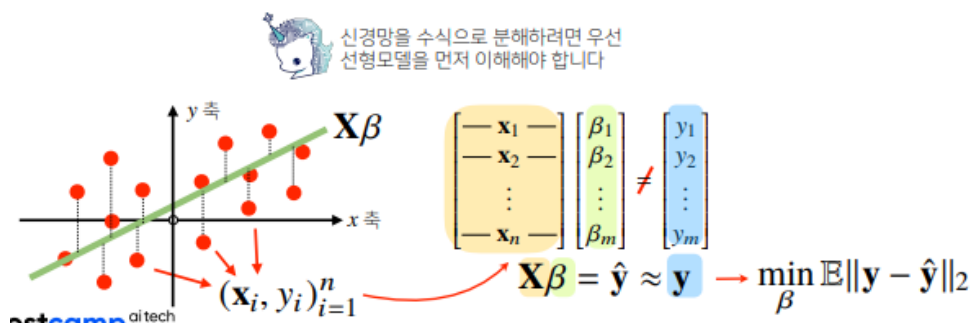


이전까지는 데이터를 선형 모델로 해석하는 방법을 배웠다. 그리고 선형 모델을 해석한 방법을 경사하강법을 이용해 학습하는 방법까지 배웠다. 이때 사용되는 목적식은 주어진 데이터의 정답에 해당하는 y 와 선형 모델의 결과인 y 의 차이의 L2-norm의 기대값을 최소화하는 베타를 찾는 것으로 학습했다. 하지만 이러한 선형 모델은 단순한 데이터를 해석할 때는 도움이 되지만 분류 문제를 풀거나 복잡한 문제를 풀 때는 선형 모델만 가지고 좋은 모델을 만들기는 힘들다. 그래서 **신경망(neural network)**라는 비선형 모델을 배워 볼 것이다.



선형 모델은 x 라는 데이터들을 w 라는 가중치 행렬과 b 라는 절편 벡터의 합으로 o 라는 출력을 내는 것이 기본적인 선형 모델의 수식이다. 절편 벡터 b 는 절편에 해당하는 벡터를 모든 행에 복제 한 것으로 각 행이 모두 같다.

각 행벡터 o_i 는 데이터 x_i 와 가중치 행렬 W 사이의 행렬곱과 절편 b 벡터의 합으로 표현된다고 가정해봅시다

$$\begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dp} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \cdots & b_p \end{bmatrix}$$

$O \quad X \quad W \quad b$
($n \times p$) ($n \times d$) ($d \times p$) ($n \times p$)

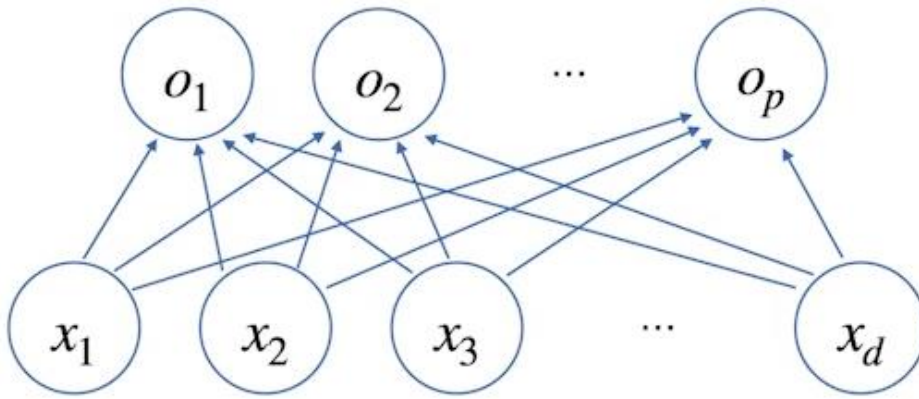
이때 d 개의 차원으로 이루어진 n 개의 데이터는 p 개의 차원으로 이루어진 n 개의 출력값을 반환한다.

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

$$\begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_n \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dp} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \cdots & b_p \end{bmatrix}$$

$O \quad X \quad W \quad b$
($n \times p$) ($n \times d$) ($d \times p$) ($n \times p$)

도식으로 나타내면 아래와 같다. $x_1 \sim x_d$ 는 행 벡터에 들어 있는 d 개의 변수이다. 화살표는 가중치 행렬 w 의 역할과 동일하다. (아래 화살표의 개수는 $p \times d$ 인데 W 는 이 값과 동일한 크기를 가진다.) 즉, 가중치 행렬의 데이터들은 화살표들의 값이라고 생각할 수 있다.



1. 소프트맥스

소프트맥스(softmax) 함수는 모델의 출력을 확률로 해석할 수 있게 변환해 주는 연산이다. 분류 문제를 풀 때 선형모델과 소프트맥스 함수를 결합해 예측한다.

이전에 설명했던 선형 모델의 결과물 \mathbf{o} 를 softmax함수에 넣으면 확률 벡터로 변환시킬 수 있다.

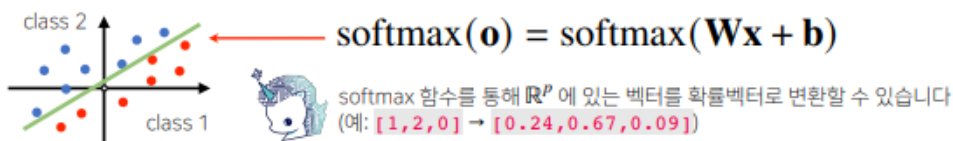
출력 벡터 \mathbf{o} 에 softmax 함수를 합성하면 확률벡터가 되므로 특정 클래스 k 에 속할 확률로 해석할 수 있다

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

$$\begin{bmatrix} -o_1 \\ -o_2 \\ \vdots \\ -o_n \end{bmatrix} = \begin{bmatrix} -x_1 \\ -x_2 \\ \vdots \\ -x_n \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1p} \\ w_{21} & w_{22} & \dots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \dots & w_{dp} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & \dots & b_p \end{bmatrix}$$

camp aitech

확률 벡터를 얻으면 주어진 데이터가 어떤 특정 클래스에 속할 확률이 얼마인지를 확인할 수 있다.



아래는 softmax()함수를 구현한 코드이다. denominator는 분자, numerator는 분모에 들어갈 것이다.

원리와는 다르게 np.max를 씌워줘서 vec 행렬에 빠졌다. 이는 지수함수를 사용하다 보니 너무 큰 벡터가 들어오면 overflow현상이 발생할 수 있어서 이를 방지하기 위함이다.

단, 학습을 할 때는 softmax()가 필요하지만, 추론을 할 때는 원-핫(one-hot)벡터로 최대값을 가진 주소만 1로 출력하면 된다.

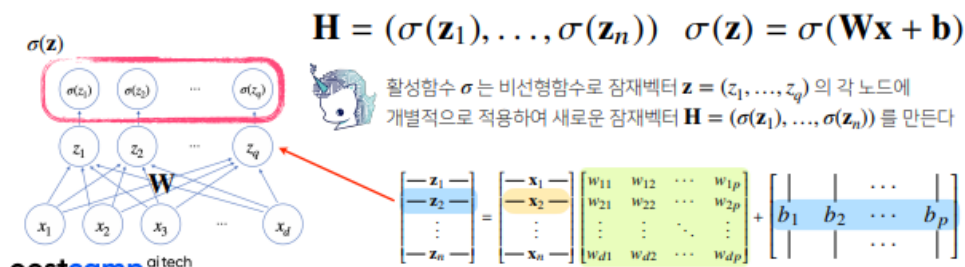
```
def softmax(vec):
    denominator = np.exp(vec - np.max(vec, axis=-1, keepdims=True))
    numerator = np.sum(denominator, axis=-1, keepdims=True)
    val = denominator / numerator
    return val

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

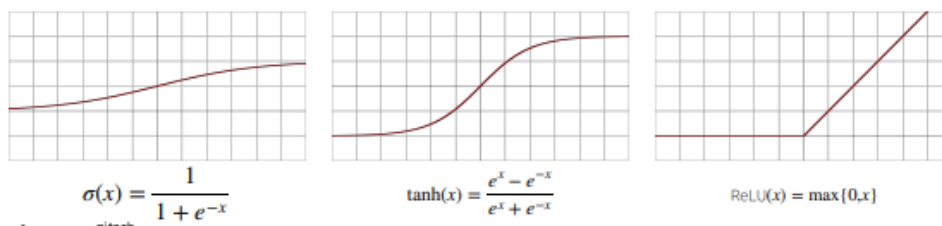
array([[2.44728471e-01, 6.65240956e-01, 9.00305732e-02],
       [9.00305732e-02, 2.44728471e-01, 6.65240956e-01],
       [2.06106005e-09, 4.53978686e-05, 9.99954600e-01]])
```

2. 신경망

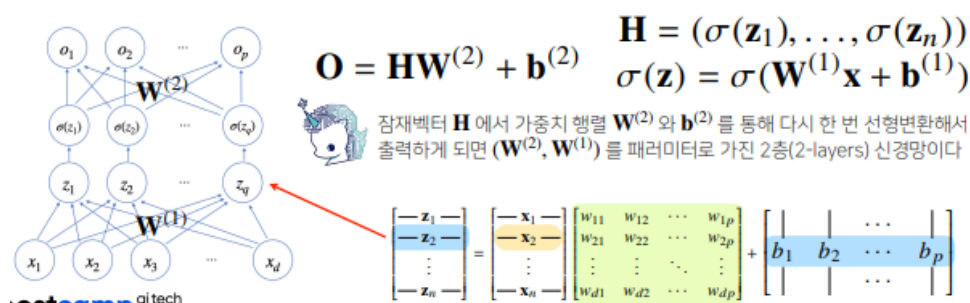
신경망은 선형 모델과 활성화함수(activation function)를 합성한 함수이다.



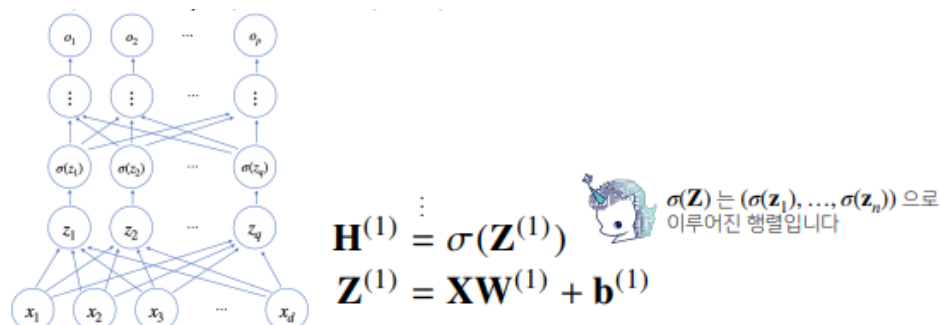
활성함수(activation function)는 비선형(nonlinear) 함수로서 딥러닝에서 매우 중요한 개념이다. 활성화함수를 쓰지 않으면 딥러닝은 선형 모델과 차이가 없다. sigmoid 함수나 tanh 함수는 전통적으로 많이 쓰이던 활성화함수지만 딥러닝에선 ReLU 함수를 많이 쓴다. ReLU 함수를 보면 선형 함수랑 별차이 없다고 생각할 수 있지만 전형적인 비선형 함수다. 또한 활성화함수로서 좋은 성질들을 가지고 있다.



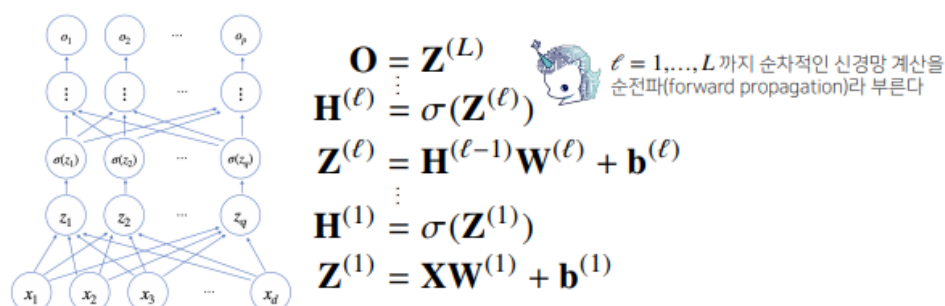
아래처럼 선형 모델이 2개가 되므로 이러한 신경망을 2-layer 신경망이라고 부른다. 이렇게 선형 모델과 활성화함수를 반복적으로 사용하는 것이 오늘날 사용하는 딥러닝의 가장 기본적인 모형이다.



그리고 이를 반복적으로 적용하면 신경망이 여러 층 합성된 함수로 **Multi-layer Perceptron(MLP)** 이라 불린다.

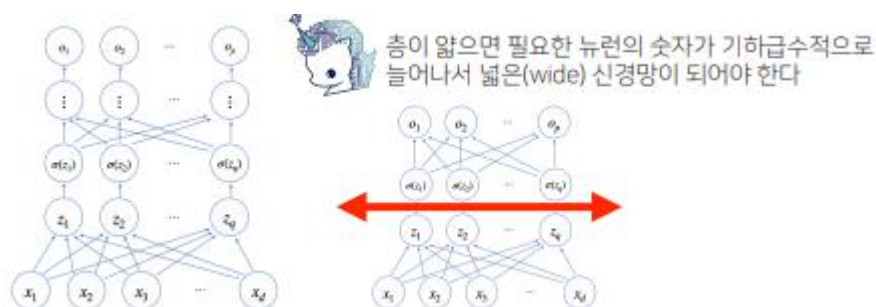


아래처럼 순차적으로 신경망을 계산하는 것을 **순전파(forward propagation)**이라고 부른다.



3. 층을 쌓는 이유

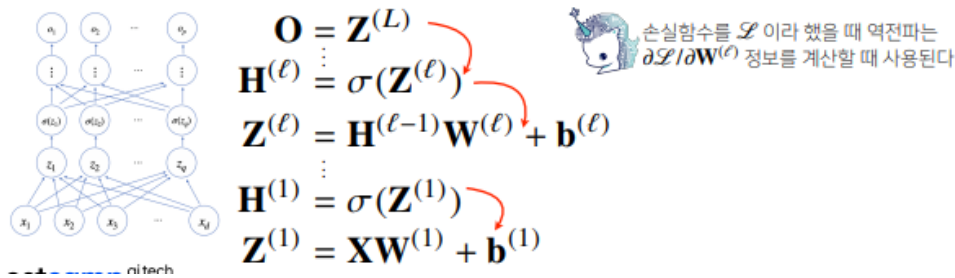
이론적으로 2층 신경망으로도 임의의 연속함수를 근사할 수 있다. 이를 universal approximation theorem이라 부른다. 그러나 층이 깊을수록 목적함수를 근사하는데 필요한 뉴런(노드)의 숫자가 훨씬 빨리 줄어들어 좀 더 효율적으로 학습이 가능하다.



단, 층이 깊어져도 최적화가 쉽다는 의미는 아니다. (이유는 이후 설명하신다고 하심.)

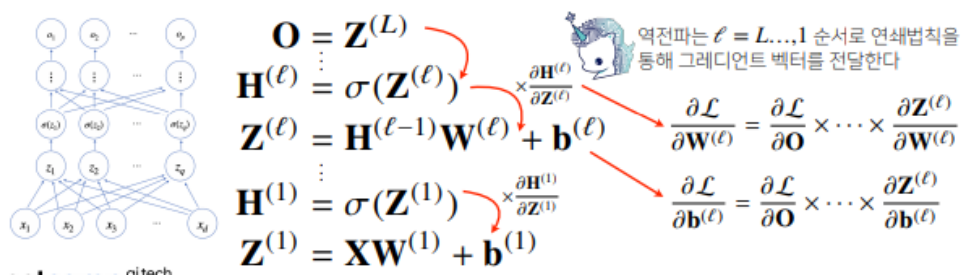
4. 역전파 알고리즘

딥러닝은 역전파(backward propagation) 알고리즘을 이용해 각 층에 사용된 파라미터를 학습한다. 각 층 파라미터의 그레디언트 벡터는 윗층부터 역순으로 계산된다.



역전파 알고리즘은 각각의 가중치 행렬 \mathbf{W} 에 대해 손실함수에 대한 미분을 계산할 때 사용하게 된다. 위에서 빨간색 화살표는 각 층에서 계산된 그래디언트를 밑에 있는 층에 전달된 flow이다. 이는 밑 층에 그래디언트를 계산할 때 윗 층의 그래디언트 벡터가 필요하기 때문이다.

딥러닝은 아래처럼 연쇄법칙을 통해 그래디언트를 밑 층에 전달하면서 가중치를 업데이트한다.



(원리)

역전파 알고리즘은 합성함수 미분법인 연쇄법칙(chain-rule)을 기반으로한 자동미분(auto-differentiation)을 사용한다. 연쇄법칙은 아래와 같은 원리로 진행된다.

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

$$z = w^2 \quad \longrightarrow \quad \frac{\partial z}{\partial w} = 2w$$


$$w = x + y \quad \longrightarrow \quad \frac{\partial w}{\partial x} = 1 \quad \frac{\partial w}{\partial y} = 1$$

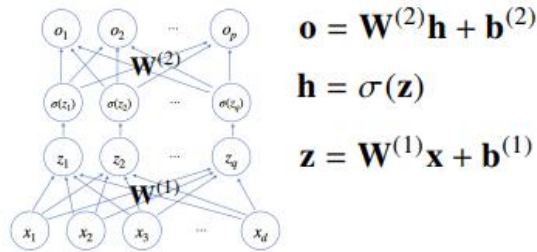
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial w} \frac{\partial w}{\partial x} = 2w \cdot 1 = 2(x + y)$$

각 노드의 텐서 값을 컴퓨터가 기억해야 미분 계산이 가능하다

2층 신경망에서 역전파 알고리즘이 어떻게 진행되는지 살펴보자.

이때 $\mathbf{W}^{(1)}$ 에 대해 우리가 경사하강법을 사용하고 싶다고하자.

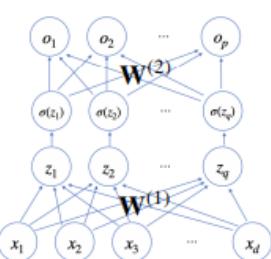
• 2층 신경망의 역전파 알고리즘 $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}} = ?$  $\mathbf{W}^{(1)}$ 은 행렬이므로 각 성분에 대한 편미분을 구해야 한다



파란색 화살표: forward propagation

빨간색 화살표: backward propagation

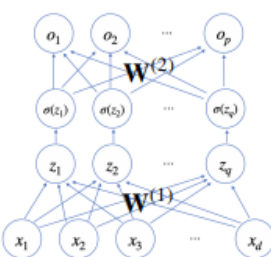
$\nabla_{\mathbf{W}^{(1)}} \mathcal{L} = (\nabla_{\mathbf{W}^{(1)}} \mathbf{z})(\nabla_{\mathbf{z}} \mathbf{h})(\nabla_{\mathbf{h}} \mathbf{o})(\nabla_{\mathbf{o}} \mathcal{L}) \longleftrightarrow \frac{\partial \mathcal{L}}{\partial W_{ij}^{(1)}} = \sum_{l,r,k} \frac{\partial \mathcal{L}}{\partial o_l}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




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

$\frac{\partial o_l}{\partial h_r} = W_{rl}^{(2)}$
 $\frac{\partial h_r}{\partial z_k} = \sigma'(z_k) \delta_{rk}$
 $\frac{\partial z_k}{\partial W_{ij}^{(1)}} = \frac{\partial}{\partial W_{ij}^{(1)}} \sum_{i'} x_{i'} W_{ik}^{(1)} = x_i \delta_{jk}$

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



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

$\frac{\partial o_l}{\partial h_r} = W_{rl}^{(2)}$  $\delta_{rk} \delta_{jk}$ 때문에 $r = k = j$ 만 남는다
 $\frac{\partial h_r}{\partial z_k} = \sigma'(z_k) \delta_{rk}$
 $\frac{\partial z_k}{\partial W_{ij}^{(1)}} = \frac{\partial}{\partial W_{ij}^{(1)}} \sum_{i'} x_{i'} W_{ik}^{(1)} = x_i \delta_{jk}$