LIKE LION

4강

인공신경망, 그게 대체 뭐죠?

WHATEVER YOU WANT, MAKE IT REAL.

강사 윤영석

- Unsupervised Learning:
  - input feature에서 의미 있는 패턴을 찾는 학습 방법
  - 데이터 분석 및 시각화, 전처리, 이상 탐지 등의 목적으로 사용

- Dimensionality Reduction
  - curse of dimensionality
  - PCA (분산 보존), MDS (거리 보존), t-SNE (Neighborhood 보존)

- Clustering
  - partitioning clustering: K-means
  - hierarchical clustering: Agglomerative Clustering
  - density-based clustering: DBSCAN

LIKE LION

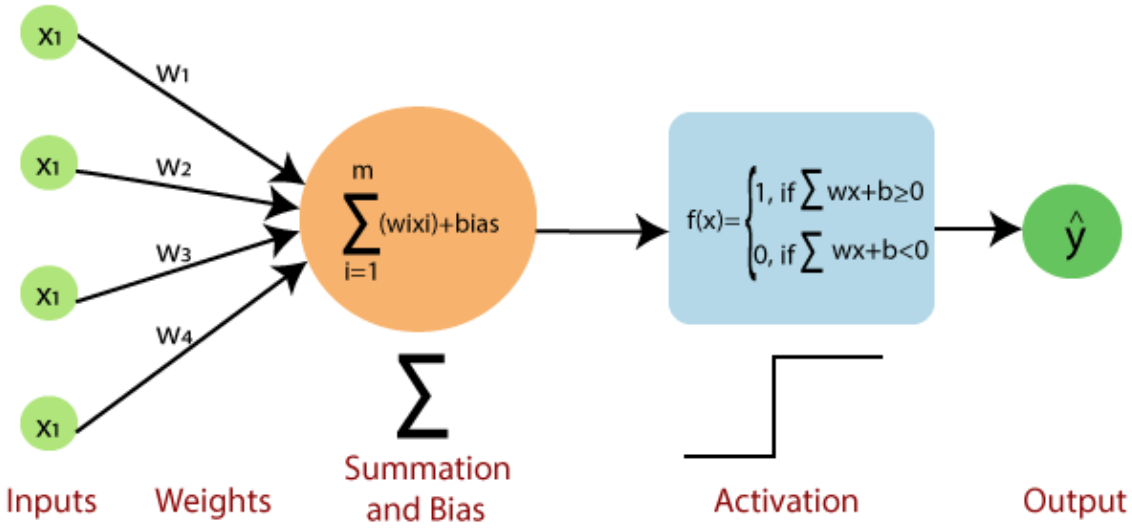
1. Multi-Layer Perceptron의 이해
2. Deep Learning의 등장
3. Forward Pass – MLP는 어떻게 작동할까요?
4. Activation Function이 왜 필요해요?
5. Loss Function은 뭐로 할까요?
6. (실습) PyTorch Tutorial
7. (실습) MLP MNIST classification (1)

LIKE LION

{ 1. Multi-Layer Perceptron의 이해 }

Perceptron

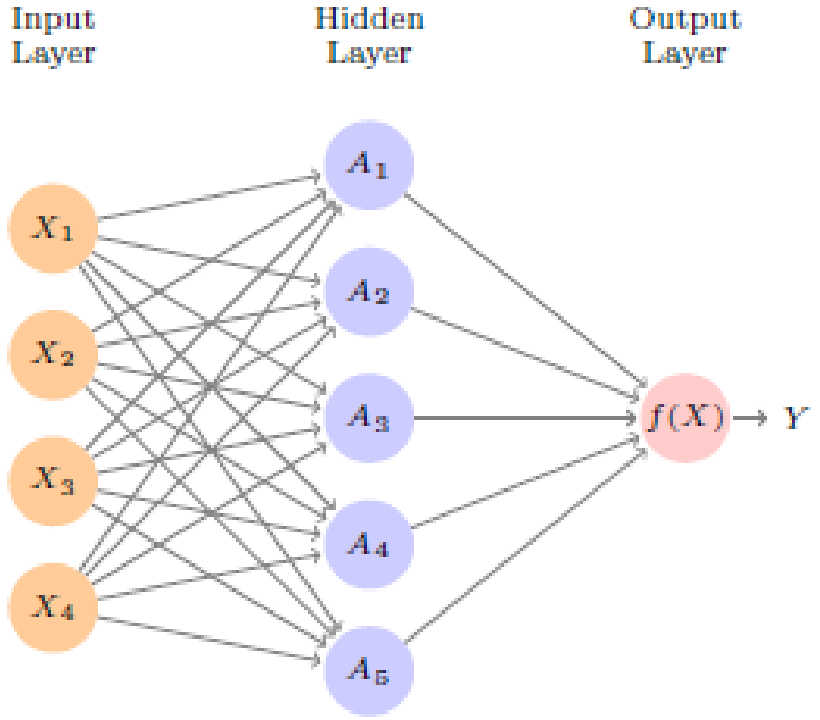
- 입력 벡터에 가중치 weights을 곱해 정답 출력 값을 얻는 알고리즘
  - 독립된 알고리즘으로 소개되어 등장했지만 선형 모델과 굉장히 비슷한 구조를 가짐
  - Weights과 activation function을 통해 출력 값을 얻으며 loss function을 이용해 weights을 학습함



source: https://www.javatpoint.com/single-layer-perceptron-in-tensorflow

Multi-Layer Perceptron (MLP)

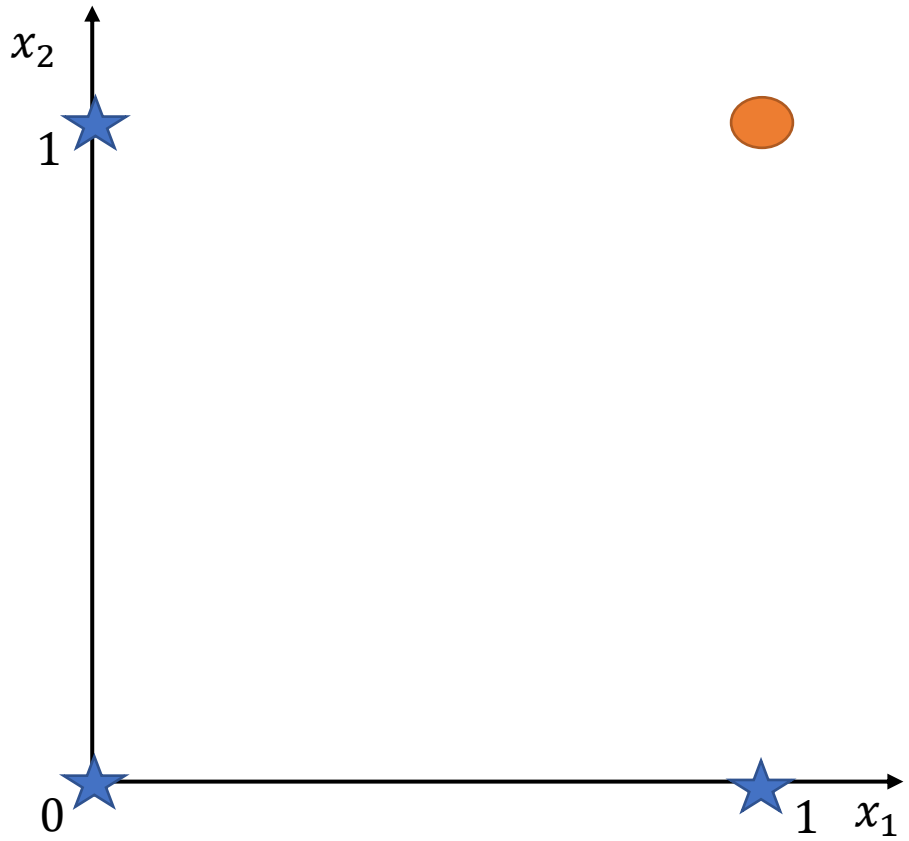
- 선형 모델(Linear model, single-layer perceptron)의 확장
  - 기존의 선형 모델이 해결하기 어려운 문제를 해결하기 위한 시도
  - XOR gate problem



source: https://www.statlearning.com/resources-second-edition

Logic Gate Problems

- 2개의 변수를 입력으로 받아 논리에 맞는 값을 출력

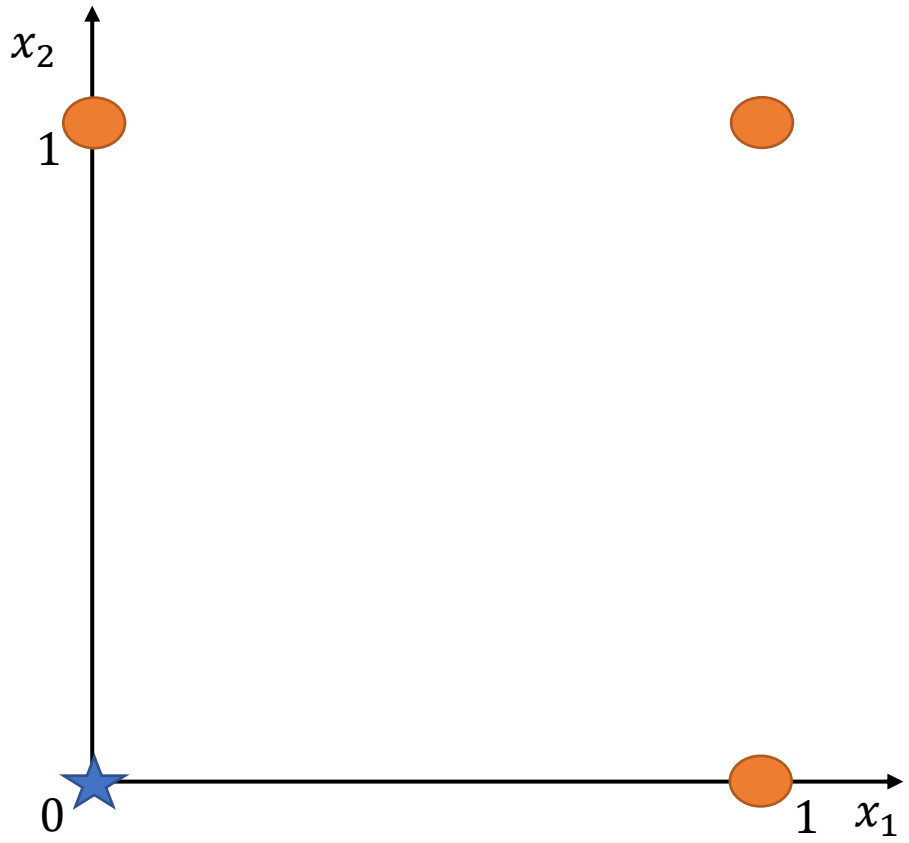


x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$$y = x_1 x_2$$

Logic Gate Problems

- 2개의 변수를 입력으로 받아 논리에 맞는 값을 출력

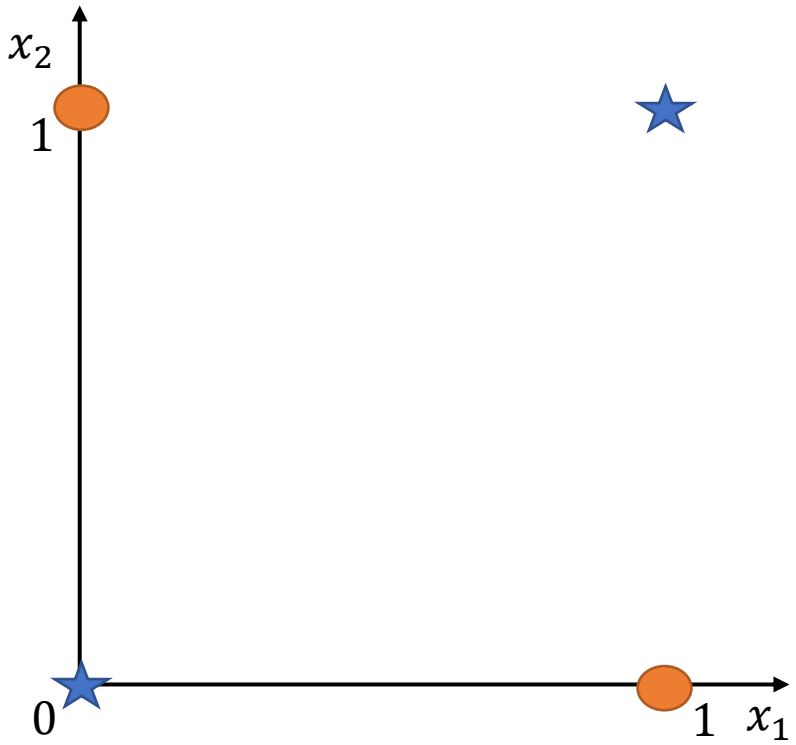


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

$$y = x_1 + x_2$$

XOR Gate Problem

- 2개의 변수를 입력으로 받아 입력 변수의 값이 같으면 0, 다르면 1을 출력하는 논리 게이트.

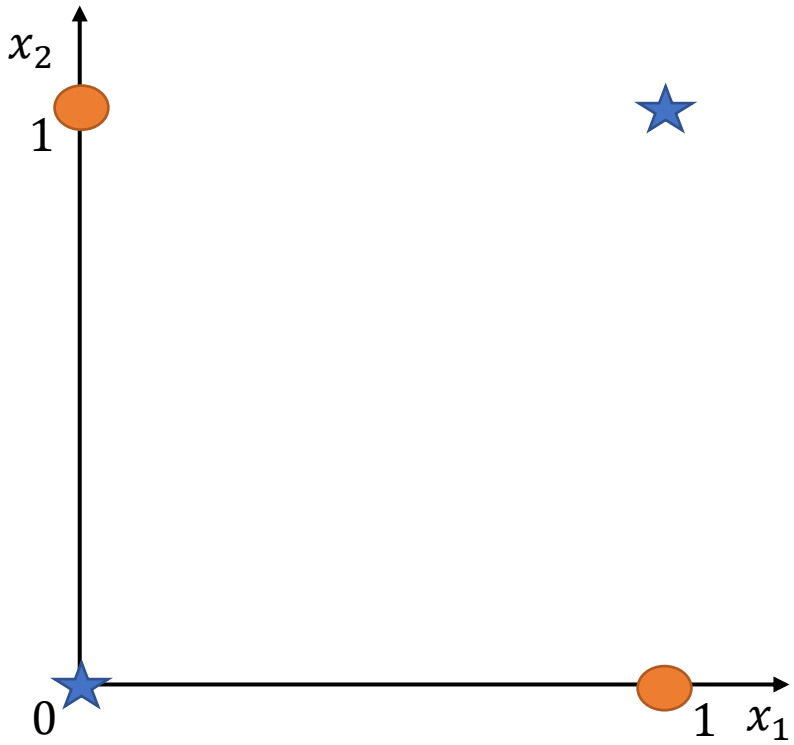


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$y = w_1x_1 + w_2x_2 + b$$

XOR Gate Problem

- 2개의 변수를 입력으로 받아 입력 변수의 값이 같으면 0, 다르면 1을 출력하는 논리 게이트.



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$y = w_1x_1 + w_2x_2 + b$$

$$0 = b$$

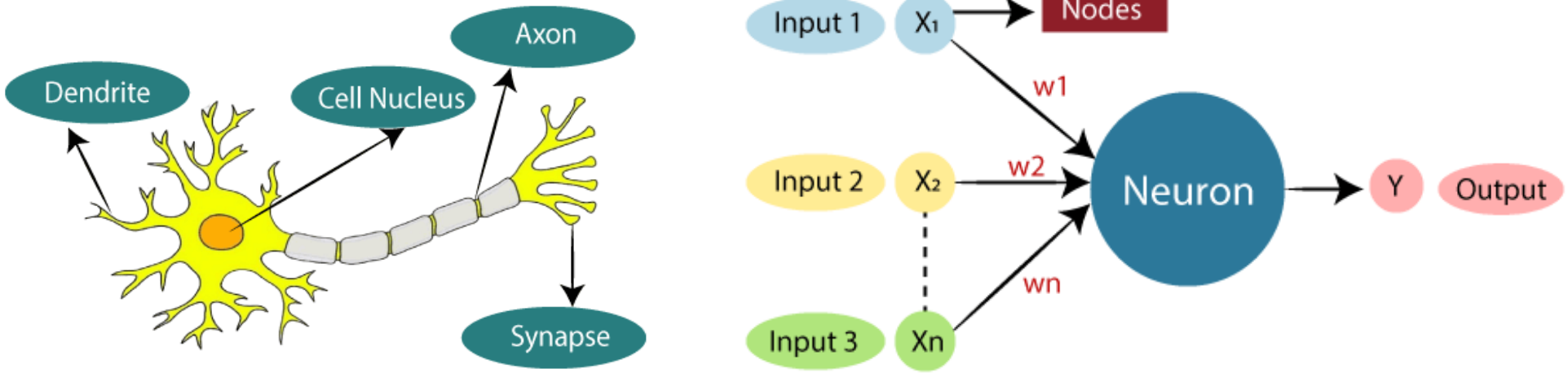
$$1 = w_1 + b$$

$$1 = w_2 + b$$

$$0 = w_1 + w_2 + b$$

Multi-Layer Perceptron (MLP)

- Artificial Neural Network ANN
  - 인공신경망
- Fully Connected Neural Network FCNN
- Affine Network
- Feed Forward Neural Network FFNN

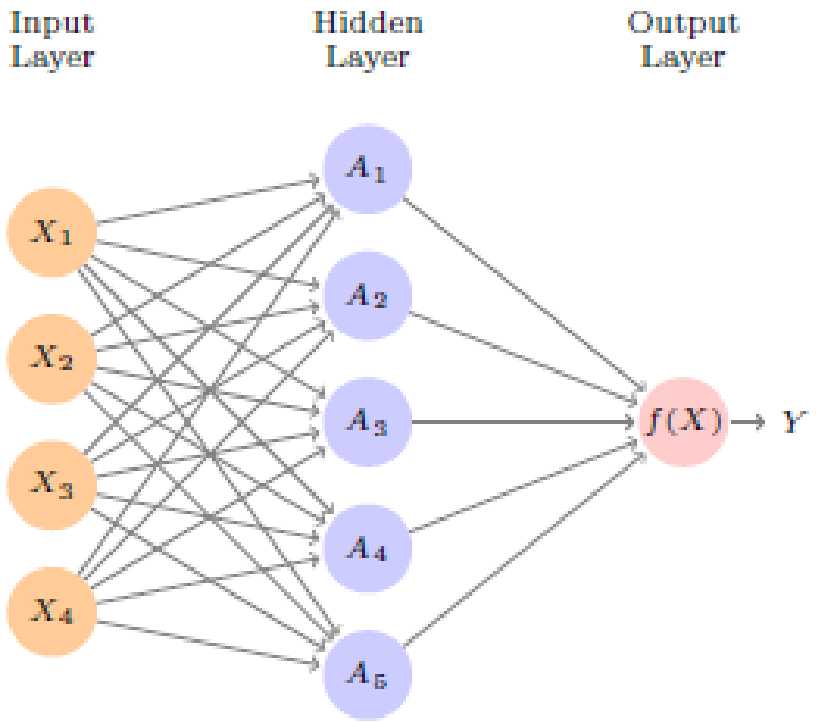


source: https://www.javatpoint.com/artificial-neural-network

LIKE LION

Structure of MLP

- Parameters
  - Weights
  - Bias
- Activation function
- Loss function



source: https://www.statlearning.com/resources-second-edition

How does MLP work?

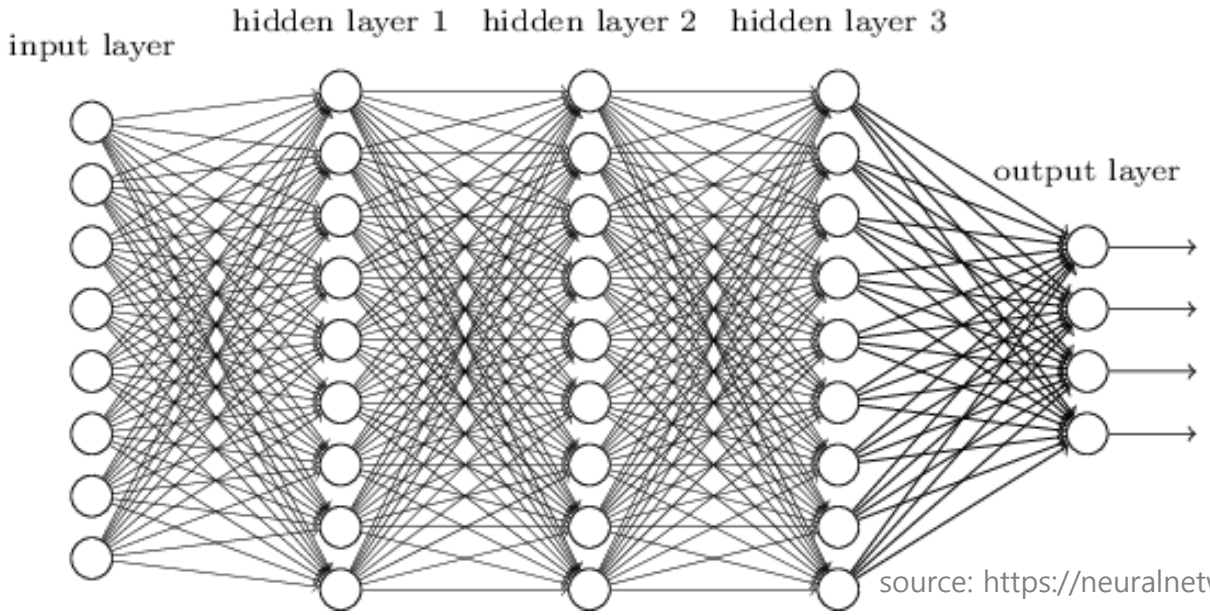
- Forward Pass
  - 입력이 주어진 상태에서 MLP 모델이 출력을 추론하는 과정
  - Parameters와 activation function을 이용해 각 입력에 대해 출력을 함수의 형태로 얻음
- Backward Pass
  - Back propagation
  - Forward pass로 얻은 출력 추론 값과 해당 입력의 참 출력 값을 이용해 loss 값을 얻어, 전체 모델을 학습하는 과정
  - Loss로부터 얻은 error signal을 back propagation 알고리즘을 이용해 모든 parameter에 전파

LIKE LION

{ 2. Deep Learning의 등장 }

History of Deep Learning

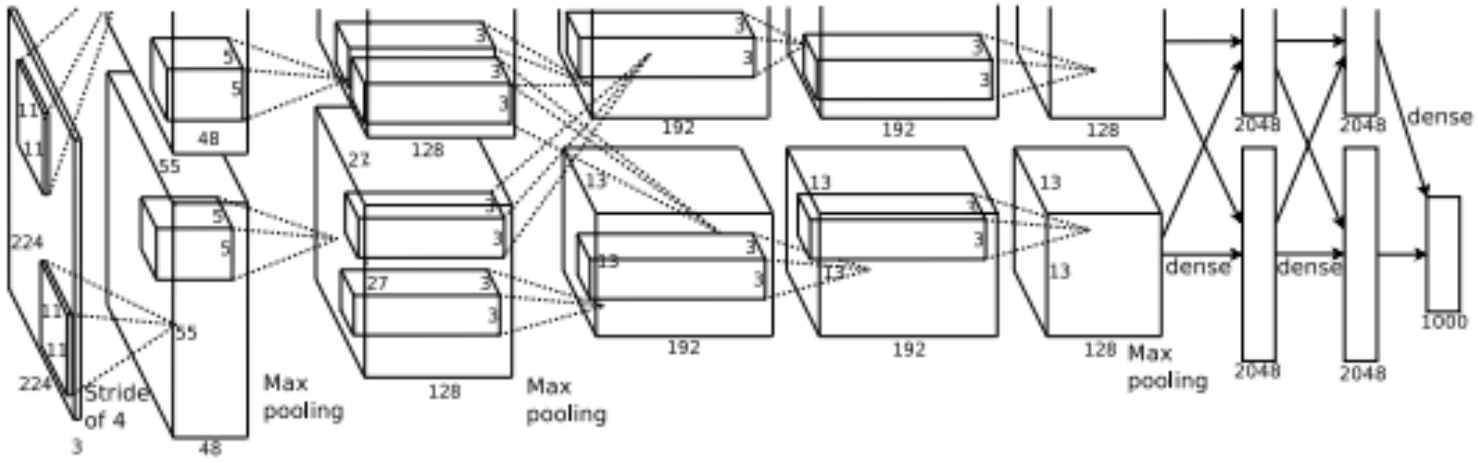
- MLP의 아이디어는 일찍이 등장했지만 계산 자원의 부족과 학습 방법의 한계로 인해 큰 주목을 받지 못 함
  - 20세기 중반, perceptron 알고리즘, stochastic gradient descent 알고리즘 등이 제안됨
  - 20세기 말, neural network 구조, back propagation 알고리즘 등이 제안됨



source: https://neuralnetworksanddeeplearning.com/chap6.html

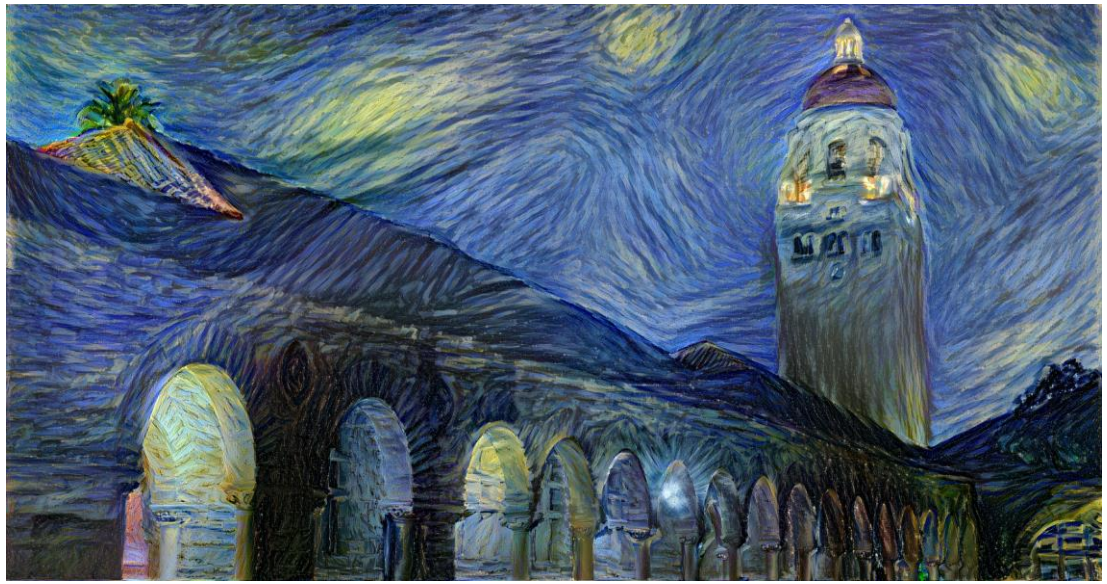
Resurgence of Deep Learning

- 근래의 하드웨어 발전과 다양한 학습 기법들의 개발로 MLP의 학습이 가능해짐
  - GPU, TPU, cloud computing, etc.
  - Backpropagation, initialization, activation, regularization, etc.
  - Big data
- 점점 많은 수의 층을 쌓아 모델을 구성하여 표현력을 키우는 방향으로 접근



source: ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky, et al.

Successful Applications

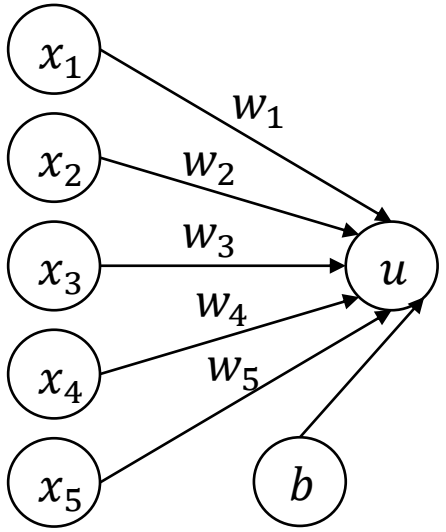


LIKE LION

{ 3. Forward Pass – MLP는 어떻게 작동할까요? }

Forward Pass

- 입력이 주어졌을 때 parameter와 activation function을 통해 출력을 추론하는 과정
  - 각 층에서 입력이 weights과 곱해지는 선형적인 형태를 가지기 때문에 행렬곱의 형태로 간단하게 표현 가능



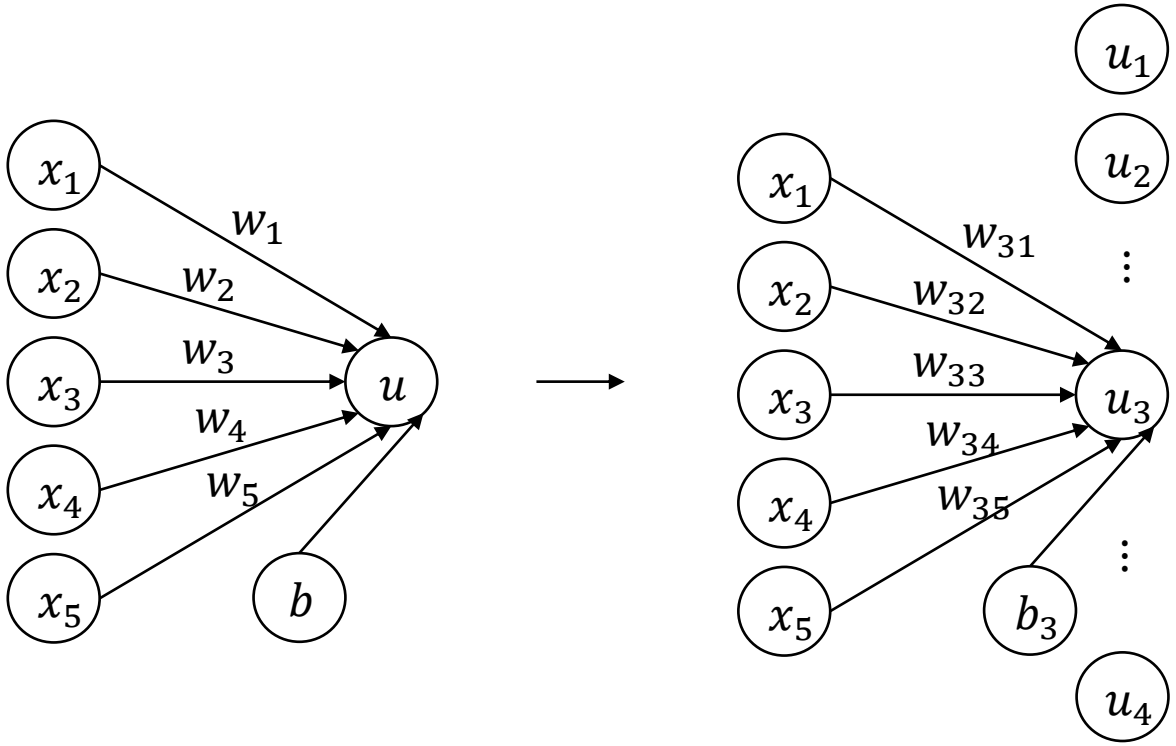
$$u = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + b$$

$$= \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + b$$

$$= \vec{w}^T \cdot \vec{x} + b$$

$$\vec{x} \in R^5, \vec{w} \in R^5$$

Forward Pass



$$u_1 = \overrightarrow{w_1}^T \cdot \vec{x} + b_1$$

$$u_2 = \overrightarrow{w_2}^T \cdot \vec{x} + b_2$$

$$u_3 = \overrightarrow{w_3}^T \cdot \vec{x} + b_3$$

$$u_4 = \overrightarrow{w_4}^T \cdot \vec{x} + b_4$$

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} \overrightarrow{w_1}^T \\ \overrightarrow{w_2}^T \\ \overrightarrow{w_3}^T \\ \overrightarrow{w_4}^T \end{bmatrix} \cdot \vec{x} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$\overrightarrow{w_i}^T = [w_{i1} \quad w_{i2} \quad w_{i3} \quad w_{i4} \quad w_{i5}]$$

$$\vec{x} \in R^5, \overrightarrow{w_i} \in R^5, \vec{u} \in R^4$$

Batch Training

- 실제 machine learning의 학습과 추론은 하나의 데이터를 이용하는 게 아니라 여러 개의 데이터를 묶어서 진행
  - 이러한 데이터의 묶음을 batch라고 부르며, 대부분의 machine learning과 deep learning framework는 batch 단위의 operation이 가능
  - MLP의 forward pass도 마찬가지로 batch로 진행
    - 이를 위해 행렬과 벡터의 곱이 행렬과 행렬의 곱으로 확장
    - 실제로 모델의 학습 (backward pass, back-propagation)도 행렬과 행렬의 곱으로 표현되어 진행됨

Matrix Multiplication

- 2차원 행렬들의 곱
- Machine learning과 deep learning에서 parameter와 data 사이의 가장 기본적인 연산 중 하나로 gpu를 통한 가속으로 deep learning의 성공을 가능하게 함

$$A = [A_{ij}] = \begin{bmatrix} \overrightarrow{A_1}^T \\ \vdots \\ \overrightarrow{A_n}^T \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{bmatrix} \qquad \mathrm{B} = [B_{ij}] = \begin{bmatrix} \overrightarrow{B_1} & \cdots & \overrightarrow{B_p} \end{bmatrix} = \begin{bmatrix} B_{11} & \cdots & B_{1p} \\ \vdots & \ddots & \vdots \\ B_{m1} & \cdots & B_{mp} \end{bmatrix}$$

$$C = AB = \begin{bmatrix} \overrightarrow{A_1}^T \\ \vdots \\ \overrightarrow{A_n}^T \end{bmatrix} \begin{bmatrix} \overrightarrow{B_1} & \cdots & \overrightarrow{B_p} \end{bmatrix} = \begin{bmatrix} \overrightarrow{A_1}^T \cdot \overrightarrow{B_1} & \cdots & \overrightarrow{A_1}^T \cdot \overrightarrow{B_p} \\ \vdots & \ddots & \vdots \\ \overrightarrow{A_n}^T \cdot \overrightarrow{B_1} & \cdots & \overrightarrow{A_n}^T \cdot \overrightarrow{B_p} \end{bmatrix}$$

$$\overrightarrow{A_i} = [A_{i1} \quad \cdots \quad A_{im}]^T \in R^m$$

$$\overrightarrow{B_i} = [B_{1i} \quad \cdots \quad A_{mi}]^T \in R^m$$

$$A \in R^{n \times m} \qquad B \in R^{m \times p} \qquad C \in R^{n \times p}$$

Batch Training

- 하나의 데이터를 벡터로 표현했을 때, 데이터의 묶음 (batch)를 행렬로 표현해 parameters와의 곱을 행렬과 행렬의 곱으로 표현
  - 이는 1차원 데이터를 쌓아 2차원으로 만드는 것으로, 2차원 혹은 3차원의 데이터는 3차원 혹은 4차원으로 쌓임

$$\vec{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} = \begin{bmatrix} \overrightarrow{w_1}^T \\ \vdots \\ \overrightarrow{w_m}^T \end{bmatrix} \cdot \vec{x} + \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \longrightarrow U = [\overrightarrow{u_1} \quad \cdots \quad \overrightarrow{u_B}] = \begin{bmatrix} \overrightarrow{w_1}^T \\ \vdots \\ \overrightarrow{w_m}^T \end{bmatrix} [\overrightarrow{x_1} \quad \cdots \quad \overrightarrow{x_B}] = WX$$

$\vec{u} \in R^m, \vec{x} \in R^n, \overrightarrow{w_i} \in R^n$

$\overrightarrow{u_j} \in R^m, \overrightarrow{x_j} \in R^n, \overrightarrow{w_i} \in R^n$

$X \in R^{n \times B} \quad W \in R^{m \times n} \quad U \in R^{m \times B}$

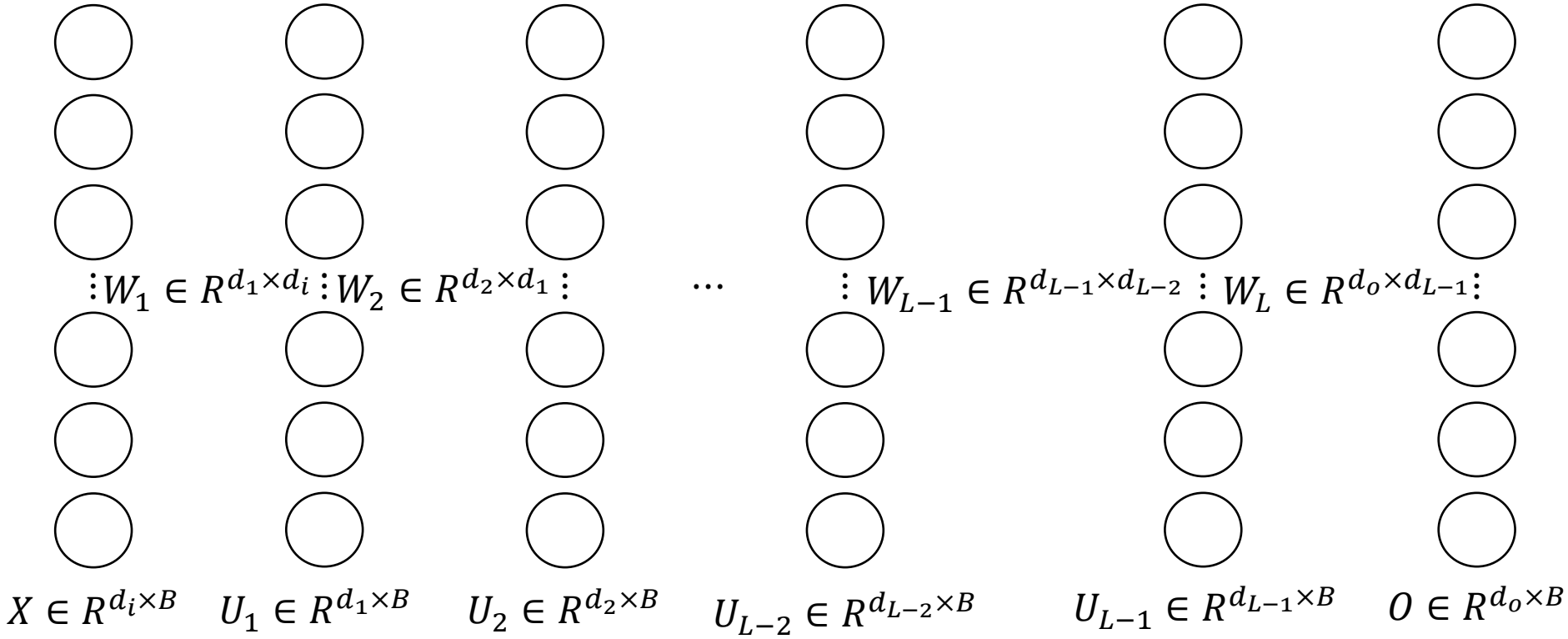
Mini-Batch Training

- 효율적인 학습을 진행하는 동시에 학습 과정에 randomness를 추가해주기 위해 데이터 셋 전체를 하나의 batch로 보는 것이 아니라 그 일부를 mini-batch로 취급하여 전체 데이터 셋으로 여러 번의 학습 과정을 진행
  - 효율적인 학습에는 시간과 메모리 관점이 둘 다 포함
  - 일반적으로는 학습을 진행하는 디바이스의 gpu memory가 허용하는 내에서 최대의 batch size를 사용
- 전체 데이터 셋으로 학습을 진행한 것을 1 epoch이라고 부르며 전체 학습 과정은 여러 epoch을 가짐
  - 각 epoch는 여러 개의 mini-batch를 통한 학습을 포함함
  - 1 epoch은 Data set size / batch size 번의 update로 진행

LIKE LION

Forward Pass

- 데이터로 batch를 구성해 2차원 이상의 행렬이 되기 때문에 각 layer에서의 연산은 행렬과 행렬의 곱



LIKE LION

{ 4. Activation Function이 왜 필요해요? }

Why?

- 각 layer의 연산은 행렬 곱으로, 선형 연산임
  - 행렬의 곱은 행렬이 되기 때문에 layer에서 다른 처리를 하지 않으면 여러 행렬들의 곱은 역시 하나의 행렬로 표현이 되고, 많은 layer를 겹치는 것이 의미가 없음
  - 중간에 모델의 선형성을 없애 줄 요소가 필요
    - 비선형 성질을 가진 activation function을 각 선형 결합 후에 적용해 모델이 더 다양한 표현력을 가지도록 함

$$O = W_L U_{L-1} = W_L W_{L-1} U_{L-2} = \cdots = W_L W_{L-1} \cdots W_2 W_1 X = WX$$

$$\downarrow$$

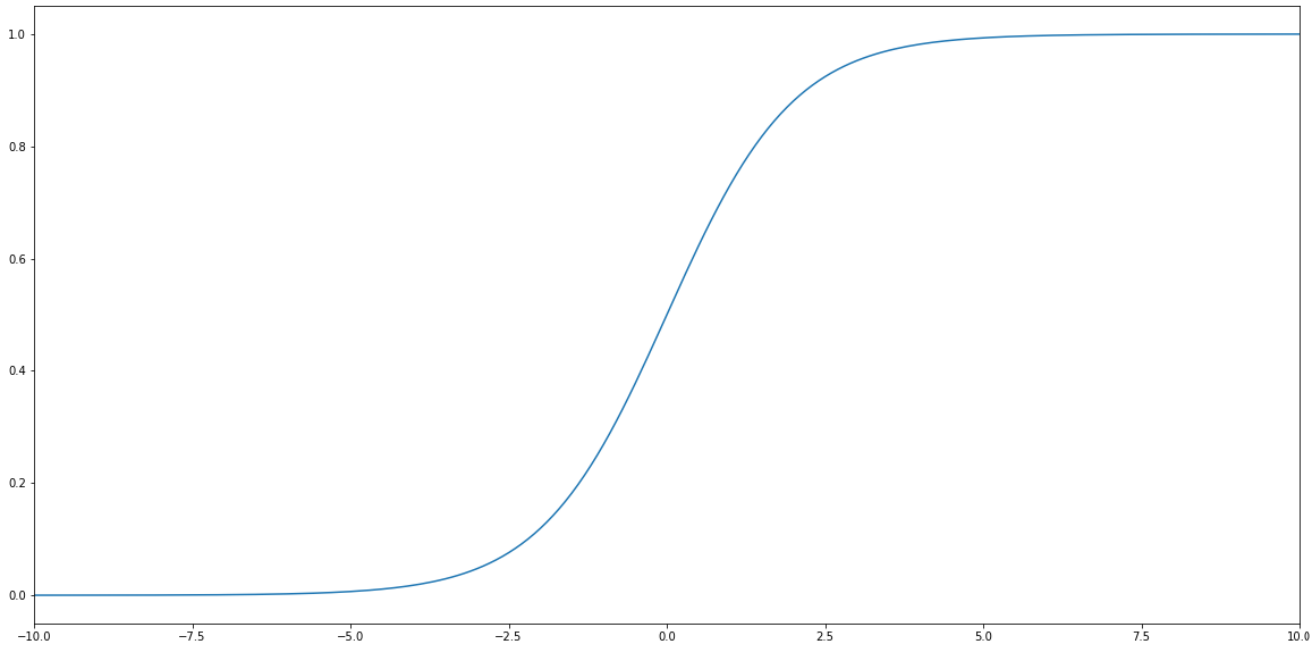
$$O = a(W_L U_{L-1}) = a(W_L a(W_{L-1} U_{L-2})) = \cdots = a(W_L a(W_{L-1} \cdots a(W_2 a(W_1 X) \cdots) \neq WX$$

Activation Functions

- 비선형 성질을 가지고 있는 함수라면 어떤 함수라도 activation function으로 사용 가능
  - 그렇지만 딥러닝 모델의 학습에 유용한 특징들 존재
    - 연산이 빠르고,
    - 학습을 위한 미분 계수의 계산이 빠르고,
    - 데이터의 분포에서 의미 있게 작동
  - Sigmoid, tanh, ReLU, Leaky-ReLU, etc.

LIKE LION

Sigmoid Function

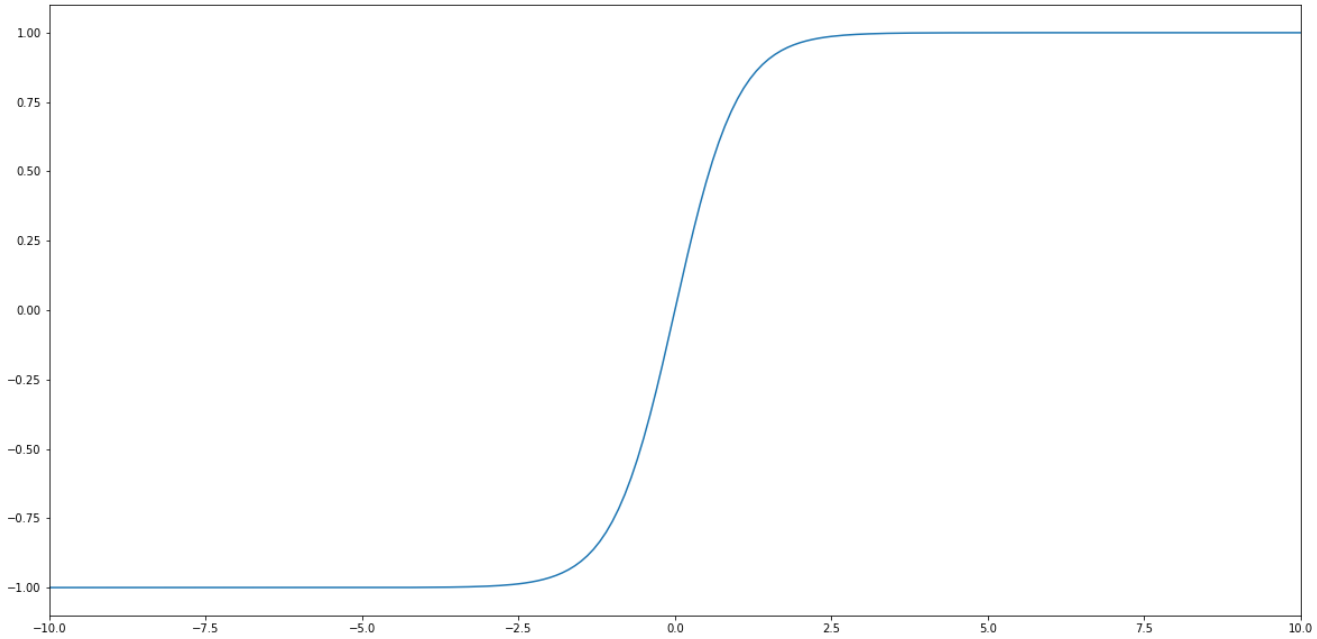


$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

- 전통적으로 많이 쓰이던 activation function
- 미분 계수의 계산이 간단
  - 그렇지만 미분 계수의 값이 0이 되는 영역이 너무 넓음

LIKE LION

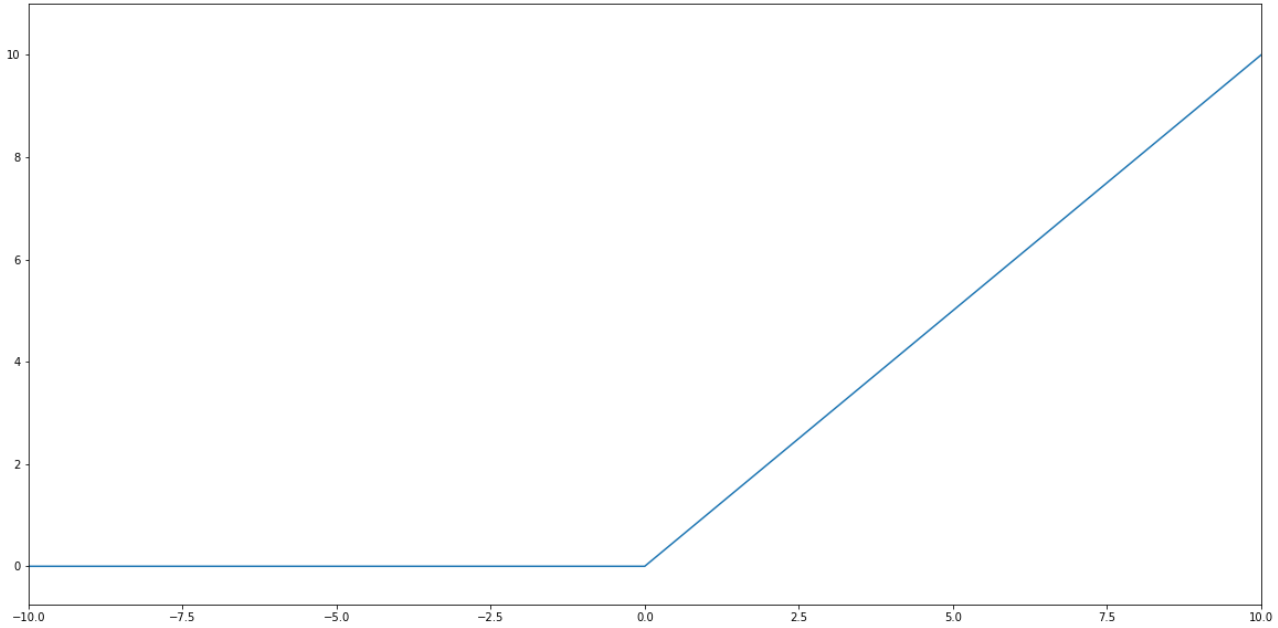
tanh Function



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

- Sigmoid와 굉장히 비슷한 activation function
- 함수 값의 범위를 음수까지 확장

ReLU Function

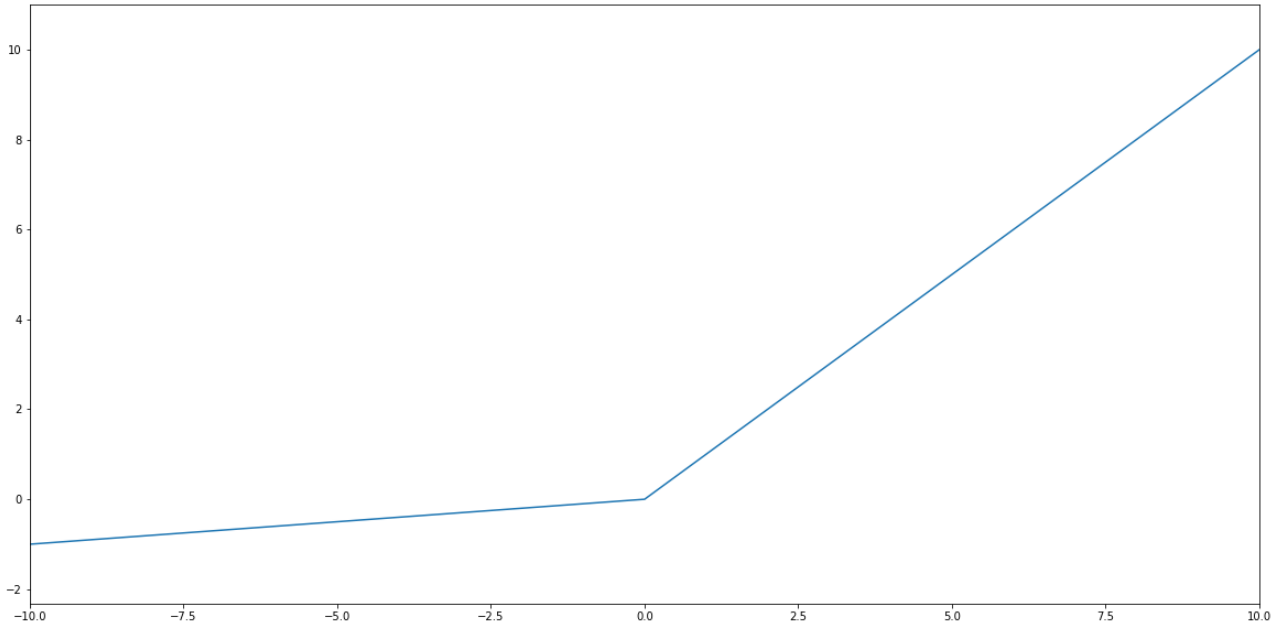


$$f(x) = \max(0, x)$$

- Sigmoid의 기울기가 사라지는 현상을 막은 activation function
- 계산이 굉장히 간단하고 빠름

LIKE LION

Leaky-ReLU Function



$$f(x) = \max(\alpha x, x), \alpha < 0$$

- ReLU의 값이 음수에서 사라지는 문제를 해결한 activation function

Softmax Function

- 모델 output을 확률 분포의 형태로 만들어주는 함수
  - 각 output의 확률을 알려주기 때문에 classification task에서 마지막 layer의 activation function으로 사용됨
  - 모델이 어떤 class로 추정했는지 직관적으로 이해하는데 도움을 줌

$$\vec{o} = \begin{bmatrix} o_1 \\ \vdots \\ o_{d_o} \end{bmatrix} \longrightarrow softmax(\vec{o}) = \frac{1}{\sum_{i=1}^{o_{d_o}} e^{o_i}} \begin{bmatrix} e^{o_1} \\ \vdots \\ e^{o_{d_o}} \end{bmatrix}$$

LIKE LION

{ 5. Loss Function은 뭐로 할까요? }

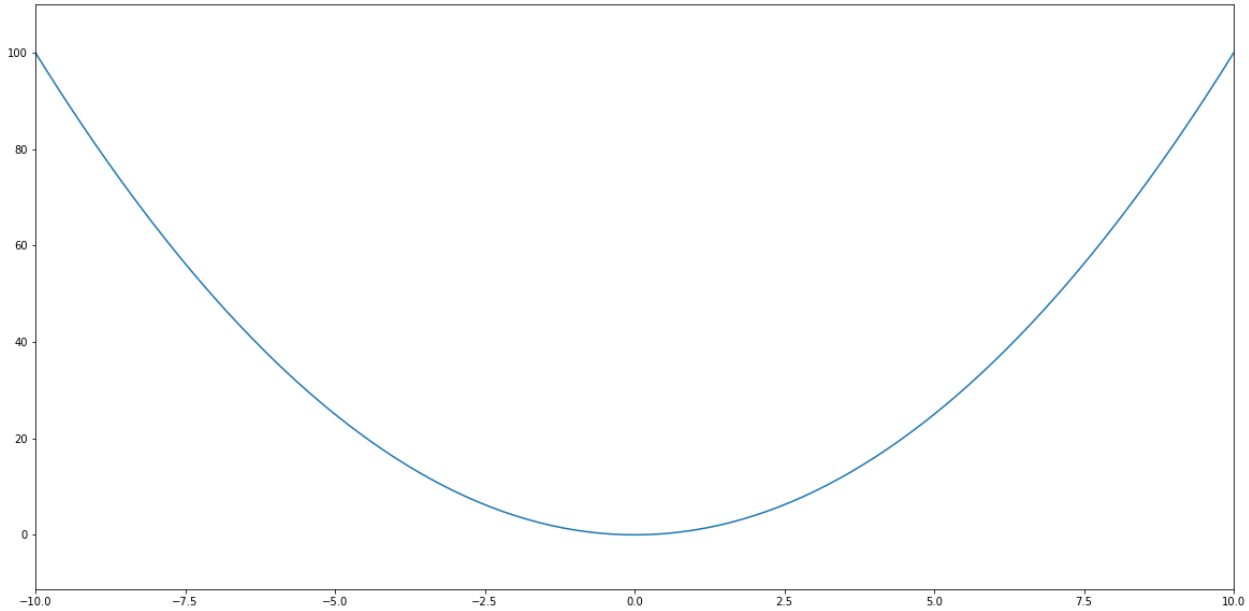
Loss Functions (Cost, Objective)

- 모델의 output이 얼마나 틀렸는지를 나타내는 척도로, 후에 backward pass에서 모델의 parameter를 얼마나 수정할 지 정하는데 사용
- 일반적으로 regression task에서는 MSE loss를, classification task에서는 cross-entropy loss를 사용하지만 각 task의 특징에 따라 다른 함수들 중에서 정하기도 함
    - Backward pass가 loss function을 미분하는 데서 시작하기 때문에 미분 형태가 중요

MSE Loss Function

- 모델의 output과 target 사이의 L2-norm을 계산하여 loss로 사용
  - Loss가 2차 식의 형태이기 때문에, target과의 error가 클수록 loss가 커짐

$$L = \sum_{i=1}^{b} l_i = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$



Cross-Entropy Loss Function

- 모델의 output을 class label에 대한 확률 분포로 가정하고 true label과의 차이를 측정하는 loss function
  - Classification task에서 주로 사용되며 softmax function과 결합하여 간단하게 계산이 됨

$$L = \sum_{i=1}^{n} l_i = -\sum_{i=1}^{n} y_i \log \hat{y}_i = -\log \hat{y}_c = -\log \frac{e^{o_c}}{\sum_{i=1} e^{o_i}} = -o_c + \log \sum_{i=1} e^{o_i}$$

Summary

- 선형 모델인 single-layer perceptron의 표현력 한계를 극복하기 위해 여러 층의 layer를 쌓아 만든 multi-layer perceptron MLP 등장
- 간단한 행렬 곱과 비선형 activation function의 결합으로 이전의 선형 모델보다 훨씬 어려운 문제를 해결할 수 있는 모델 탄생
- Activation function은 Sigmoid와 ReLU 계열의 함수들이 대표적이며 최근에는 ReLU가 가장 대표적으로 사용된다
- Loss function은 해결하려는 task에 맞게 적당한 함수를 선택하며 일반적으로 regression은 MSE loss, classification은 cross-entropy loss를 사용한다

LIKE LION

{

6. (실습) PyTorch Tutorial

7. (실습) MLP MNIST classification (1)

}