mathbf{h} + \mathbf{b}^{(2)} \\ \mathbf{h} &= \sigma(\mathbf{z}) \\ \mathbf{z} &= \mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \end{aligned}$$

$\frac{\partial o_l}{\partial h_r} = W_{rl}^{(2)}$  $\delta_{rk}\delta_{jk}$ 때문에
 $r = k = j$ 만 남는다

$$\frac{\partial h_r}{\partial z_k} = \sigma'(z_k)\delta_{rk}$$

$$\frac{\partial z_k}{\partial W_{ij}^{(1)}} = \frac{\partial}{\partial W_{ij}^{(1)}} \sum_{i'} x_{i'} W_{ik}^{(1)} = x_i \delta_{jk}$$

THE END

다음 시간에 보아요!

Layer Selection ↵ MDP

