🍎

11강: Introduction to Neural Networks

알파고는 그냥 망치로 깨부시면 끝!

~~근데 그건 사람도 마찬가지다...~~

Outline

앞서서는 계속 θ에 대해 linear한 모델들을 살펴보았는데 , 이제는 파라미터 θ와 입력값 x에 대해 모두 non-linear한 모델을 살펴보고자 한다. 이러한 non-linear한 모델 중 하나가 바로 이제 알아볼 neural network이다.

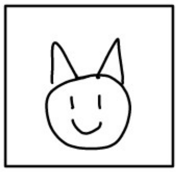
▼ 이해를 돕는 세가지 예시

1. Logistic Regression으로 본 신경망

Logistic Regression

goal : Find cats in images $\begin{cases} 1 \rightarrow \text{presence of a cat} \\ 0 \rightarrow \text{absence of a cat} \end{cases}$

64, 3, 64

$X =$  $= \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \rightarrow \boxed{wx+b \mid \sigma} \rightarrow \overset{(1.1)}{\hat{y}} = \sigma(\theta^T X)$

$= \sigma(\underset{(1,12288)}{w}\underset{(12288,1)}{x}+b)$

1) initialize w.b

2) Find the optimal w.b $\longrightarrow$

3) Use $\hat{y} = \sigma(wx+b)$ to predict

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

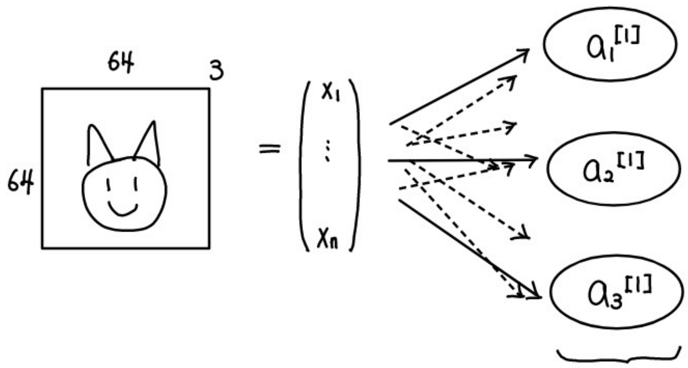
$$\begin{cases} w = w - \alpha \cdot \dfrac{\partial L}{\partial w} \\ b = b - \alpha \cdot \dfrac{\partial L}{\partial b} \end{cases}$$

Eq 1) newron = linear + activation

Eq 2) model = architecture + parameter

2. 이미지에서 고양이, 사자, 이구아나를 찾는 문제

goal 2 : Find cat / lion / iguana in image



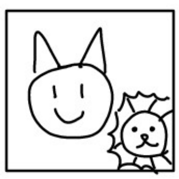
$\rightarrow \hat{y}_1 = a_1^{[1]} = \sigma(w_1^{[1]}x + b_1^{[1]})$

$\rightarrow \hat{y}_2 = a_2^{[1]} = \sigma(w_2^{[1]}x + b_2^{[1]})$

$\rightarrow \hat{y}_3 = a_3^{[1]} = \sigma(w_3^{[1]}x + b_3^{[1]})$

neurons independent

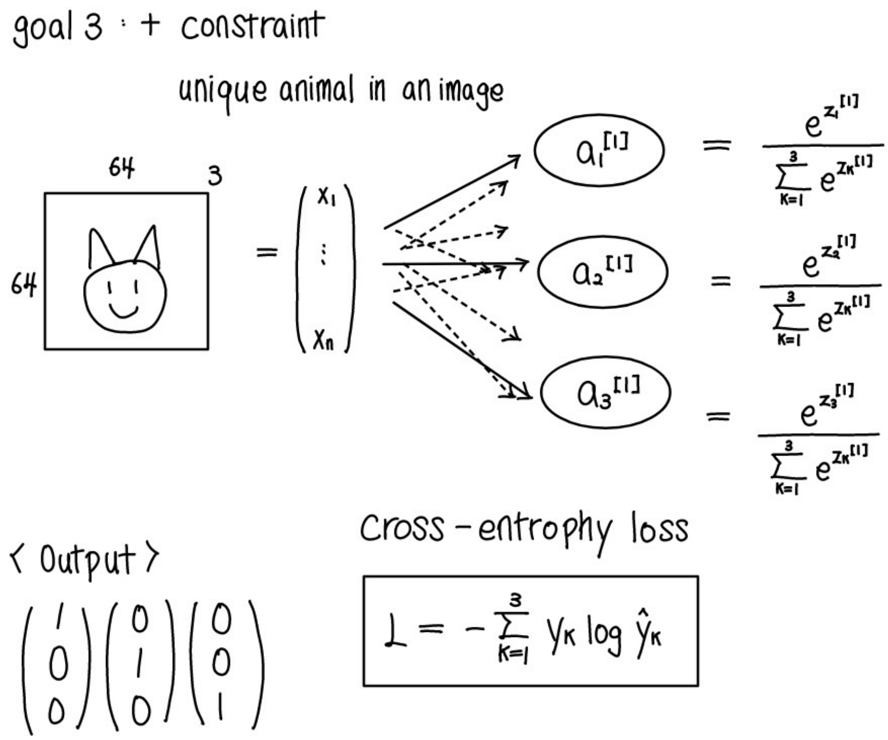
< Output >



$$\Rightarrow \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{array}{l} p(\text{cat}) \\ p(\text{lion}) \\ p(\text{iguana}) \end{array}$$

$$\underbrace{L_{3N} = -\sum_{k=1}^{3} \left[y_k \log \hat{y}_k + (1-y_k) \log (1-\hat{y}_k) \right]}_{\text{loss function}}$$

3. 소프트 맥스



Cost/loss function

알고 있지만…

- 회귀

$$\underbrace{J^{(i)}(\theta) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2}_{\text{개별값에 대한 loss function}} \rightarrow \underbrace{J(\theta) = \frac{1}{n}J^{(i)}(\theta)}_{\text{mean-square loss function}}$$

- 분류

$$\mathcal{L} = -\sum_{k=1}^{n} y_k log \hat{y}_k$$

Optimization 알고리즘

1. Define a **neural network** parameterization $h_\theta(x)$
2. Write the **backpropagation algorithm** to compute the gradient of the loss function $J^{(j)}(\theta)$
3. run SGD or mini batch SGD with the loss function $J^{(j)}(\theta)$
    a. **Algorithm 1** Stochastic Gradient Descent

> 1. Hyeperparameter : learning rate α, 총 반복 횟수 n_{iter}
> 2.
> θ을 랜덤으로 초기화
> 3.
> **for** i =1 to n_{iter} **do**
>
> Sample j uniformly from
> $\{1, \ldots, n\}$, and update θ by
>
> $$\theta := \theta - \alpha \nabla_{\theta} J^{(i)}(\theta)$$

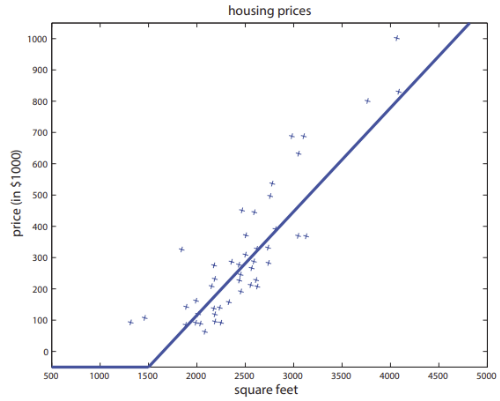
b. **Algorithm 2** Mini-batch Stochastic Gradient Descent

> 1. Hyeperparameter : learning rate α,batch size B,총 반복 횟수 n_{iter}
> 2.
> θ을 랜덤으로 초기화
> 3. for i =1 to
> n_{iter} do
>
> Sample B examples
> $j_1, \ldots j_B$(복원추출 x)
>
> uniformly from
> $\{1, \ldots, n\}$, and update θ by
>
> $$\theta := \theta - \alpha \nabla_{\theta} J^{(i)}(\theta)$$

Neural networks

A Neural Network with a Single Neuron.

집값 데이터를 다시 끌고와서 얘기해보자. 다만, 직선을 fit하는게 아니라



최소값을 0으로 설정해서 음의 값이 나오는 것을 방지한다.(이걸 'kink'라고 한다.)

그럼 , parameterized function $h_\theta(x)$는 다음과 같이 나타낼 수 있다.

$$h_\theta(x) = max(wx + b, 0), where\ \theta = (w, b) \in \mathbb{R}^2$$

이렇게 $max\{t, 0\}$꼴로 주어진 함수를 ReLU함수라고 한다.

Relu 함수는 일종의 activation function인데, 나중에 차차 알아보자.

만일 입력값 $x \in \mathbb{R}^d$가 다차원의 공간에 있다면,
single neuron의 neural network는 다음과 같다.

$$h_\theta(x) = ReLU(w^T x + b), \text{where } w \in \mathbb{R}^d, b \in \mathbb{R} \text{ and } \theta = (w, b)$$

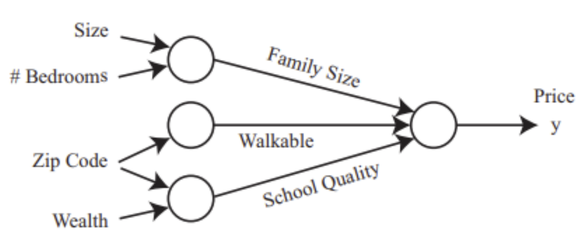
이때, b를 bias, w를 weight vector라고 한다.

위의 식은 neural network가 layer가 1개 있는 경우이다.

Stacking Neurons.

더 복잡한 neural network를 만들고 싶으면 어떡하나요?

single neuron을 여러 개 쌓으면 된다!



앞서서와 마찬가지로 계속 집값 예측을 해보자.

size, number of bedrooms, zip code, wealth (각각 x_1, x_2, x_3, x_4)같은 input features가 주어진다면, 우리는 각각의 feature들의 정보를 이용해서 위의 그림처럼 intermediate feature을 만들 수 있다.

(a_1, a_2, a_3, a_4; a_i를 hidden units, hidden neurons)라고 한다.

a_i를 수식을 이용해 적어보자면 다음과 같다.

$$
\begin{aligned}
a_1 &= ReLU(\theta_1 x_1 + \theta_2 x_2 + \theta_3) \\
a_2 &= ReLU(\theta_4 x_3 + \theta_5) \\
a_3 &= ReLU(\theta_6 x_3 + \theta_7 x_4 + \theta_8)
\end{aligned}
$$

그럼 , parameterized function $h_\theta(x)$는 다음과 같이 나타낼 수 있다.

$$
h_\theta(x) = \theta_9 a_1 + \theta_{10} a_2 + \theta_{11} a_3 + \theta_{12}
$$
$$
\theta = \{\theta_1 ... \theta_{12}\}
$$

Fully Connected Neural Network

앞서서 집값을 예측하는 경우에는 우리의 사전 정보를 이용해서 일종의 ‘파생변수’를 만들었지만 ,
다른 경우에는 그러한 사전정보가 없을 수도 있다.

(사전 정보를 이용해 일종의 ‘파생변수’를 만드는 것을 feature engineering)이라고 한다.

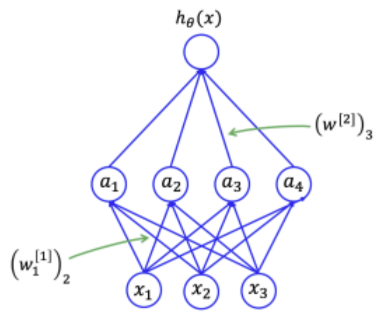
Neural Network은 데이터 간에서 이러한 중요성과 관계를 스스로 학습한다.

→ generic parameterization 을 이용해 indermediate variable을 표현해보자.

간단한 방법은 a_i를 input variable x_1, x_2, x_3, x_4 전부를 이용해 표현하는 것이다.

$$
\begin{aligned}
a_1 &= ReLU(w_1^\top x + b), \text{where } w_1 \in \mathbb{R}^4, b_1 \in \mathbb{R} \\
a_2 &= ReLU(w_2^\top x + b), \text{where } w_2 \in \mathbb{R}^4, b_2 \in \mathbb{R} \\
a_3 &= ReLU(w_3^\top x + b), \text{where } w_3 \in \mathbb{R}^4, b_3 \in \mathbb{R}
\end{aligned}
$$

이렇게 표현된 a_i를 이용해 $h_\theta(x)$를 정의할 때, 우리는 이를 fully-connected neural network라 한다.



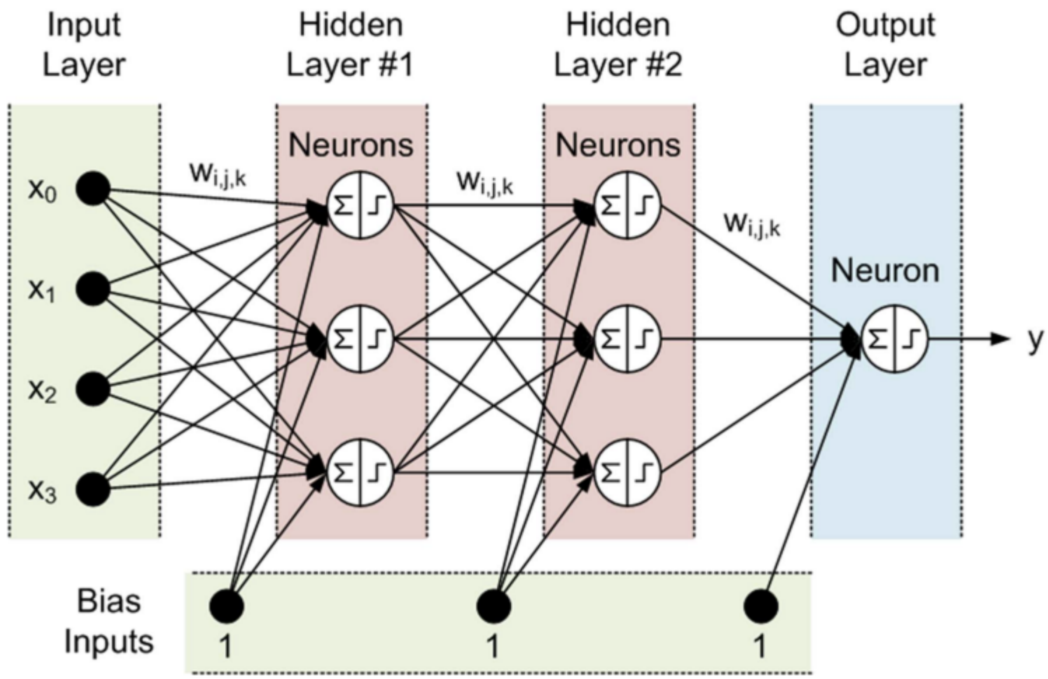
Two-layer Fully-Connected Neural Networks.
activation
a_j는 각각 x_i들의 가중합으로 표현된다.

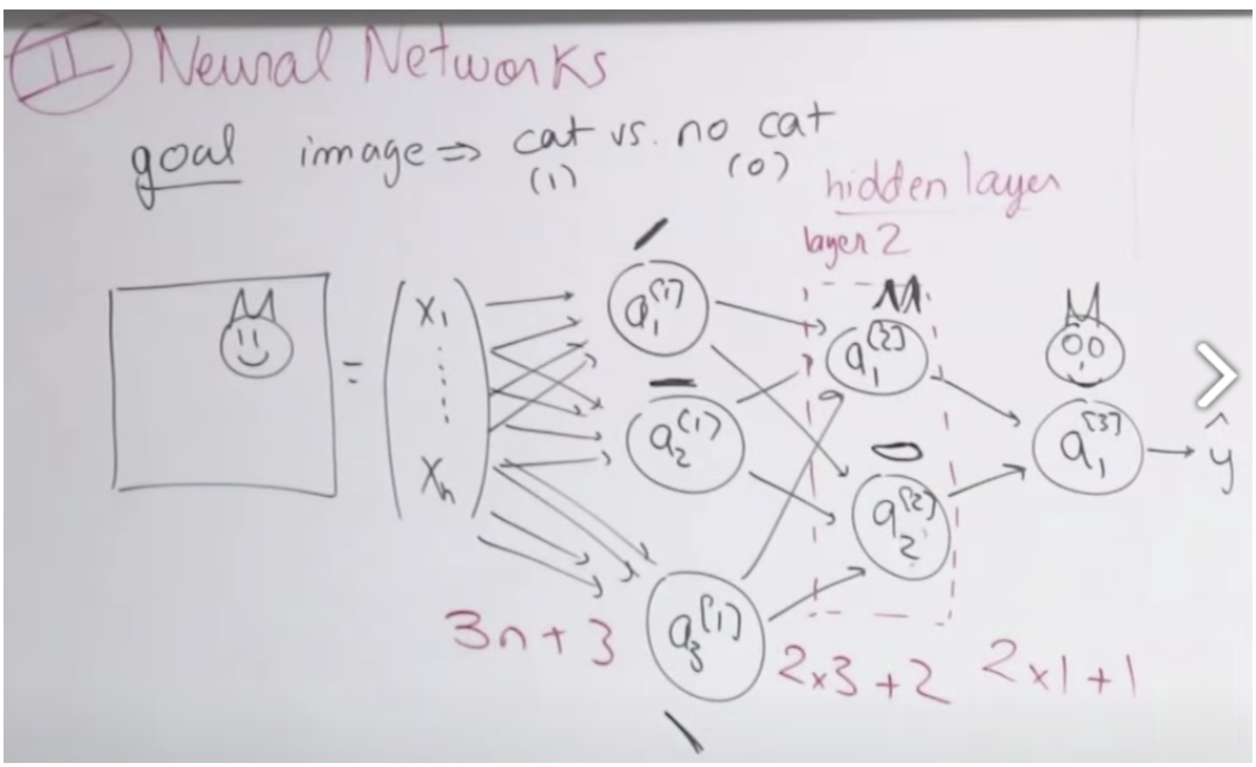
위 그림의 Two-layer Fully-Connected Neural Networks를 식으로 표현해보자면 다음과 같다.

$$
\begin{aligned}
\forall j \in [1, ...m],\ z_j &= w_j^{[1]^\top} x + b_j^{[1]}, \text{where } w_j^{[1]^\top} \in \mathbb{R}^d, b_j^{[1]} \in \mathbb{R} \\
a_j &= ReLU(z_j), \\
a &= [a_1, ...a_m]^\top \in \mathbb{R}^m \\
h_\theta(x) &= w^{[2]^\top} a + b^{[2]}, \text{where } w^{[2]^\top} \in \mathbb{R}^d, b^{[2]} \in \mathbb{R}
\end{aligned}
$$

Multi-layer fully-connected neural networks

$$
\begin{aligned}
a^{[1]} &= ReLU(W^{[1]} x + b^{[1]}) \\
a^{[2]} &= ReLU(W^{[2]} a^{[1]} + b^{[2]}) \\
\cdots \\
a^{[r-1]} &= ReLU(W^{[r-1]} a^{[r-2]} + b^{[r-1]}) \\
h_\theta(x) &= W^{[r]} a^{[r-1]} + b^{[r]}
\end{aligned}
$$





딥러닝 모형의 특징

- end to end learning : 입력과 출력을 정의하고 네트워크 모형을 정의한 뒤에 데이터를 통해 훈련하는 방법 , 중간에 사람이 개입하여 작업하는 것이 없음

- blackbox : 사람은 모델이서 어떤 것을 학습해서 다음 레이어에 전달하고 또 그것이 의미하는 것이 무엇인지 알 수 없기 때문에 딥러닝을 블랙박스 모델이라고 부름.

Vectorization

입력값과 은닉층의 차원이 높기 때문에, 반복문을 쓰면 코드가 느리게 돈다.

따라서, 병렬 처리가 중요하다. 연산을 빠르게 하기 위해서, 반복문 대신 vectorization을 쓴다.

- weight matrix $W^{[1]}$=$[w^{[1]^\top}]$

$$W^{[1]} = \begin{bmatrix} -w_1^{[1]^\top}- \\ -w_2^{[1]^\top}- \\ \vdots \\ -w_m^{[1]^\top}- \end{bmatrix} \in \mathbb{R}^{m\times d}$$

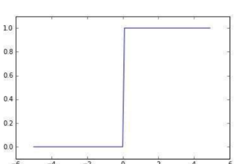
- $z = W^{[1]}x + b^{[1]}$

$$\underbrace{\begin{bmatrix} z_1 \\ \vdots \\ \vdots \\ z_n \end{bmatrix}}_{z\in\mathbb{R}^{m\times 1}} = \underbrace{\begin{bmatrix} -w_1^{[1]^\top}- \\ -w_2^{[1]^\top}- \\ \vdots \\ -w_m^{[1]^\top}- \end{bmatrix}}_{W^{[1]}\in\mathbb{R}^{m\times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{x\in\mathbb{R}^{d\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}}_{b^{[1]}\in\mathbb{R}^{m\times 1}}$$

Activation Function

- 이름 그대로 뉴런이 얼마나 활성화 되는지 결정
- 신경망에 비선형성을 더해주기 위함 → 활성화 함수는 비선형적인 모양을 가짐
- 목적과 성능에 따라 선택하면 됨

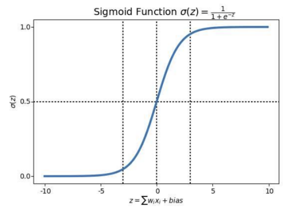
1. 계단함수 (Step Function)



- 0 또는 1의 값만 갖는 함수
- 단층 퍼셉 트론의 예시와 같이 특정 값을 넘으면 1, 넘지 못하면 0
- 두가지 값 외에는 표현할 수 없기 때문에 사용 빈도가 낮음

2. 시그모이드 함수 (Sigmoid Function)

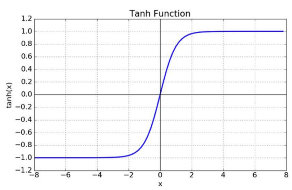
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



- 0과 1사이 값 가짐
- 미분 가능 → 역전파 가능
- 0과 1 사이값이 나오는 영역 매우 작음 → 많은 값이 0과 1에 수렴
- 중앙값이 0.5이기 때문에 최적화에 어려움

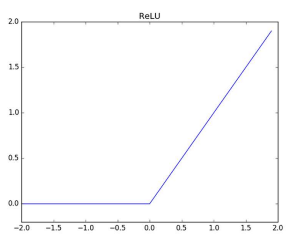
3. 하이퍼볼릭탄젠트 함수 (tanh)

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



- 중앙값이 0으로 음수 출력 가능
- 출력값의 범위가 더 크기 때문에 학습 더 효율적으로 수행 가능

4. ReLU (Rectified Linear Unit)

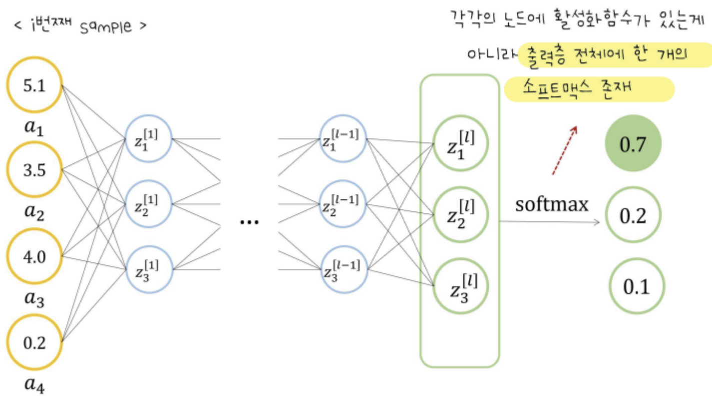


- 기울기 소실 문제를 해결한 활성화 함수
  (∵ 출력값의 범위가 아주 넓지만 특정 수로 수렴하지 않기 때문)
- 계산이 쉬움 → 모델 학습시간 줄어듬
- 은닉층의 활성화 함수로 자주 사용

5. 소프트맥스 (Softmax)

$$y_k = \frac{exp(a_k)}{\sum_{i=1}^{n} exp(a_i)} = \frac{C exp(a_k)}{C \sum_{i=1}^{n} exp(a_i)} = \frac{exp(a_k + logC)}{\sum_{i=1}^{n} exp(a_i + logC)}$$

- 다중 분류(multi-classification) 문제에서 출력층에서 쓰이는 활성화함수
- 출력값을 모두 더하면 1이 됨



▼ 왜 $\sigma(z) = z$이면 안 될까?

$$
\begin{aligned}
h_\theta(x) &= W^{[2]}a^{[1]} & \\
&= W^{[2]}\sigma(z^{[1]}) & \text{by definition} \\
&= W^{[2]}z^{[1]} & \text{since } \sigma(z) = z \\
&= W^{[2]}W^{[1]}x & \text{from Equation (7.13)} \\
&= \tilde{W}x & \text{where } \tilde{W} = W^{[2]}W^{[1]}
\end{aligned}
$$

Without non-linear activation functions, the neural network will simply perform linear regression