Artificial Neural Network

Jehyuk Lee

Department of Data Science

Kookmin University

국민대학교

Contents

1. What is Artificial Neural Network?

Artificial Neural Network

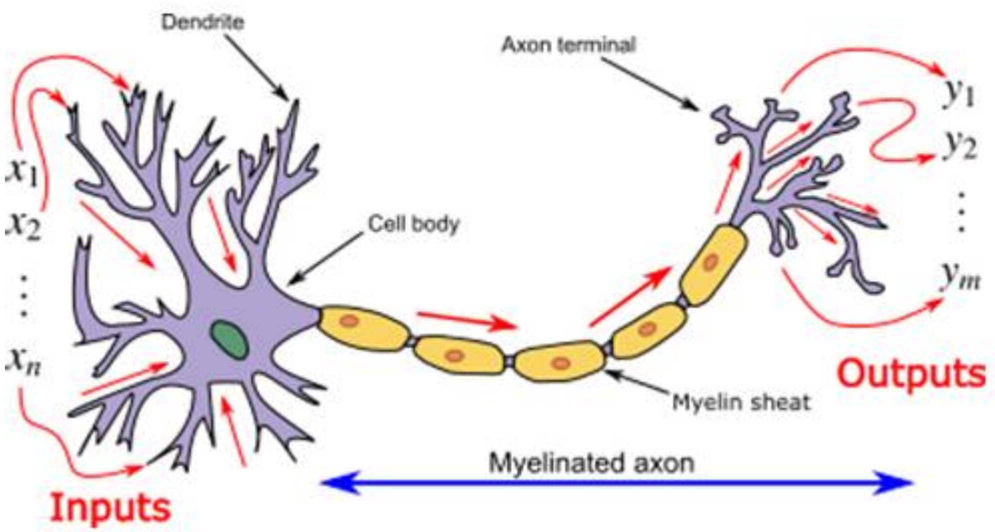
- **생물학적 신경망에서 영감을 얻은 통계학적 학습 알고리즘**(From wiki)
  - 명확하게 정의하기는 어렵다.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives signals then processes them and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called *edges*. Neurons and edges typically have a *weight* that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold.

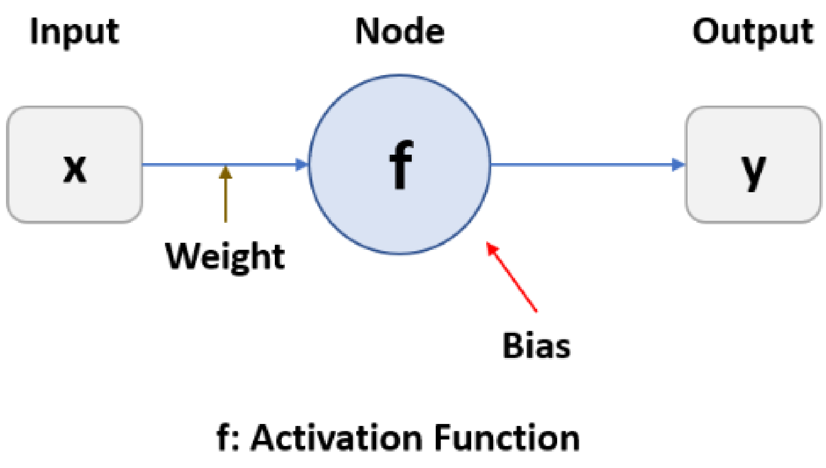
(출처: https://en.wikipedia.org/wiki/Artificial_neural_network)

Artificial Neural Network

- **Neuron(신경망 vs 인공신경망)**
  - **신경망의 뉴런:**
    - 시냅스를 통하는 전기신호를 이용하여 뉴런간 의사소통
  - **인공신경망의 뉴런**
    - 이전 층 뉴런의 결과값을 다음 층의 뉴런에 전달하여 뉴런간 의사소통



신경망의 뉴런



인공신경망의 뉴런

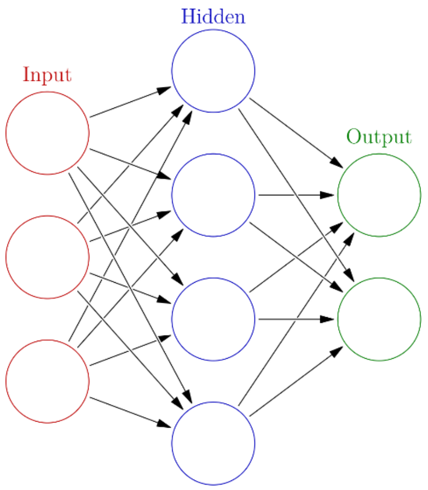
(Source: https://brunch.co.kr/@gdhan/6)

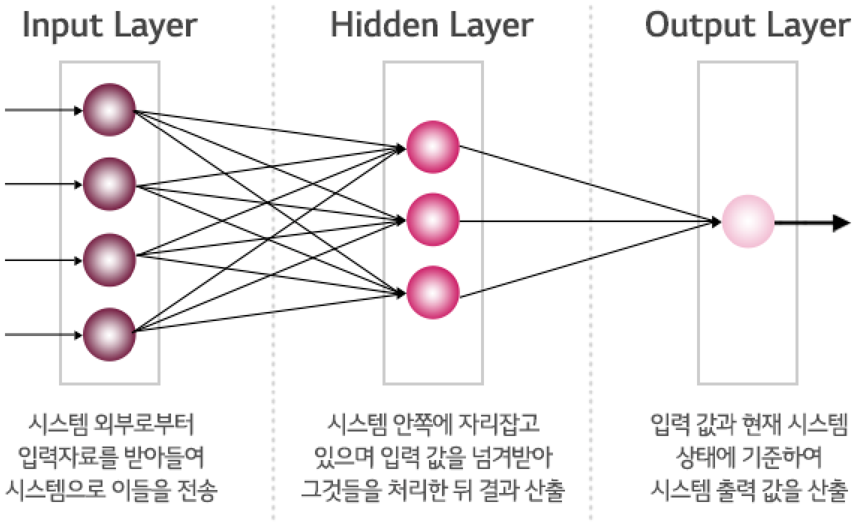
Artificial Neural Network

- **Artificial Neural Network & Deep Learning**
  - 일반적으로 입력층, 한 개 이상의 은닉층, 출력층으로 구성
  - 이 때, Deep Learning은 은닉층이 2개 이상인 neural network를 일컫는다

The "deep" in deep learning is referring to the depth of layers in a neural network. A neural network that consists of more than three layers—which would be inclusive of the inputs and the output—can be considered a deep learning algorithm.

(Source: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks)

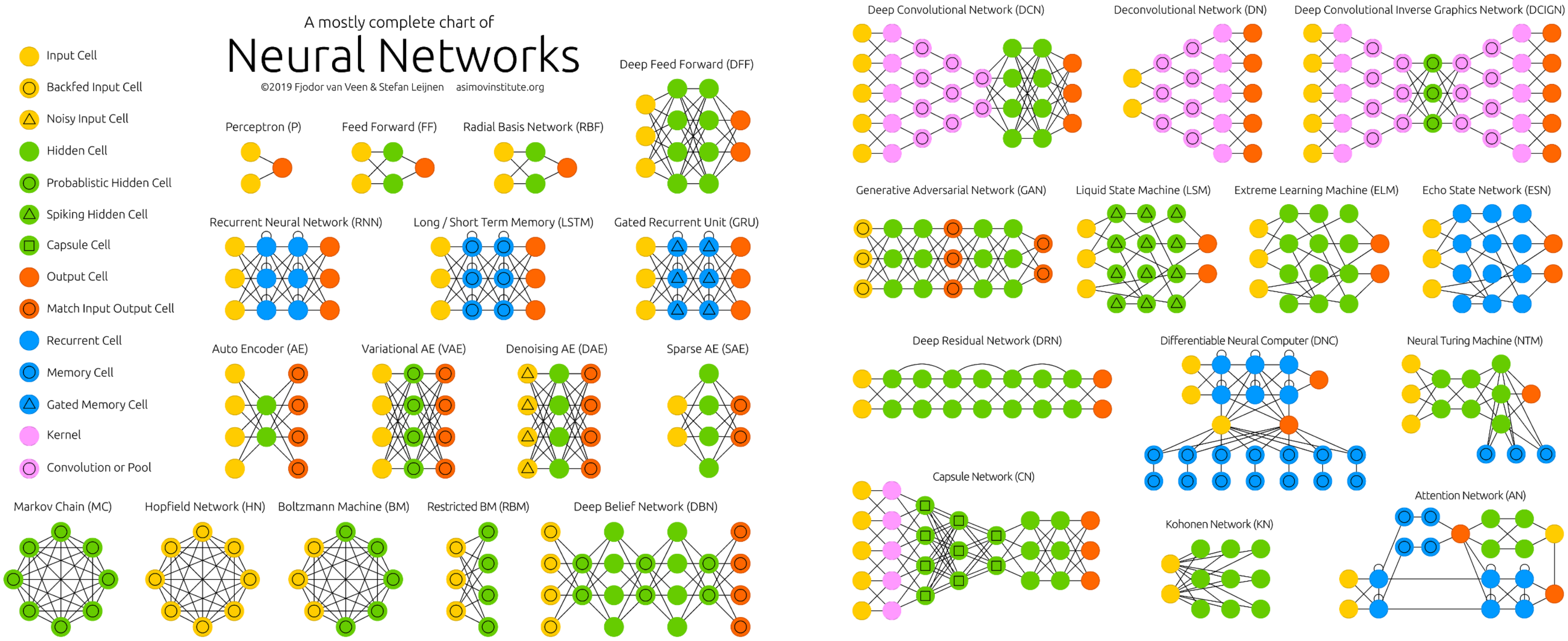




(Source: https://ko.wikipedia.org/wiki/%EC%9D%B8%EA%B3%B5_%EC%8B%A0%EA%B2%BD%EB%A7%9D, https://blog.lgcns.com/1359)

Artificial Neural Network

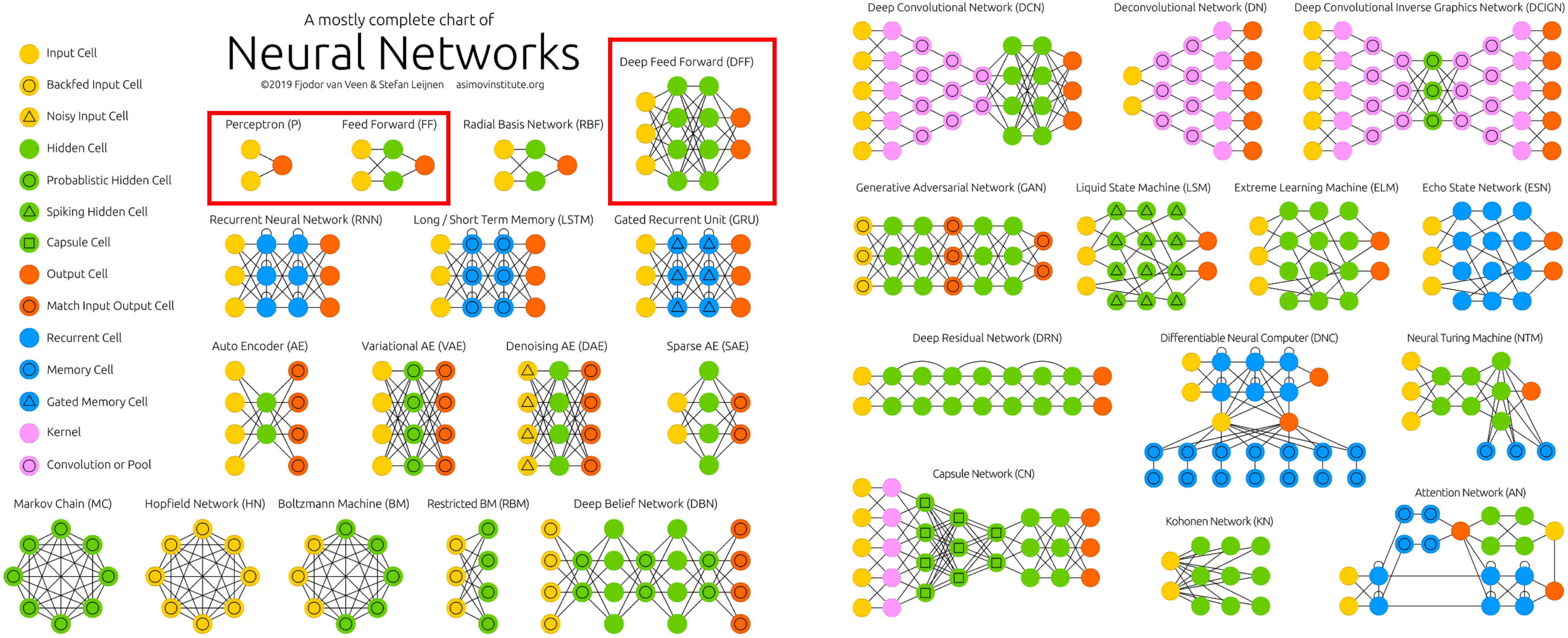
- **Neural network zoo**



(출처: https://www.asimovinstitute.org/neural-network-zoo/)

Artificial Neural Network

- **Neural network zoo**



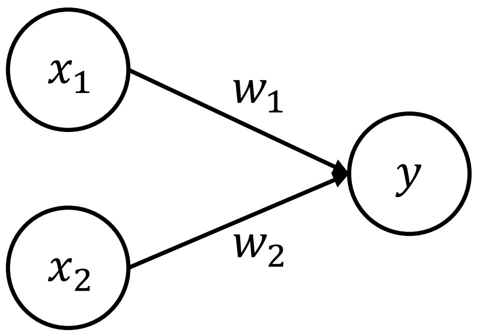
(출처: https://www.asimovinstitute.org/neural-network-zoo/)

2. Perceptron Basic

Perceptron

- **Perceptron**
  - 다수의 신호를 입력받아 하나의 신호를 출력하는 알고리즘
    - 신호는 0 or 1의 값을 갖는다
    - (앞으로 0/1은 신호가 흐르지 않음/신호가 흐름 의 의미로 사용)

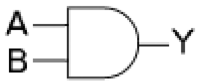
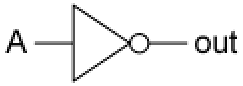
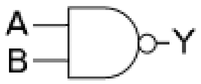
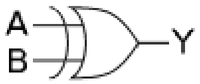
  - 왜 갑자기???
    - Perceptron은 신경망의 기원이 되는 알고리즘
    - **Perceptron을 사용하면 복잡한 함수도 근사하게(approximately)하게 표현 가능**



(신호가 2개인 Perceptron 예시)

Perceptron

- **Perceptron으로 논리 게이트 나타내보기**
  - 논리회로
    - 1개 이상의 논리 입력을 일정한 논리 연산에 의해 논리 출력을 얻는 회로

<table>
<tr><th>논리</th><th>AND</th><th>OR</th><th>NOT</th><th>NAND</th><th>NOR</th><th>XOR</th></tr>
<tr><td>회로 기호</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td></tr>
<tr><td>진리표</td>
<td><table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td>
<td><table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td>
<td><table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td>
<td><table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td>
<td><table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td>
<td><table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
</table>

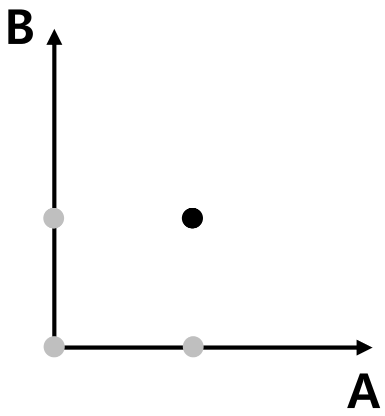
(Source: https://ko.wikipedia.org/wiki/%EB%85%BC%EB%A6%AC_%ED%9A%8C%EB%A1%9C)

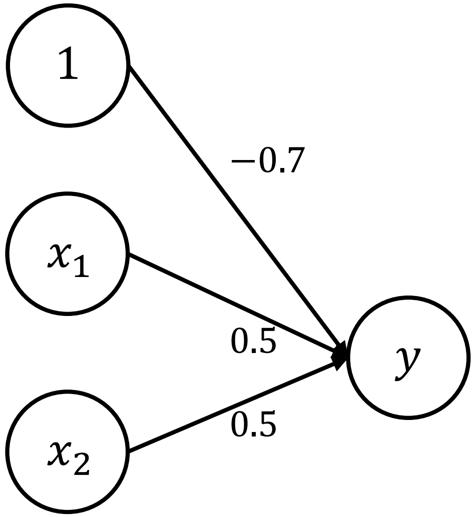
Perceptron

- **Perceptron으로 논리 게이트 나타내보기**
  - AND 게이트
    - 모든 입력 값이 1일 때, 출력값이 1, 그렇지 않으면 0
  - 이러한 논리회로는 다양한 가중치 조합의 perceptron으로 표현 가능
    - $(w_1, w_2, b) = (0.5, 0.5, -0.7), (1.0, 1.0, -1.1), \ldots$

$$y = \begin{cases} 0, b + w_1x_1 + w_2x_2 \le 0 \\ 1, b + w_1x_1 + w_2x_2 > 0 \end{cases}$$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1





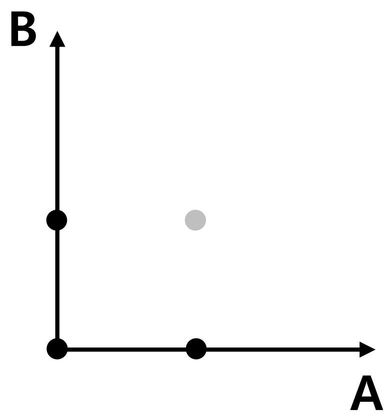
(신호가 2개인 Perceptron으로 AND게이트 표현)

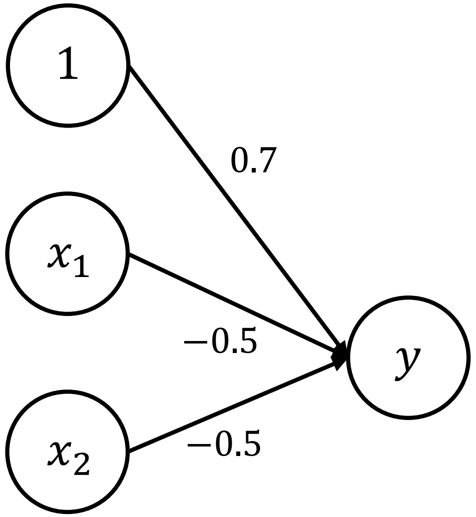
Perceptron

- **Perceptron으로 논리 게이트 나타내보기**
  - NAND 게이트 (Not AND)
    - AND 게이트의 출력을 뒤집은 출력값
  - 이러한 논리회로는 다양한 가중치 조합의 perceptron으로 표현 가능
    - $(w_1, w_2, b) = (-0.5, -0.5, 0.7), (-1.0, -1.0, 1.1), \ldots$

$$y = \begin{cases} 0, b + w_1x_1 + w_2x_2 \le 0 \\ 1, b + w_1x_1 + w_2x_2 > 0 \end{cases}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0





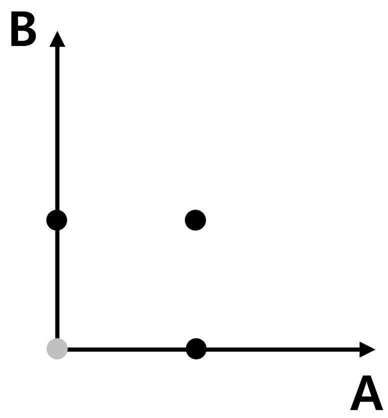
(신호가 2개인 Perceptron으로 NAND게이트 표현)

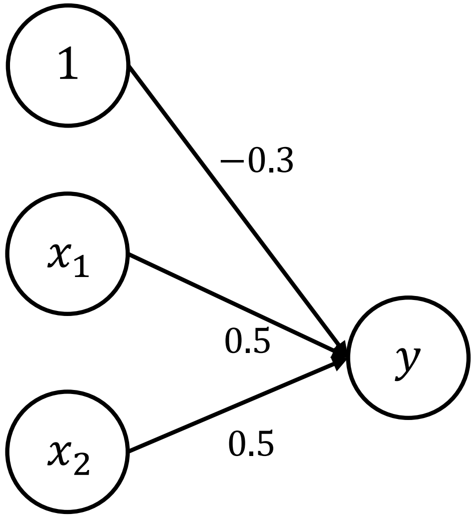
Perceptron

- **Perceptron으로 논리 게이트 나타내보기**
  - OR 게이트
    - 입력값이 적어도 하나가 1이면 출력값이 1, 그렇지 않으면 0
  - 이러한 논리회로는 다양한 가중치 조합의 perceptron으로 표현 가능
    - $(w_1, w_2, b) = (0.5, 0.5, -0.3), (1.0, 1.0, -0.9), \ldots$

$$y = \begin{cases} 0, b + w_1x_1 + w_2x_2 \le 0 \\ 1, b + w_1x_1 + w_2x_2 > 0 \end{cases}$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



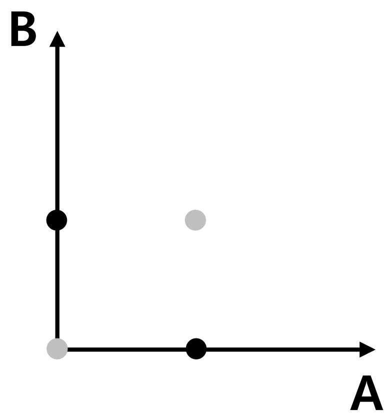


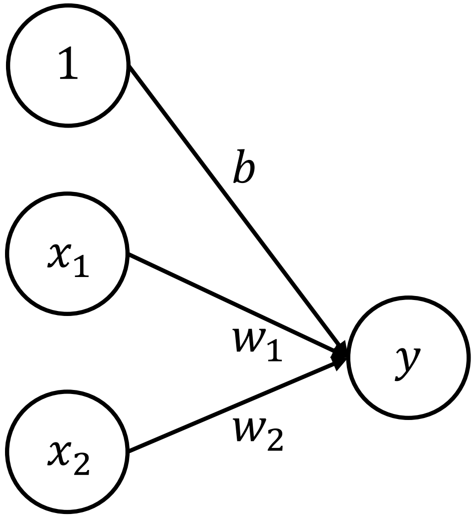
(신호가 2개인 Perceptron으로 OR게이트 표현)

Perceptron

- **Perceptron으로 논리 게이트 나타내보기**
  - XOR 게이트
    - 하나의 입력값만 1이면 출력값이 1, 그렇지 않으면 출력값이 0
  - Perceptron으로 XOR 게이트를 나타낼 수 있을까?

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0





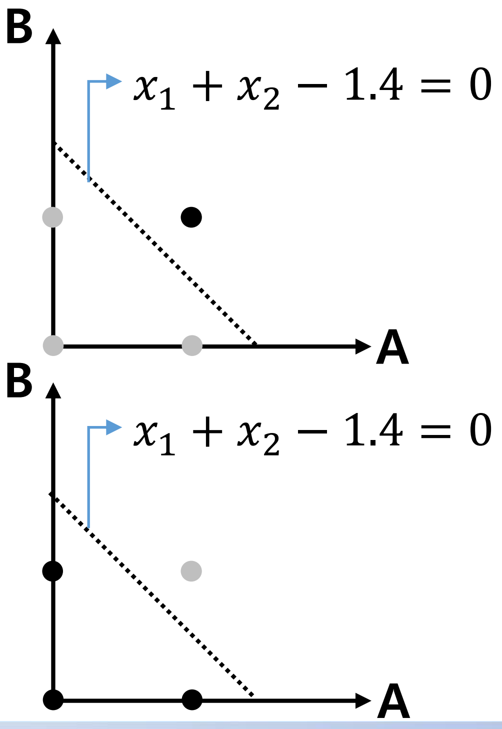
(신호가 2개인 Perceptron 예시)

Perceptron

- **Perceptron으로 논리 게이트 나타내보기**
  - AND와 NAND를 다시 보자
    - AND의 가중치 조합: $(w_1, w_2, b) = (0.5, 0.5, -0.7), \ldots$
    - NAND의 가중치 조합: $(w_1, w_2, b) = (-0.5, -0.5, 0.7), \ldots$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



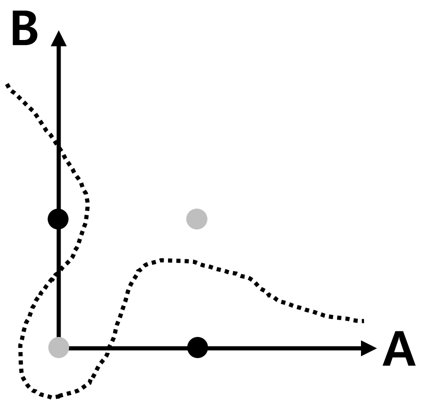
Perceptron은 선형 분류기!

Perceptron

- **Perceptron으로 논리 게이트 나타내보기**
  - XOR 게이트
    - 하나의 입력값만 1이면 출력값이 1, 그렇지 않으면 출력값이 0
  - Perceptron으로 XOR 게이트를 나타낼 수 있을까?
    - Perceptron은 선형 분류기를 나타낸다.
    - XOR은 선형 분류기를 나타낼 수 없다

→ Perceptron으로 XOR를 나타낼 수 없다

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

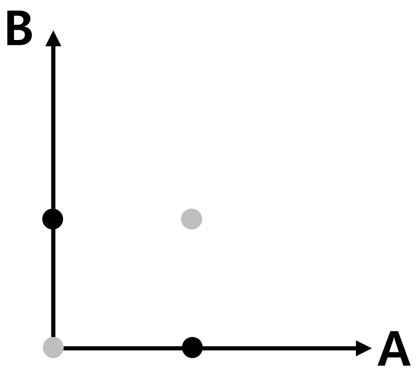


선형 분류기로 XOR의 논리표를 나타낼 수 없음

Perceptron

- **Multilayer Perceptron**
  - 단층 (층이 1개) Perceptron으로 XOR 게이트를 나타내는 건 불가능
    - 하지만 다층 Perceptron이 출동하면 어떨까?
  - 다층 Perceptron으로 비선형 분류기를 나타낼 수 있을까?

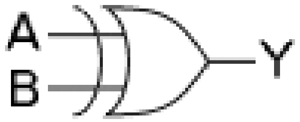
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



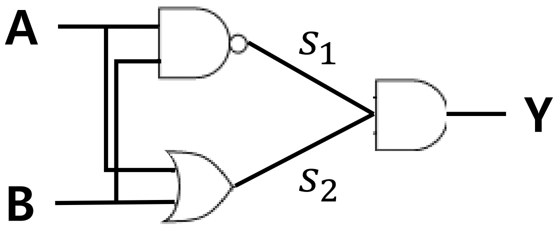
선형 분류기로 XOR의 논리표를 나타낼 수 없음

Perceptron

- **Multilayer Perceptron**
  - XOR게이트는 AND, NAND, OR 게이트를 조합하여 만들 수 있다.



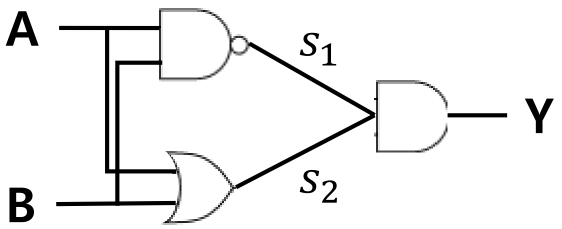
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



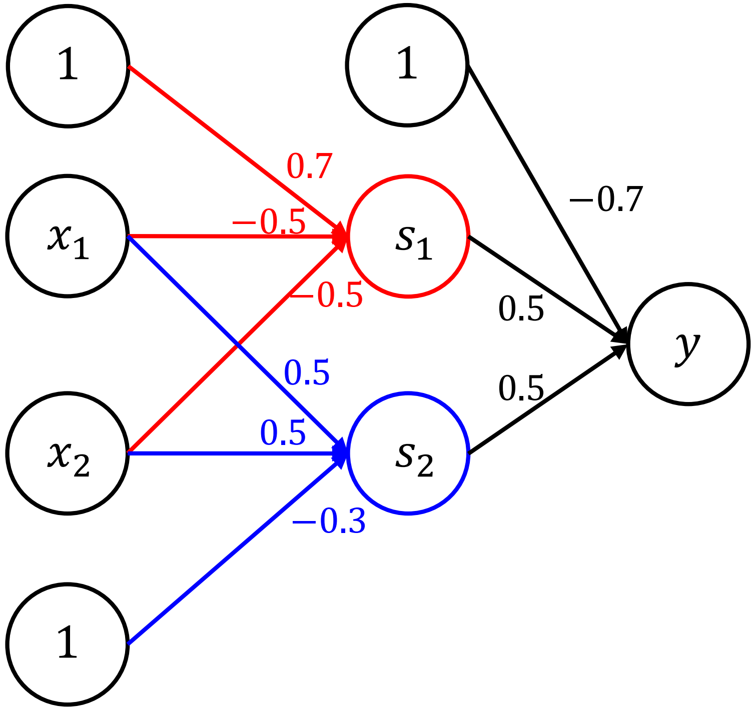
A	B	s_1	s_2	Y
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

Perceptron

- **Multilayer Perceptron**
  – 즉, 다음과 같이 XOR를 나타내는 perceptron을 만들 수 있다.
    - 0층에서 입력신호를 받아 AND, NAND를 나타내는 perceptron의 출력값을 1층으로 보냄
    - 전달받은 결과를 OR를 나타내는 perceptron에 입력하여 출력값을 산출



A	B	s_1	s_2	Y
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0



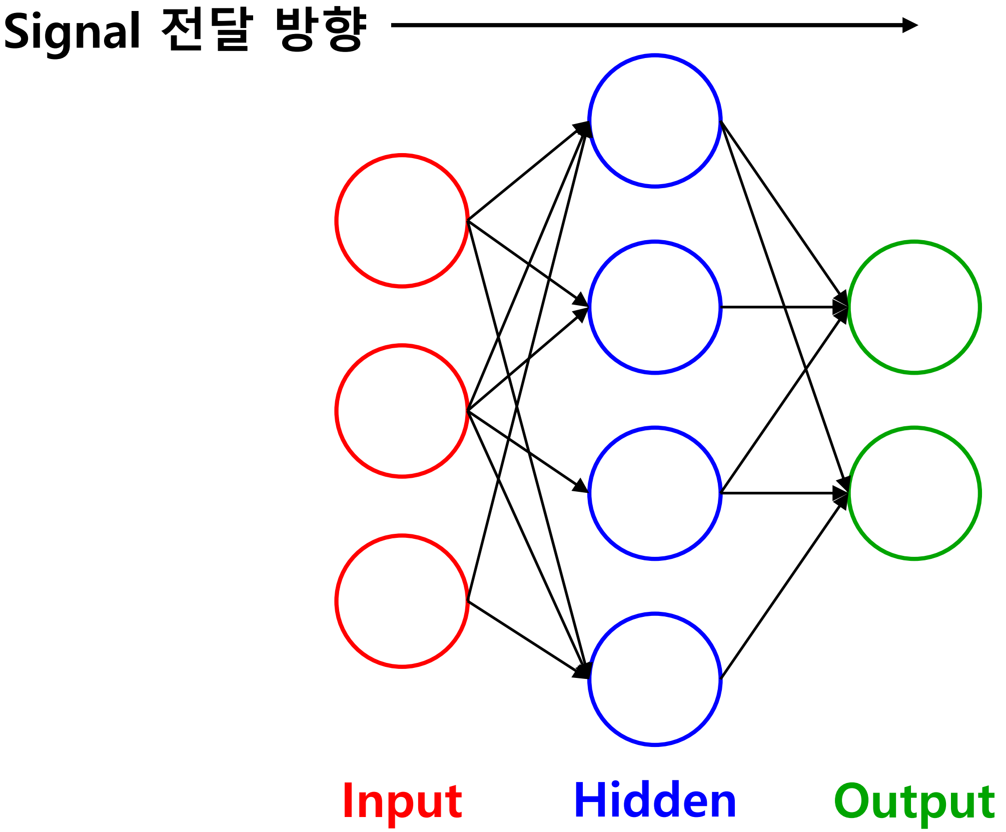
$$y = \begin{cases} 0, b + w_1x_1 + w_2x_2 \le 0 \\ 1, b + w_1x_1 + w_2x_2 > 0 \end{cases}$$

x_1	x_2	s_1	s_2	y
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

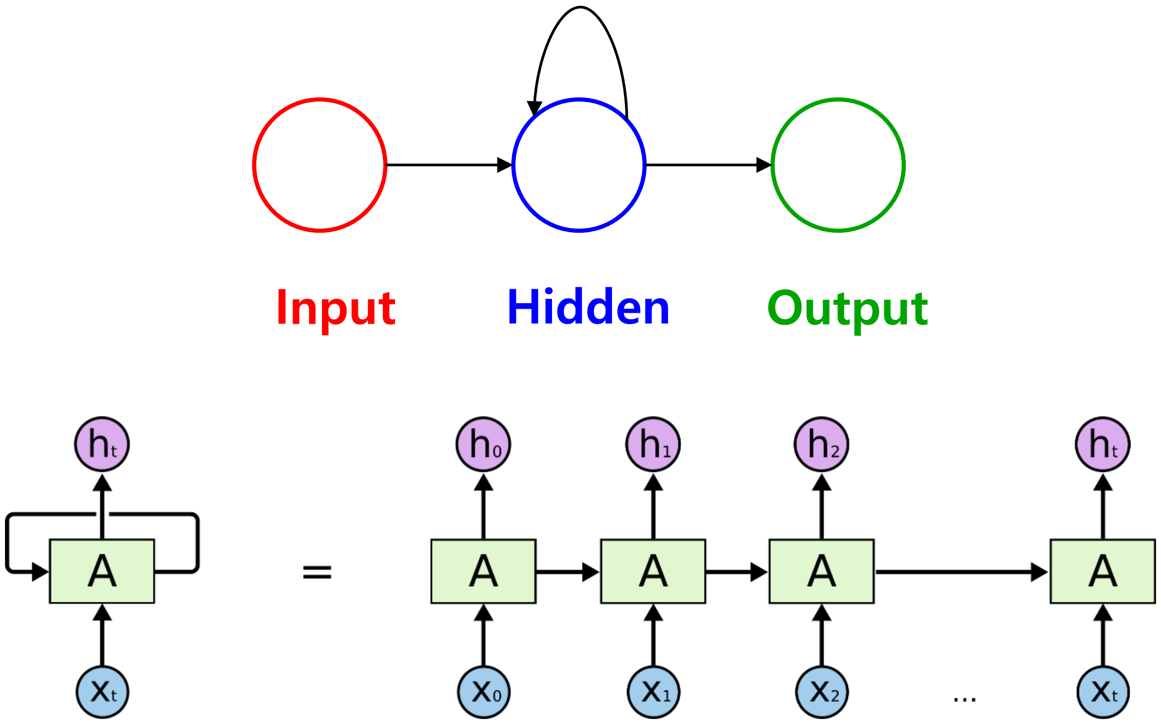
3. Neural Network Basic

Feed Forward Neural Network (FFNN)

- **Signal이 한쪽 방향으로만 흐르는 인공신경망의 일종 (input → output)**
  - Neuron이 signal을 입력받으면 연산을 진행한 뒤, 다음 층의 neuron으로 결과를 전달



Feed Forward Neural Network

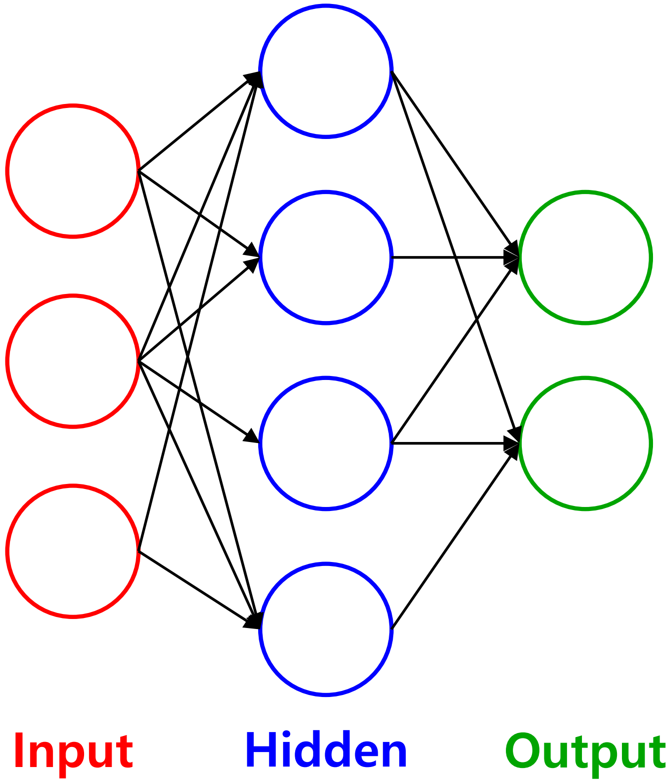


Recurrent Neural Network

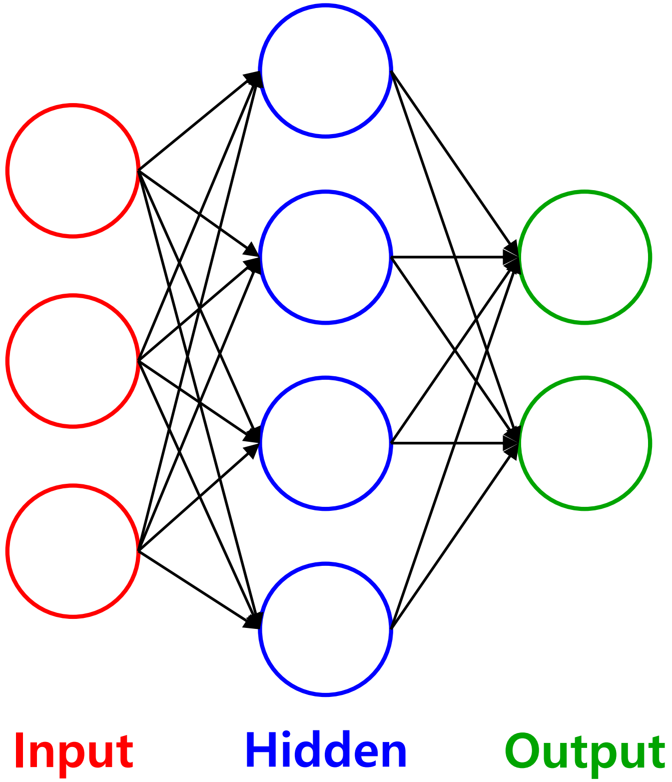
(출처: https://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Fully Connected Layer

- **Layer의 모든 뉴런이 다음 layer의 뉴런과 전부 연결된 layer**
  - Dense layer라고 부르기도 한다.



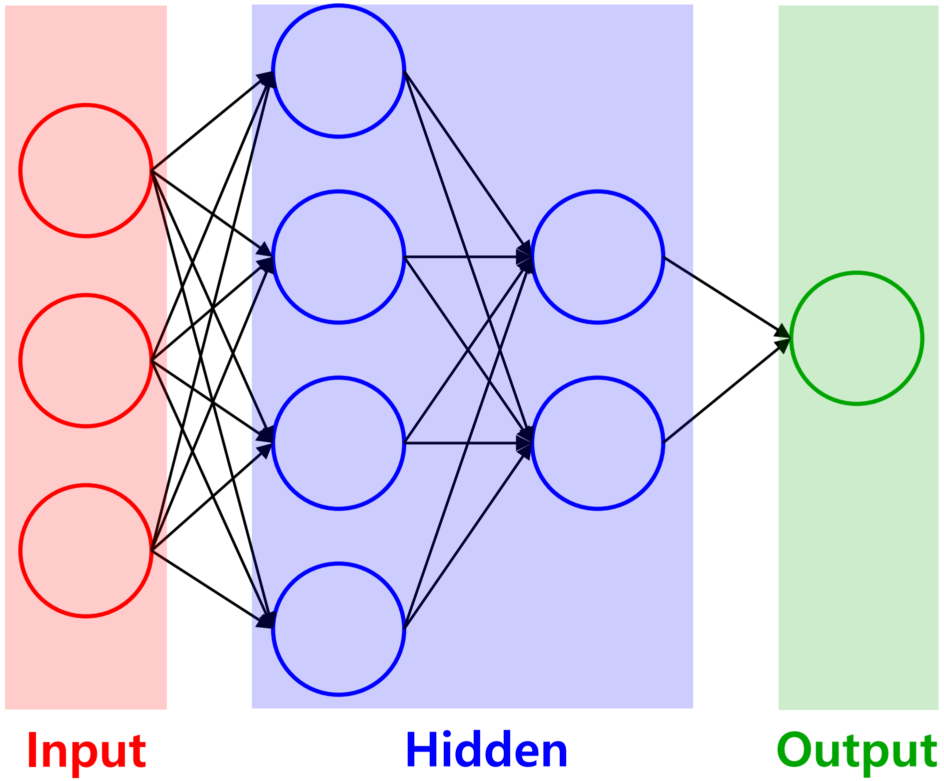
Hidden layer is not fully connected



Hidden layer is fully connected

Multi-layer Perceptron (MLP)

- **MLP는 다음과 같이 구성**
  - 입력층(Input layer): 데이터를 입력받는 층 (0층)
  - 출력층(Output layer): 최종 출력값이 존재하는 층 (1층, 2층,…)
  - 은닉층(Hidden layer): 입력층과 출력층 사이에 있는 모든 층



→ **Three-layer FFNN**

Activation Function

- **Perceptron**
  - 다수의 신호를 입력받아 하나의 신호를 출력하는 알고리즘
    - 입력값에 가중치를 곱한 값이 특정 임계값(θ, threshold)를 넘어서면 y에 신호가 흐름

$$y = \begin{cases} 0, b + w_1x_1 + w_2x_2 \le 0 \\ 1, b + w_1x_1 + w_2x_2 > 0 \end{cases}$$

$$\Downarrow$$

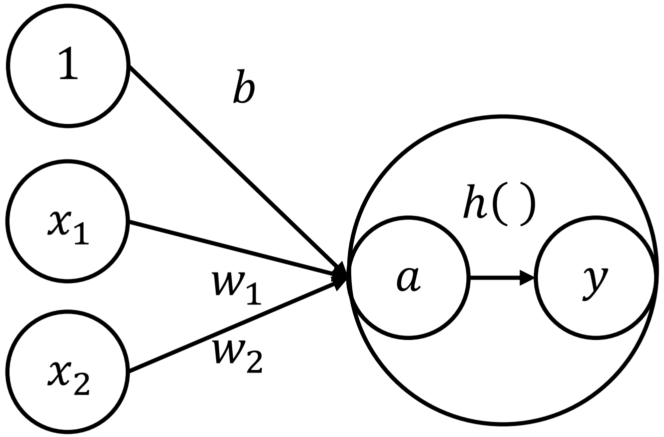
$$y = h(b + w_1x_1 + w_2x_2)$$

$$\boxed{h(a) = \begin{cases} 0(a \le 0) \\ 1(a > 0) \end{cases}}$$

활성 함수 (Activation Function)

Activation Function

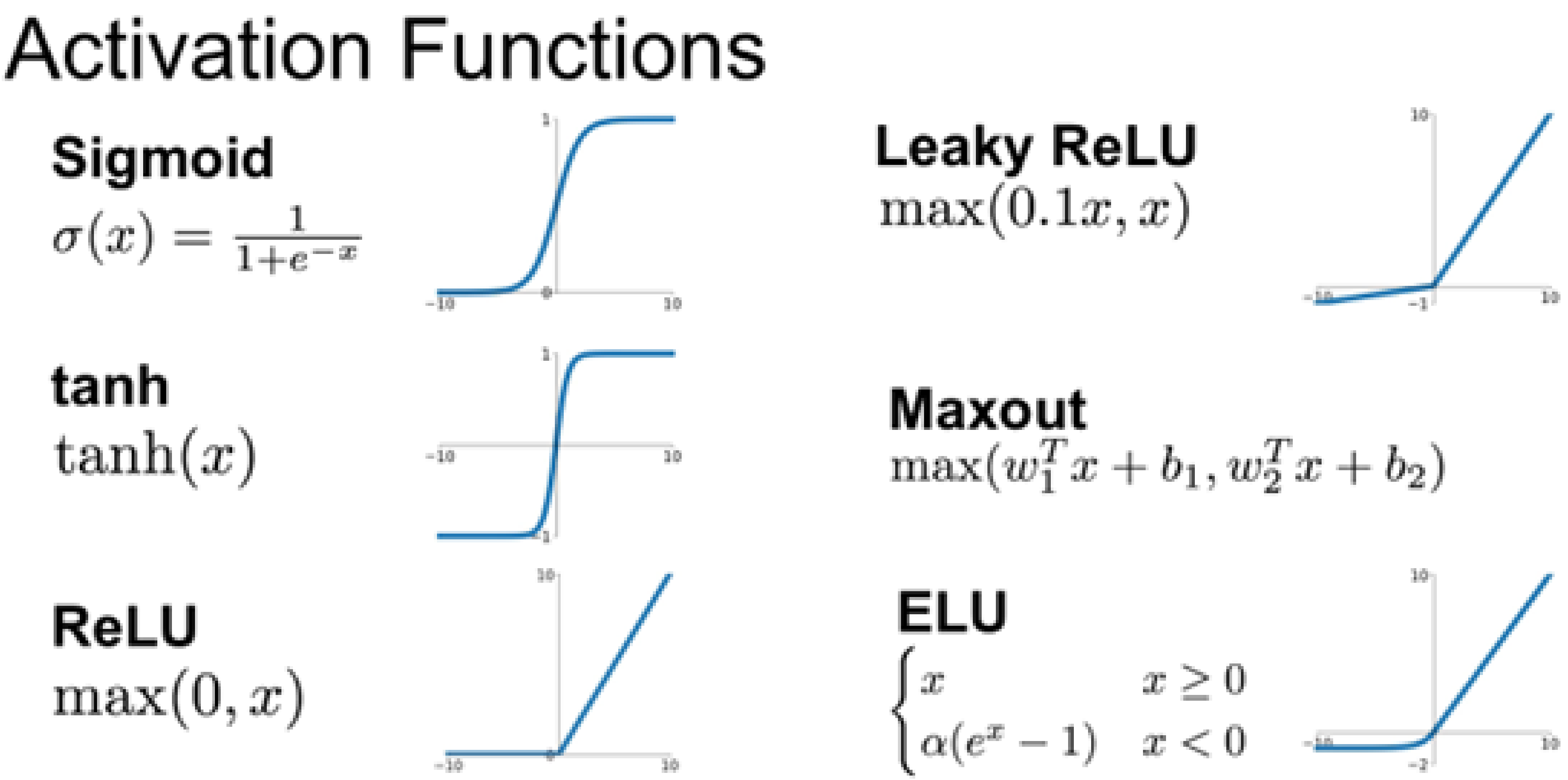
- **뉴런에 들어오는 입력 신호의 총합을 출력 신호로 변환하는 함수**
  - 입력 신호$(x_1, x_2, 1)$에 가중치(w_1, w_2, b)를 곱한 값을 더하고, 이를 a라고 함
  - 위에서 계산된 a를 활성화 함수 $h(\,)$에 넣어 y를 출력



(활성화 함수가 포함된 Perceptron)

Activation Function

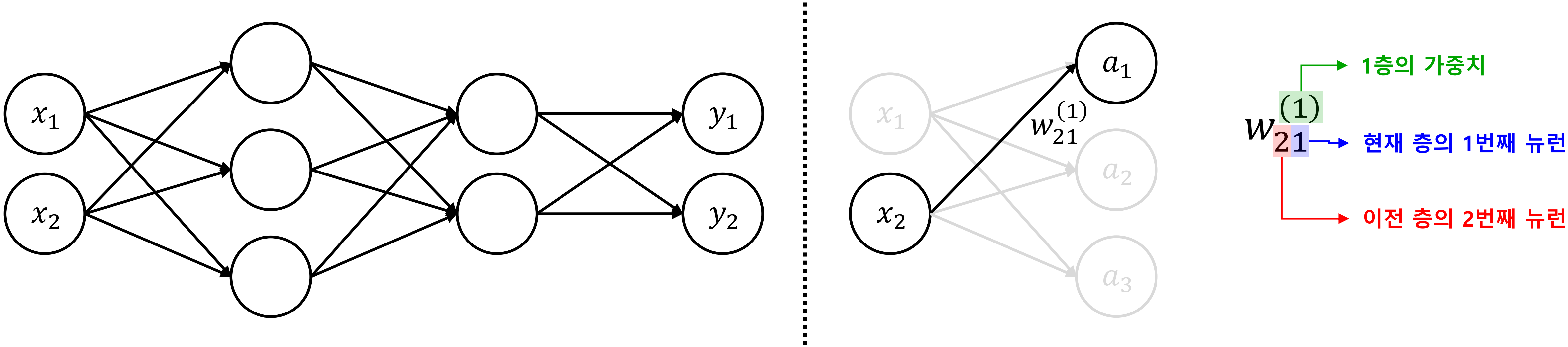
- 다양한 종류의 활성화 함수



(출처: https://heeya-stupidbutstudying.tistory.com/entry/ML-%ED%99%9C%EC%84%B1%ED%99%94-%ED%95%A8%EC%88%98Activation-Function)

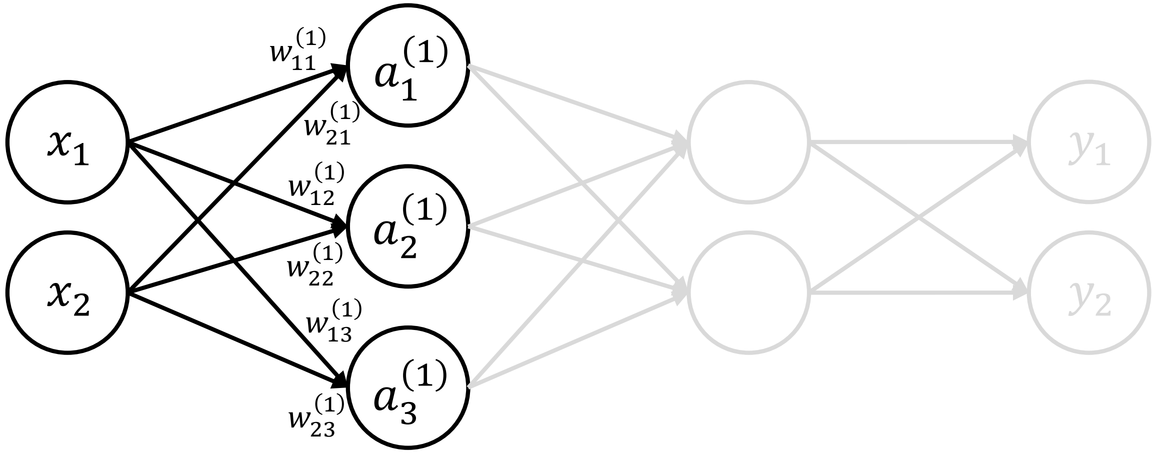
Forward Propagation

- **Neural network에 데이터를 입력하였을 때, 출력값이 나오는 과정**
  - Example: 3층 신경망의 forward propagation



Forward Propagation

- **Example: 3층 신경망의 forward propagation**
  - (1) Input Layer → 1st Hidden Layer

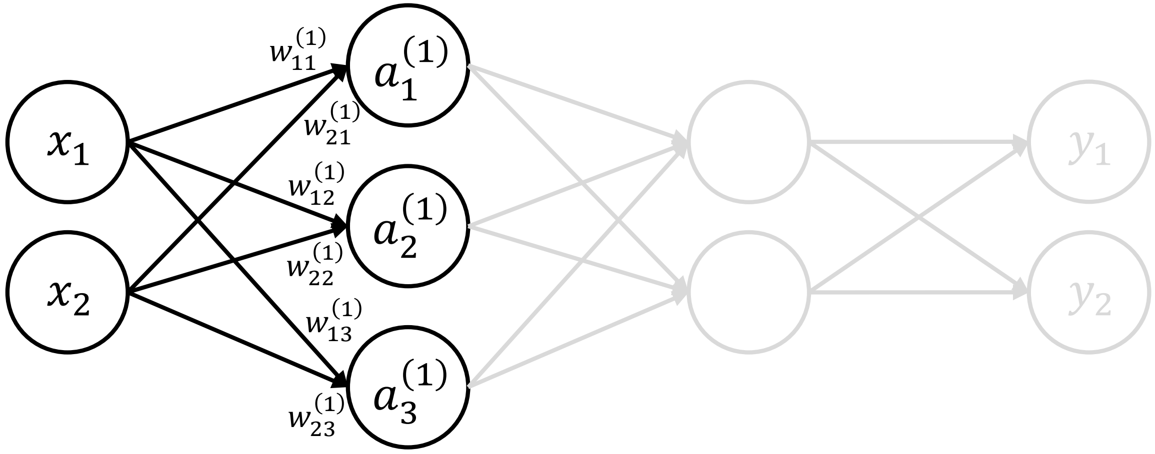


$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + b_1^{(1)}$$
$$a_2^{(1)} = w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + b_2^{(1)}$$
$$a_3^{(1)} = w_{13}^{(1)}x_1 + w_{23}^{(1)}x_2 + b_3^{(1)}$$

$$\begin{bmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}$$
$$+ \begin{bmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{bmatrix}$$

Forward Propagation

- **Example: 3층 신경망의 forward propagation**
  - (1) Input Layer → 1st Hidden Layer



$$a_1^{(1)} = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + b_1^{(1)}$$
$$a_2^{(1)} = w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}$$
$$a_3^{(1)} = w_{13}^{(1)} x_1 + w_{23}^{(1)} x_2 + b_3^{(1)}$$

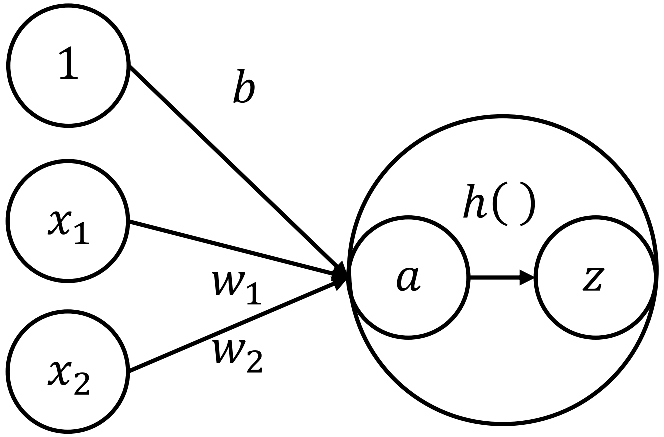
$$\begin{bmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{bmatrix}$$

$$\boldsymbol{A}^{(1)} = \boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{B}^{(1)}$$

Forward Propagation

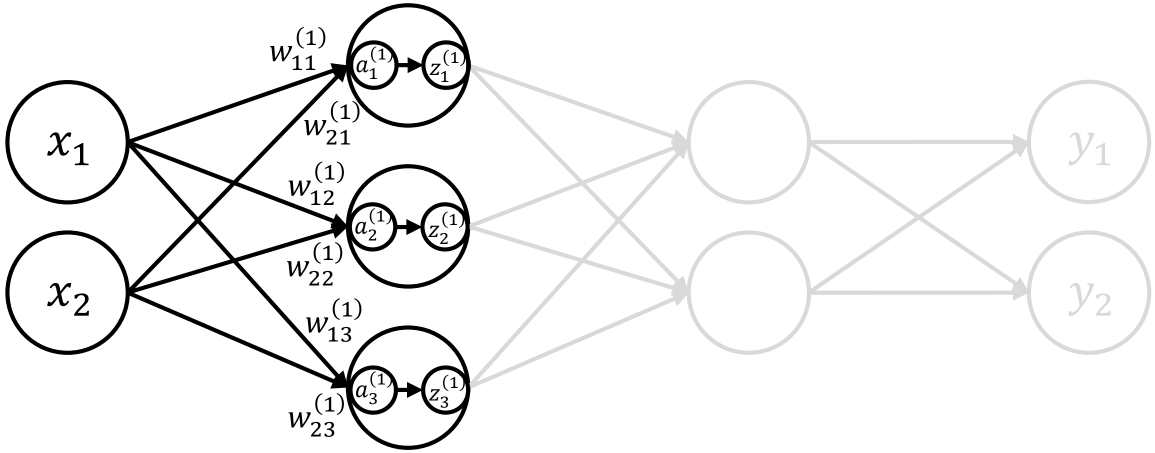
- **Activation Function**
  - 입력 신호($x_1, x_2, 1$)에 가중치(w_1, w_2, b)를 곱한 값을 더하고, 이를 a라고 함
  - 위에서 계산된 a를 활성화 함수 $h(\,)$에 넣어 z를 출력



(활성화 함수가 포함된 Perceptron)

Forward Propagation

- **Example: 3층 신경망의 forward propagation**
  - (1) Input Layer → 1st Hidden Layer
    - With Activation function



$$\begin{bmatrix} a_1^{(1)} & a_2^{(1)} & a_3^{(1)} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1^{(1)} & b_2^{(1)} & b_3^{(1)} \end{bmatrix}$$

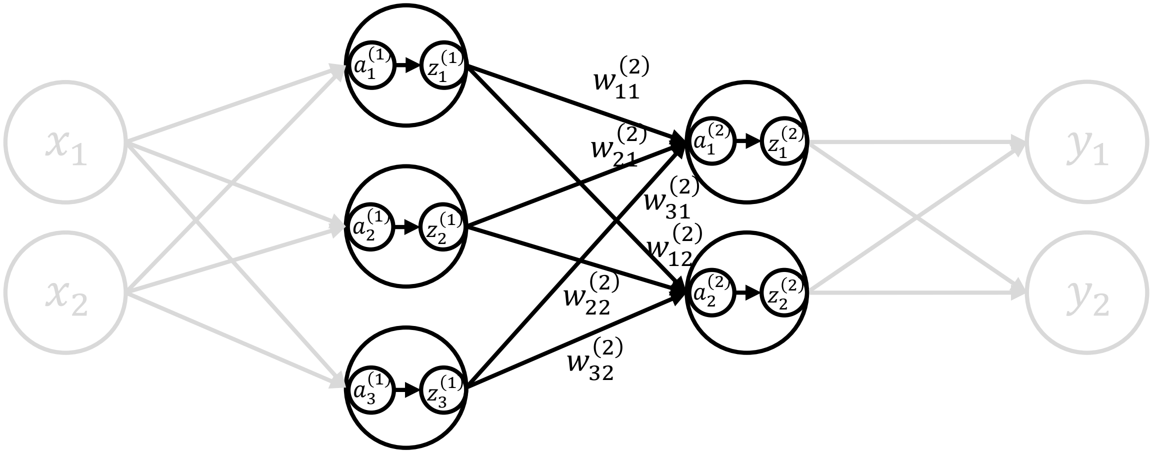
$$\begin{bmatrix} z_1^{(1)} & z_2^{(1)} & z_3^{(1)} \end{bmatrix} = \begin{bmatrix} h\left(a_1^{(1)}\right) & h\left(a_2^{(1)}\right) & h\left(a_3^{(1)}\right) \end{bmatrix}$$

$$\boldsymbol{A}^{(1)} = \boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{B}^{(1)}, \ \boldsymbol{Z}^{(1)} = h\left(\boldsymbol{A}^{(1)}\right)$$

Forward Propagation

- **Example: 3층 신경망의 forward propagation**
  - (2) 1st Hidden Layer → 2nd Hidden Layer
    - With Activation function

$$a_1^{(2)} = w_{11}^{(2)} z_1^{(1)} + w_{21}^{(2)} z_2^{(1)} + w_{31}^{(2)} z_2^{(1)} + b_1^{(2)}$$
$$a_2^{(2)} = w_{12}^{(2)} z_1^{(1)} + w_{22}^{(2)} z_2^{(1)} + w_{33}^{(2)} z_2^{(1)} + b_2^{(2)}$$



$$\begin{bmatrix} a_1^{(2)} & a_2^{(2)} \end{bmatrix} = \begin{bmatrix} z_1^{(1)} & z_2^{(1)} & z_3^{(1)} \end{bmatrix} \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} \end{bmatrix}$$
$$+ \begin{bmatrix} b_1^{(2)} & b_2^{(2)} \end{bmatrix}$$

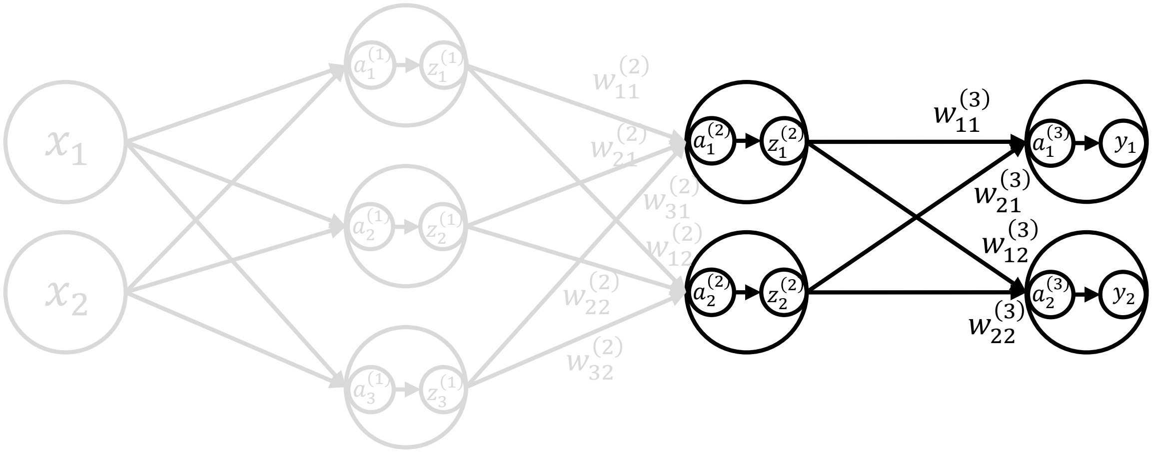
$$\begin{bmatrix} z_1^{(2)} & z_2^{(2)} \end{bmatrix} = \begin{bmatrix} h\left(a_1^{(2)}\right) & h\left(a_2^{(2)}\right) \end{bmatrix}$$

$$\boldsymbol{A}^{(2)} = \boldsymbol{Z}^{(1)} \boldsymbol{W}^{(2)} + \boldsymbol{B}^{(2)}, \ \boldsymbol{Z}^{(2)} = h\left(\boldsymbol{A}^{(2)}\right)$$

Forward Propagation

- **Example: 3층 신경망의 forward propagation**
  - (3) 2nd Hidden Layer → Output Layer

$$y_1 = w_{11}^{(3)} z_1^{(2)} + w_{21}^{(3)} z_2^{(2)} + b_1^{(3)}$$
$$y_2 = w_{12}^{(3)} z_1^{(2)} + w_{22}^{(3)} z_2^{(2)} + b_2^{(3)}$$



$$\begin{bmatrix} a_1^{(3)} & a_2^{(3)} \end{bmatrix} = \begin{bmatrix} z_1^{(2)} & z_2^{(2)} \end{bmatrix} \begin{bmatrix} w_{11}^{(3)} & w_{12}^{(3)} \\ w_{21}^{(3)} & w_{22}^{(3)} \end{bmatrix}$$

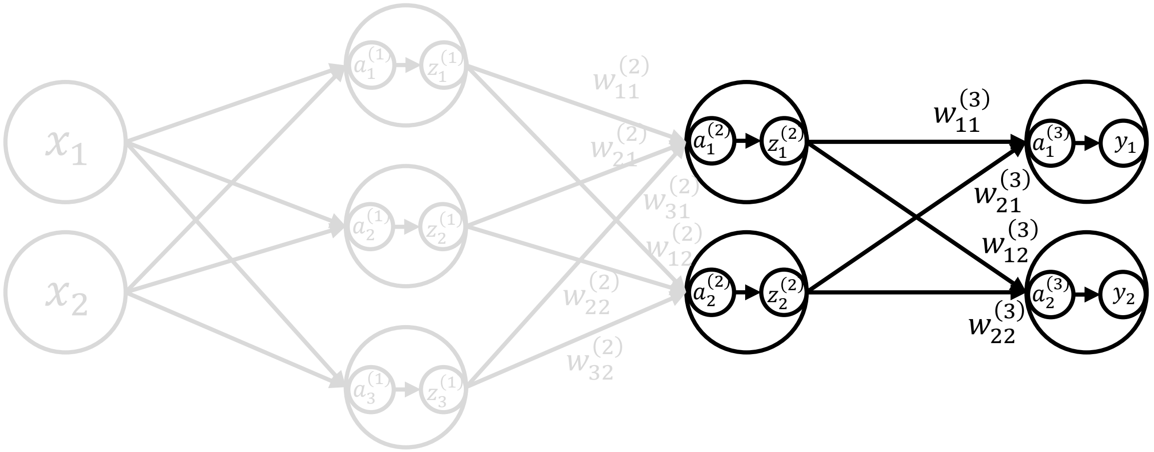
$$+ \begin{bmatrix} b_1^{(3)} & b_2^{(3)} \end{bmatrix}$$

$$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = \begin{bmatrix} h\left(a_1^{(3)}\right) & h\left(a_2^{(3)}\right) \end{bmatrix}$$

$$\boldsymbol{A}^{(3)} = \boldsymbol{Z}^{(2)} \boldsymbol{W}^{(3)} + \boldsymbol{B}^{(3)}, \ \boldsymbol{Y} = h\left(\boldsymbol{A}^{(3)}\right)$$

Forward Propagation

- **Example: 3층 신경망의 forward propagation**
  - (4) Output Layer의 활성화 함수
    - 문제의 목적에 따라 output layer에서 사용하는 활성화 함수가 다르다.
      - 회귀: 항등 활성화함수 (identity activation function)
      - 분류: 소프트맥스 활성화함수 (softmax activation function)



- **Identity**

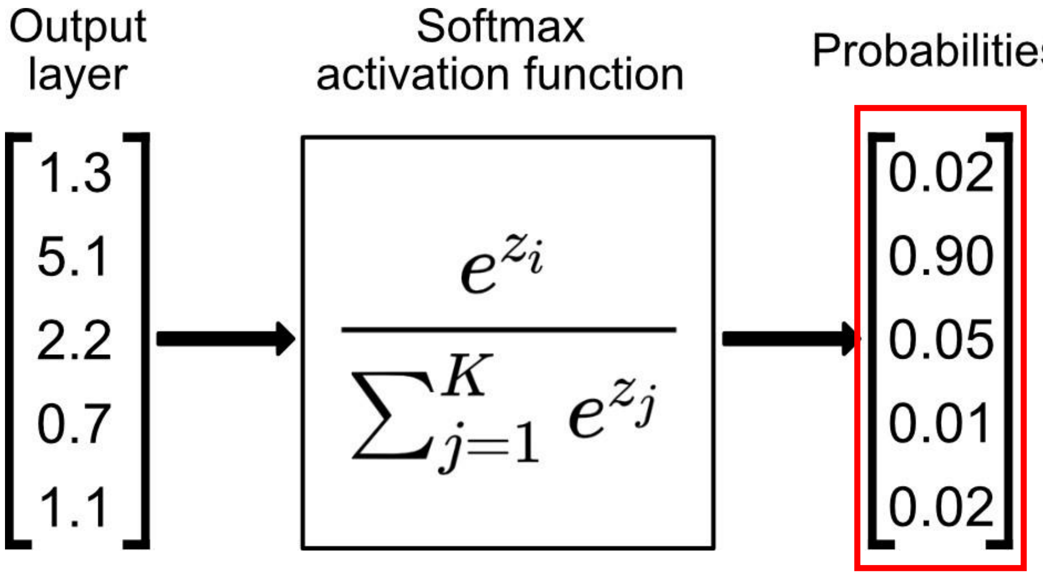
$$h(x) = x$$

- **Softmax**

$$h(x) = \frac{exp(a_i)}{\sum_{i=1}^{n} exp(a_i)}$$

Forward Propagation

- **Softmax activation function**
  - 분류(classification) 문제에서 사용되는 활성화 함수
  - 어떤 데이터가 각 클래스에 속할 확률을 계산하는 함수
  - 특징
    - (1) 모든 output의 값은 0보다 크다
    - (2) 각 클래스의 output값의 합은 1



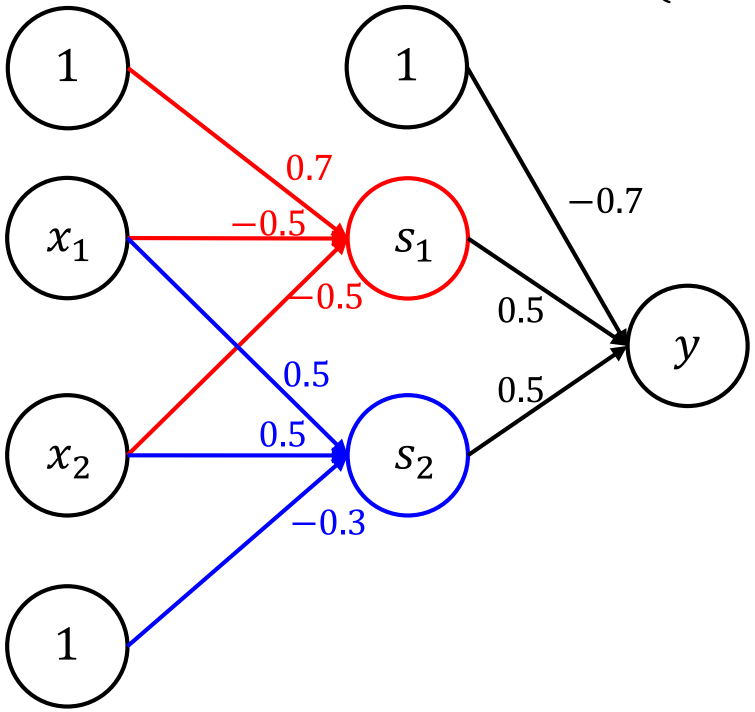
Sum(Probabilities)=1

(출처: https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60)

Training Neural Network

- **Multi-layer Perceptron으로 비선형 분류기도 생성할 수 있다.**
  - **예시: XOR게이트**

$$y = \begin{cases} 0, b + w_1x_1 + w_2x_2 \le 0 \\ 1, b + w_1x_1 + w_2x_2 > 0 \end{cases}$$



Weight를 제대로 설정해야 XOR을 표현합니다.

→ 여기서 이 weight들은 어떻게 알았을까요?

→ 지금은... 그냥 대충 맞췄습니다.

→ 그러나, 층이 깊어지거나 뉴런이 많아진다면? 불가능

→ Data를 활용하여 학습합니다! How??

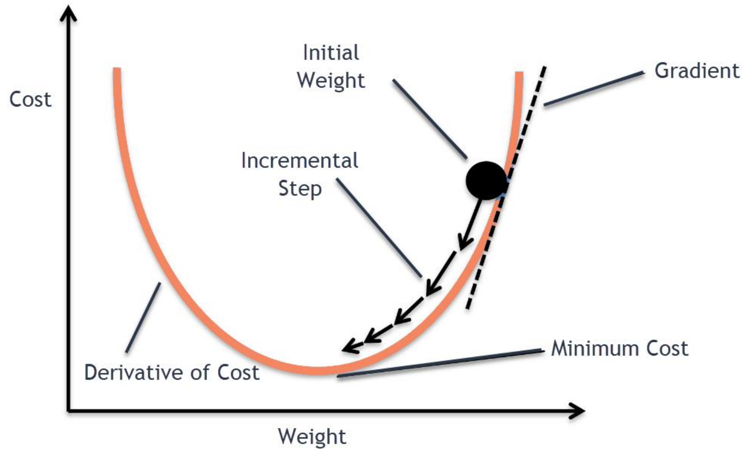
→ Perceptron을 여러 층 쌓아서 왠만한 함수는 근사하게(approximate)하게 표현할 수 있다.

Training Neural Network

- **단계적으로 적합한 weight를 탐색**
  - 사람의 직관으로 찾았던 이전 예시와 달리, NN의 weight들은 적합한 weight를 단계적으로 탐색
  - Weight들을 바꿔가면서 Neural Network를 좀 더 나은 방향으로 개선

더 나은 Neural Network?? → Loss function

어떻게 weight를 바꾸는지? → Gradient Descent



(출처: https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1)

Loss Function

- **신경망 성능의 '나쁨'을 나타내는 지표**
  - 현재의 신경망이 훈련 데이터를 얼마나 잘 처리하지 못하는지 나타냄
  - 모델이 예측한 값과 실제 값의 차이
  - 다음과 같이 크게 2개의 함수를 주로 사용함
    - 회귀: 평균 제곱 오차(Mean Squared Error, MSE)
    - 분류: 교차 엔트로피 (Cross Entropy)

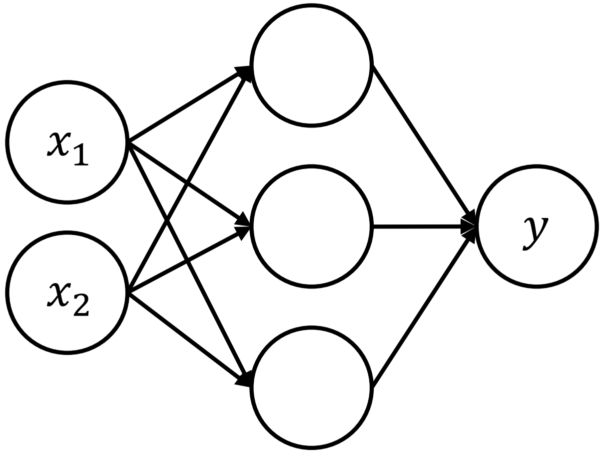
Loss Function

- **Loss Function(1): 오차제곱합(Sum of Squares for Error, SSE)**
  - **회귀** 모델링 시에 주로 사용되는 loss function (분류 모델링에도 사용 가능)

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

y_i: i번째 실제 data
$\hat{y}_i$: i번째 예측 data
n: 데이터의 수

예제1. 실제 데이터와 신경망이 출력한 값이 다음과 같이 주어졌을 때, SSE값을 구하여라.



$$y = 0.7, \hat{y} = 0.5$$

$$SSE = (0.7 - 0.5)^2 = 0.2^2 = 0.04$$

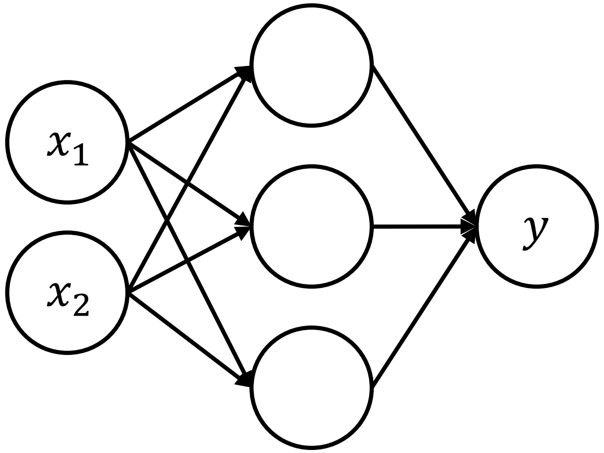
Loss Function

- **Loss Function(2): 평균 제곱 오차 (Mean Squared Error, MSE)**
  - **회귀** 모델링 시에 주로 사용되는 loss function (분류 모델링에도 사용 가능)

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \Rightarrow MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

y_i: i번째 실제 data
$\hat{y}_i$: i번째 예측 data
n: 데이터의 수

예제2. 실제 데이터와 신경망이 출력한 값이 다음과 같이 주어졌을 때, MSE값을 구하여라.



$$y = \begin{bmatrix}3\\4\\5\end{bmatrix}, \hat{y} = \begin{bmatrix}2.7\\4.2\\5.5\end{bmatrix}$$

$$MSE = \frac{1}{3}\{(3-2.7)^2 + (4-4.2)^2 + (5-5.5)^2\} = \frac{0.38}{3}$$

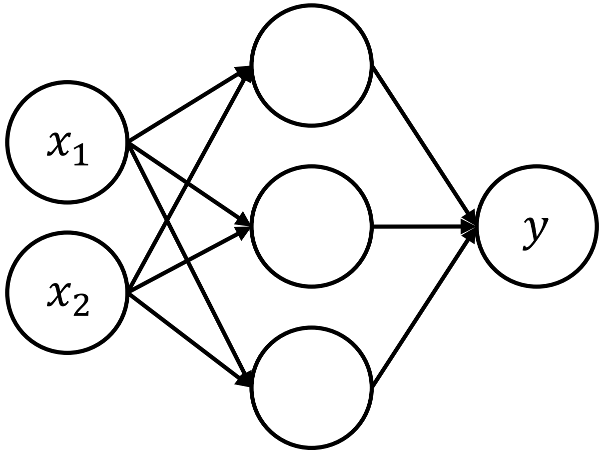
Loss Function

- **Loss Function(3): 제곱근 평균 제곱 오차 (Root Mean Squred Error, RMSE)**
  - **회귀** 모델링 시에 주로 사용되는 loss function (분류 모델링에도 사용 가능)
  - MSE에 제곱근을 적용한 값

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad \Rightarrow \quad RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

y_i: i번째 실제 data
$\hat{y}_i$: i번째 예측 data
n: 데이터의 수

예제3. 실제 데이터와 신경망이 출력한 값이 다음과 같이 주어졌을 때, RMSE값을 구하여라.



$$y = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}, \hat{y} = \begin{bmatrix} 2.7 \\ 4.2 \\ 5.5 \end{bmatrix}$$

$$RMSE = \sqrt{\frac{1}{3}\{(3-2.7)^2 + (4-4.2)^2 + (5-5.5)^2\}} = \sqrt{\frac{0.38}{3}}$$

Loss Function

- **Loss Function(4): 교차 엔트로피 오차(Cross Entropy Error)**
  - 두 분포 간의 차이를 나타내는 척도
  - **분류** 모델링 시에 주로 사용되는 loss function (실제 label은 onehot encoding)

$$E = -\sum_{k=1}^{d} y_k \log \hat{y}_k$$

y_k: 실제 값의 k번째 차원 값
$\hat{y}_k$: 예측 값의 k번째 차원 값 (신경망 출력)
d: 데이터의 차원
log: 밑이 자연상수(e)인 log (=ln)

  - 참고할만한 자료
    - 엔트로피의 의미: https://hyunw.kim/blog/2017/10/14/Entropy.html
    - 교차 엔트로피의 의미: https://hyunw.kim/blog/2017/10/26/Cross_Entropy.html

Loss Function

- **Loss Function(4): 교차 엔트로피 오차(Cross Entropy Error)**
  - 두 분포 간의 차이를 나타내는 척도
  - **분류** 모델링 시에 주로 사용되는 loss function (실제 label은 onehot encoding)

$$E = -\sum_{k=1}^{d} y_k \log \hat{y}_k$$

y_k: 실제 값의 k번째 차원 값
$\hat{y}_k$: 예측 값의 k번째 차원 값 (신경망 출력)
d: 데이터의 차원

  - 예시: 이진 교차 엔트로피(Binary Cross Entropy)
    - 레이블이 2개일 때, 교차 엔트로피 값
    - $\hat{p}$: 모델이 y이라고 예측할 확률 → $(1-\hat{p})$: 모델이 y가 아니라고 예측할 확률

$$E = -\sum y_k \log \hat{y}_k = -y \cdot \log p - (1-y) \cdot \log(1-p)$$

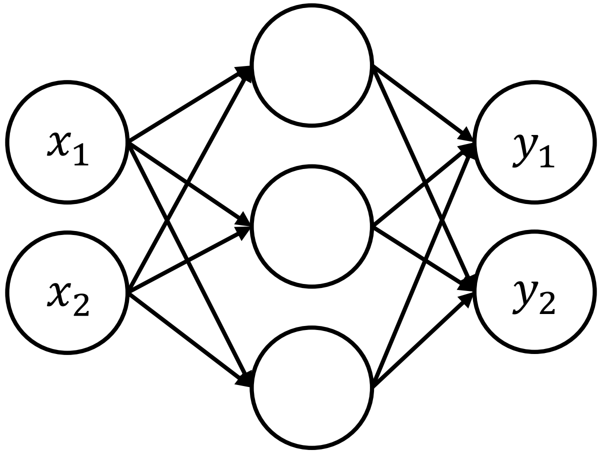
Loss Function

- **Loss Function(4): 교차 엔트로피 오차(Cross Entropy Error)**
  - 두 분포 간의 차이를 나타내는 척도
  - **분류** 모델링 시에 주로 사용되는 loss function (실제 label은 onehot encoding)

$$E = -\sum_{k=1}^{d} y_k \log \hat{y}_k$$

y_k: 실제 값의 k번째 차원 값
$\hat{y}_k$: 예측 값의 k번째 차원 값 (신경망 출력)
d: 데이터의 차원

예제4. 실제 데이터와 신경망이 출력한 값이 다음과 같이 주어졌을 때, Cross Entropy값을 구하여라.



$\boxed{y = 1}, \hat{y} = [0.7 \quad 0.3], y \in \{0,1\}$

y를 one-hot encoding 하자. →

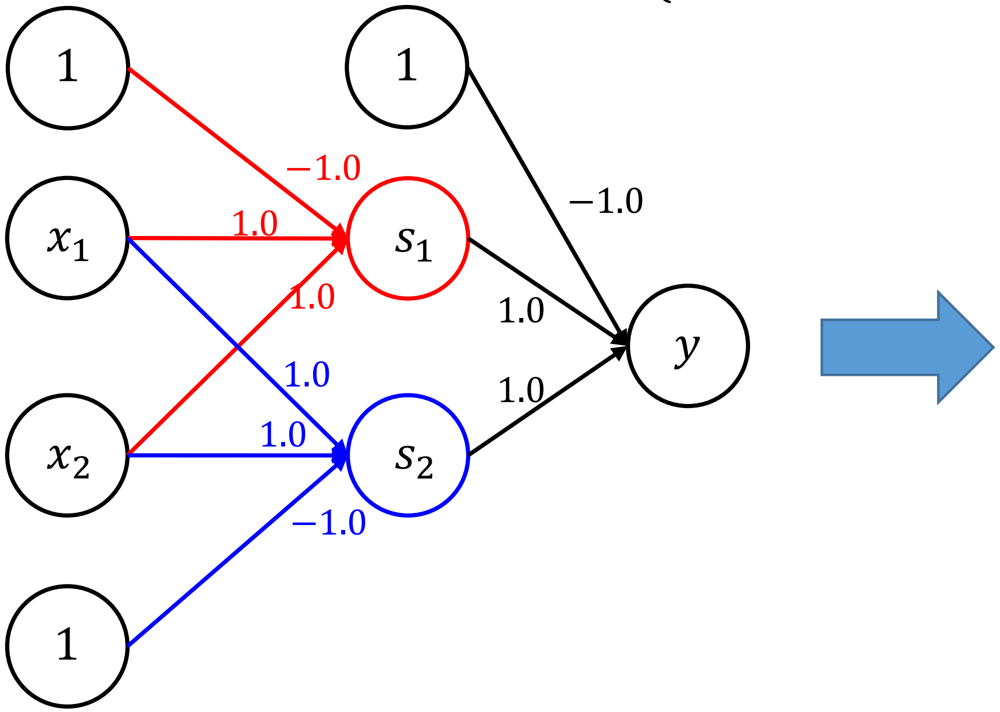
$y = 0 \rightarrow [1 \quad 0]$
$\boxed{y = 1 \rightarrow [0 \quad 1]}$

$$E = -\sum y_k \log \hat{y}_k = -(0 \times \log 0.7) - (1 \times \log 0.3) = -\log 0.3$$

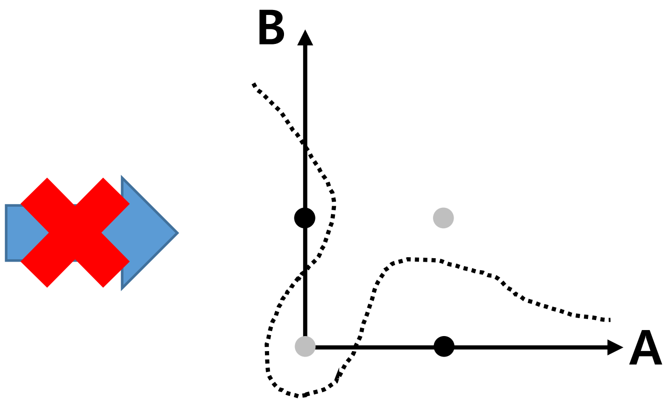
Not trained Perceptron

- 2-layer perceptron이 다음과 같이 weight가 설정되면 어떨까요?
  - 예시: XOR게이트

$$y = \begin{cases} 0, b + w_1x_1 + w_2x_2 \le 0 \\ 1, b + w_1x_1 + w_2x_2 > 0 \end{cases}$$



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



→ 여기서부터 적절한 weight를 찾을 수 있도록 찾아가야 합니다 → Training Neural Network!

Revisit Loss Function

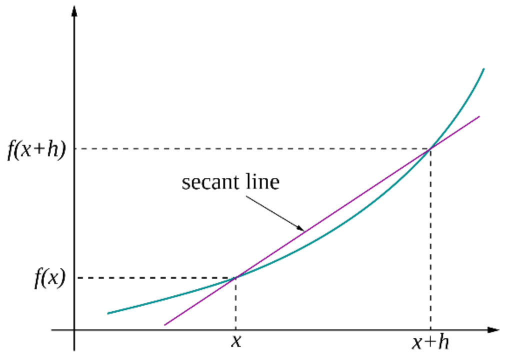
- **신경망 성능의 '나쁨'을 나타내는 지표**
  - 현재의 신경망이 훈련 데이터를 얼마나 잘 처리하지 못하는지 나타냄
  - 모델이 예측한 값과 실제 값의 차이
  - 다음과 같이 크게 2개의 함수를 주로 사용함
    - 회귀: 평균 제곱 오차(Mean Squared Error, MSE)
    - 분류: 교차 엔트로피 (Cross Entropy)

→ 인공 신경망을 training: Loss function이 최소 값이 되도록 하는 weight를 찾기 → How??

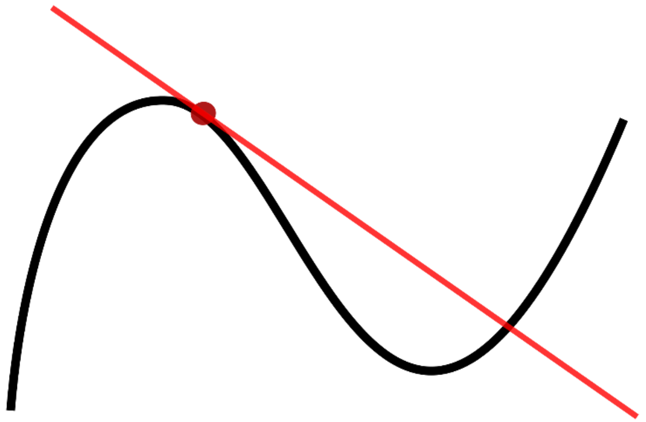
Differentiation (미분)

- **특정 지점에서의 함수의 기울기**
  - (더 정확하게는 함수의 접선의 기울기)
  - 정의

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



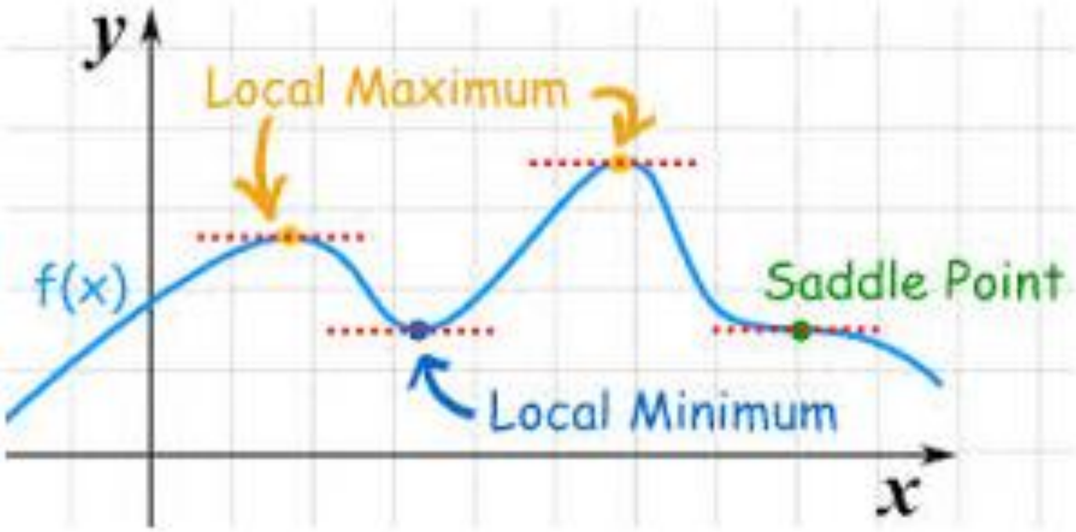
$h \to 0$



(출처: https://ko.m.wikipedia.org/wiki/%ED%8C%8C%EC%9D%BC:Secant-calculus.svg, https://en.wikipedia.org/wiki/Derivative)

Differentiation (미분)

- **고등학교 때를 떠올려 봅시다... (미분을 배우셨다면)**
  - 어떤 함수의 최소, 최대(혹은 극소, 극대)값은?
    - 미분 값이 0인 위치를 찾고 → 그 위치에서의 함수 값을 계산하였습니다.



(출처: https://www.mathsisfun.com/calculus/maxima-minima.html)

Partial Derivatives(편미분)

- **특정 지점에서의 함수의 기울기**
  - 여러 개의 변수를 사용하는 함수에 대해서 특정 지점에서의 함수의 기울기
    - 기울기를 변수별로 따로 계산 (편미분)
    - 계산시, 한 변수는 고정하고 다른 변수에 대해서만 미분을 계산

  - 예제: $f(x_1, x_2) = x_1^2 + x_2^2$에 대하여 $x_1 = 3, x_2 = 4$일 때, 편미분 $\frac{\partial f}{\partial x_1}$값을 구하여라.

$$\left.\frac{\partial f}{\partial x_1}\right|_{x_1=3} = \left.2x_1\right|_{x_1=3} = 6$$

Gradient

- **특정 지점에서의 함수의 기울기**
  - 변수 별로 편미분한 결과값을 벡터 형태로 나타냄
  - 예시: 변수 x_1, x_2를 사용하는 함수 $f(x_1, x_2)$의 gradient

$$grad\ f(x_1, x_2) = \boxed{\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)}$$

→ **Gradient는 vector라는걸 기억해주세요!**

Vector!

  - 예제: $f(x_1, x_2) = x_1^2 + x_2^2$에 대하여 $x_1 = 3, x_2 = 4$일 때, gradient값을 구하여라.

$$grad\ f(x_1, x_2) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)$$

$$\left.\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)\right|_{x_1=3, x_2=4} = \left.(2x_1, 2x_2)\right|_{x_1=3, x_2=4} = (6, 8)$$

Gradient

- **특정 지점에서의 함수의 기울기**
  - **Gradient의 의미**
    - 함수 값을 가장 크게 증가시키는 방향

  - 예제: $f(x_1, x_2) = x_1^2 + x_2^2$에 대하여 $x_1 = 3, x_2 = 4$일 때, gradient값을 구하여라.

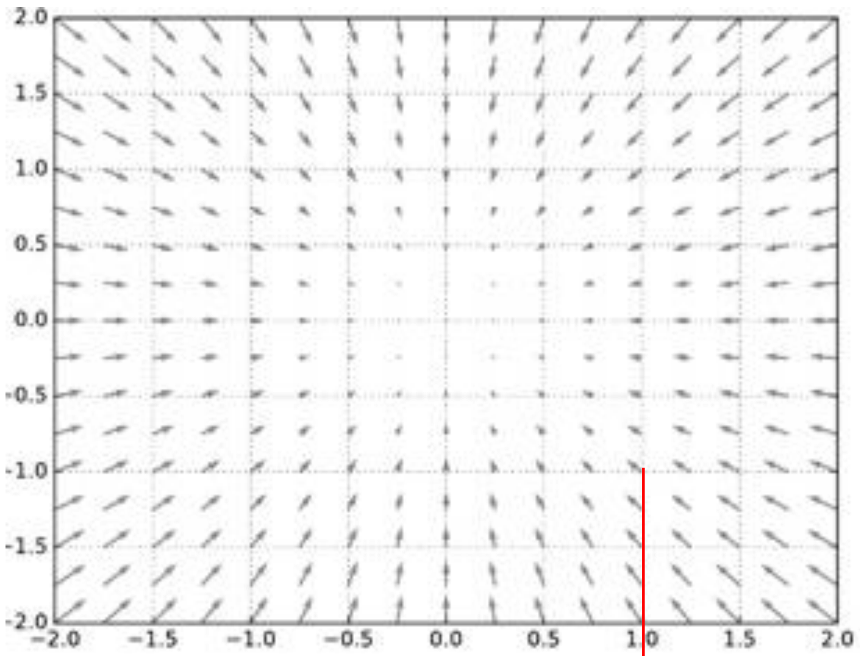
$$grad\ f(x_1, x_2) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)$$

$$\left.\left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}\right)\right|_{x_1=3, x_2=4} = (2x_1, 2x_2)\Big|_{x_1=3, x_2=4} = (6, 8)$$

→ 의미: 내 현재 위치는 (3,4)인데, 이 위치에서 (6,8)의 방향으로 움직이면 값이 가장 크게 증가한다.

Gradient

- **특정 지점에서의 함수의 기울기**
  - 예시: $f(x_1, x_2) = x_1^2 + x_2^2$에서 몇몇 지점에서 gradient에 –값을 붙인 map
    - **기울기가 가리키는 쪽은 각 장소에서 함수의 출력 값을 가장 크게 줄이는 방향**
    - (원래 gradient는 함수의 크기를 가장 크게 증가시키는 방향을 의미 → 반대방향!)

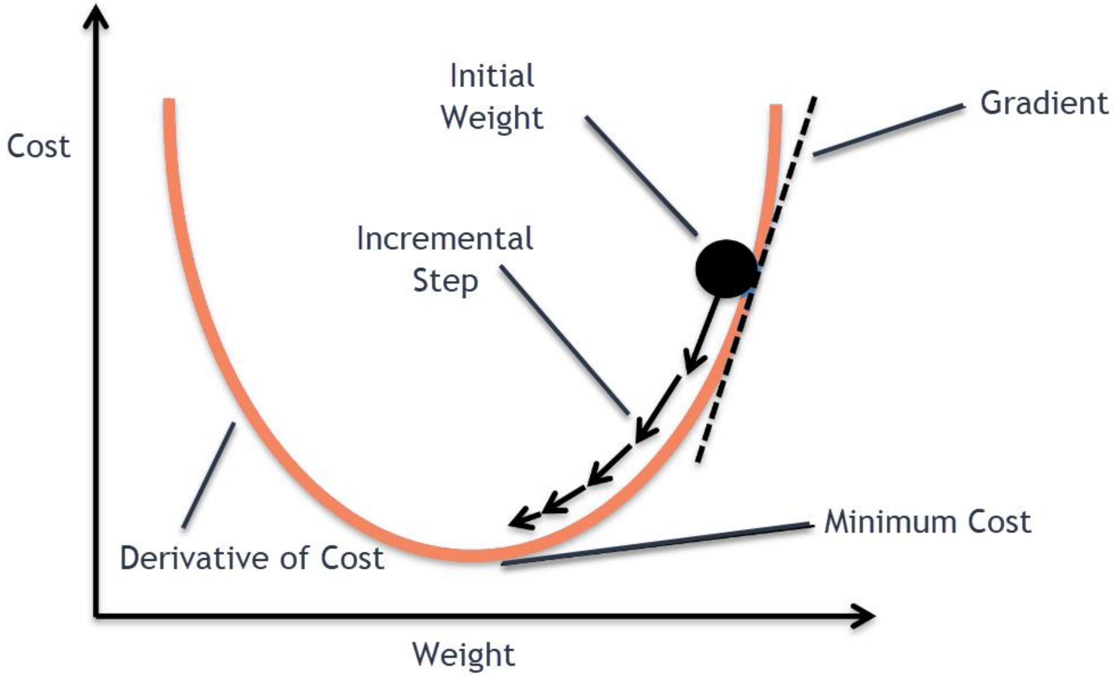


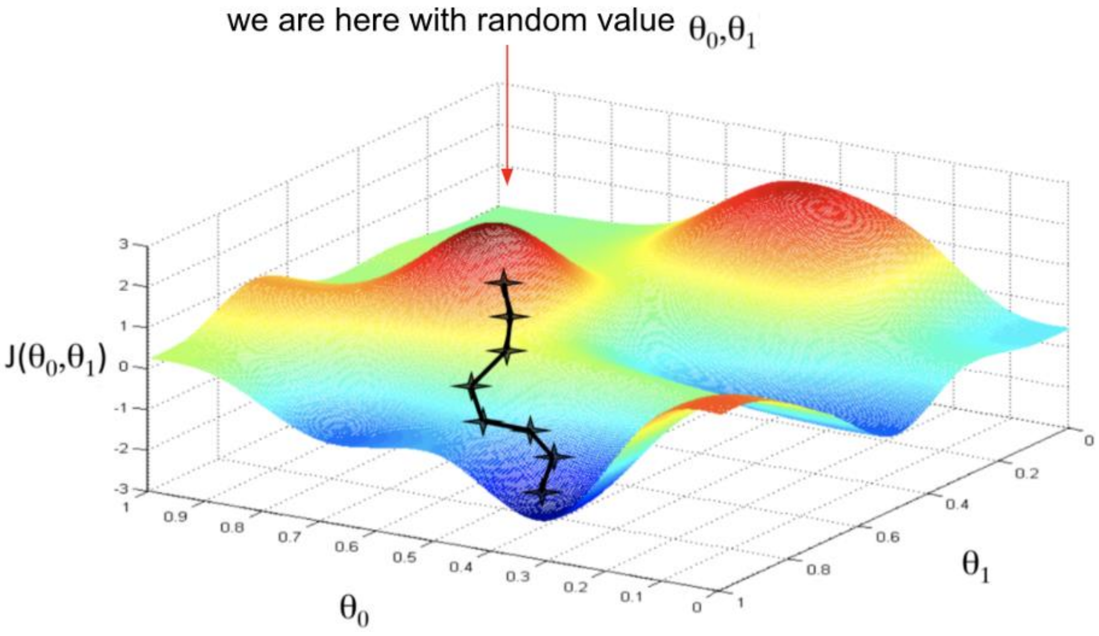
ex) $-grad\ f(1, -1) = (-2, 2)$

Gradient Descent (경사 하강법)

- Gradient: 함수의 출력 값을 가장 크게 줄이는 방향

- Gradient방향으로 일정 거리 만큼 이동하면 원래 값보다 함수 값이 감소





(출처: https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1, https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html)

Gradient Descent (경사 하강법)

- **Learning Rate (학습률)**
  - 앞에서 Gradient를 계산하여 weight가 이동해야 할 방향을 파악
  - 자, 이제 얼마나 이동해야 할까?
    - 약간의 복잡한 수학적 지식이 필요하지만, weight는 다음과 같이 이동해야 함
  - 학습률: 한번의 학습으로 가중치를 얼마나 갱신해야 하는지 결정

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{prev} - \alpha \cdot grad\big(E(\boldsymbol{w})\big)$$

새로운 weight는

기존의 weight로부터

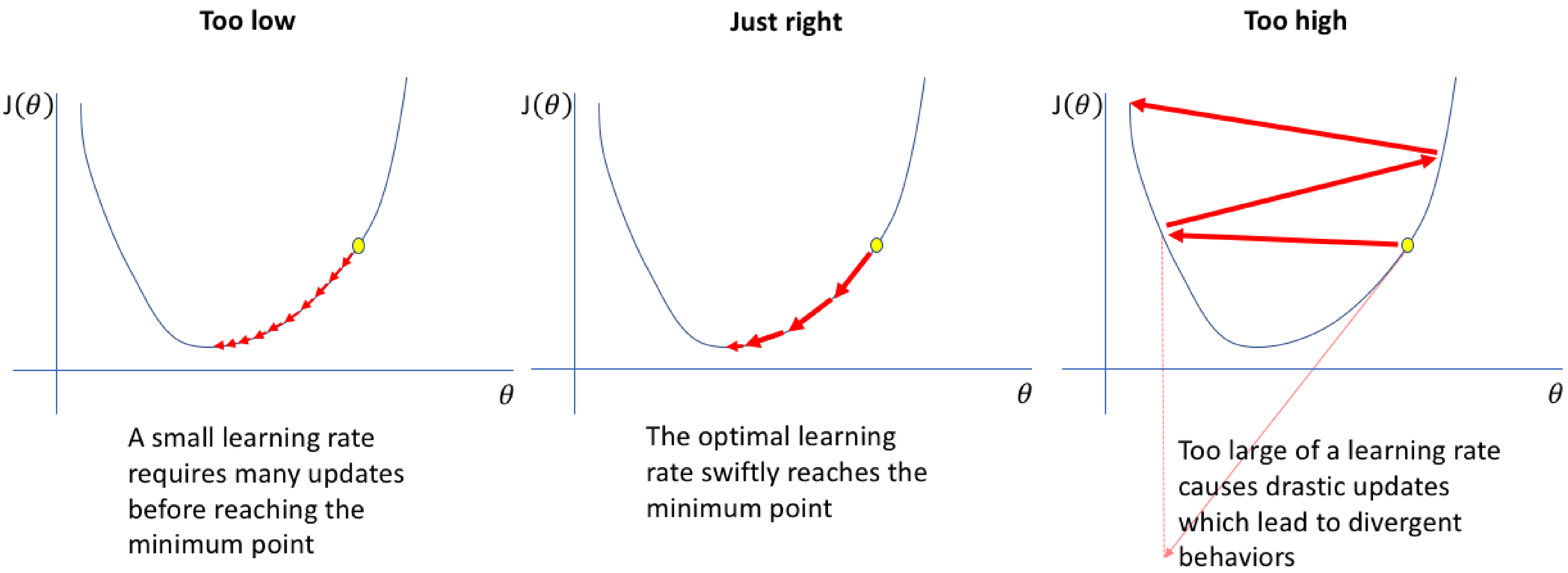
학습률(learning rate)

$\alpha \cdot grad\big(E(\boldsymbol{w})\big)$ 만큼 이동한 위치

(출처: https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1, https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html)

Gradient Descent (경사 하강법)

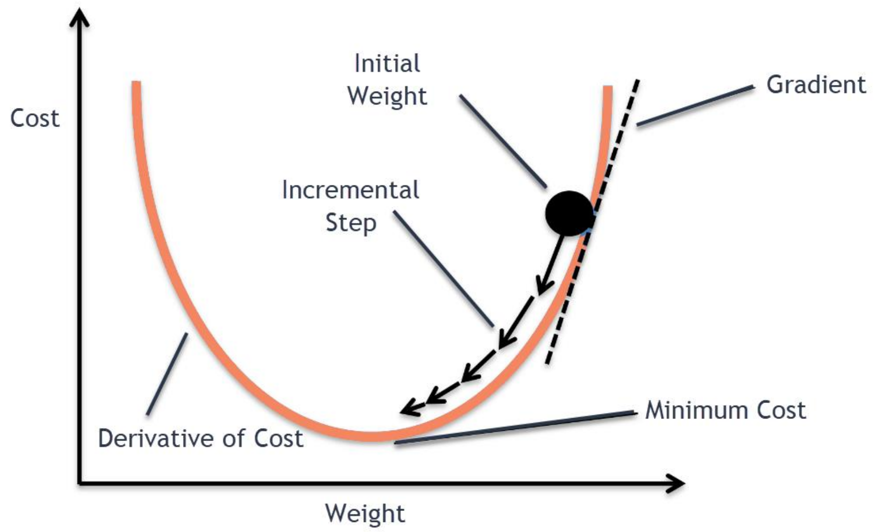
- **Learning Rate (학습률)**
  - 학습률은 적당하게 결정하는 것이 중요

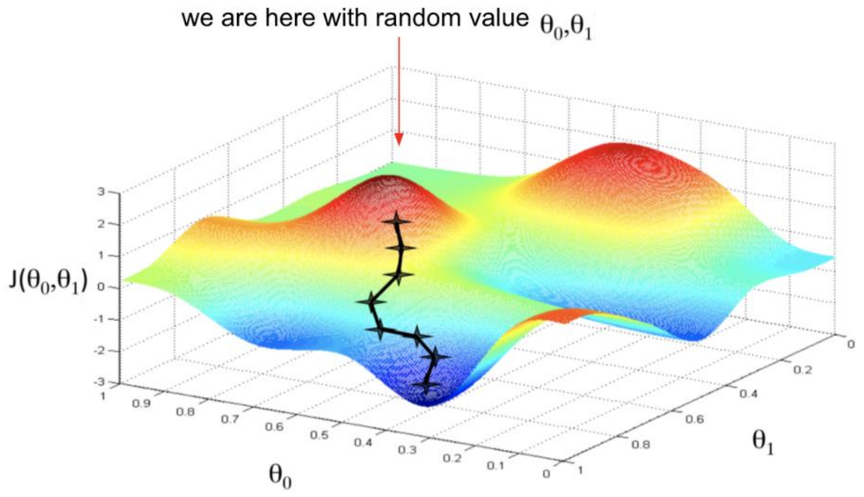


(출처: https://www.jeremyjordan.me/nn-learning-rate/)

Gradient Descent

- **Algorithm**
  - Step 1. 초기 weight 값을 지정
  - Step 2. 현재 weight값에서 gradient값을 계산
  - Step 3. learning rate * gradient만큼 이동하여 업데이트된 weight값을 산출
  - Step 4. 새로운 weight에서 step 2~3을 반복 (특정 조건을 만족할 때까지)

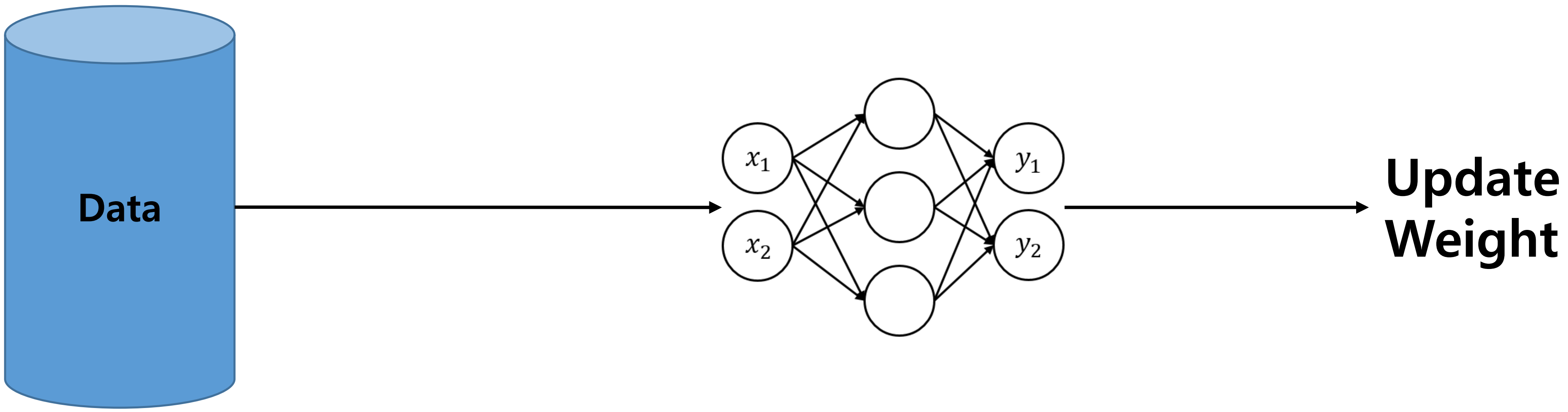




(출처: https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1, https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html)

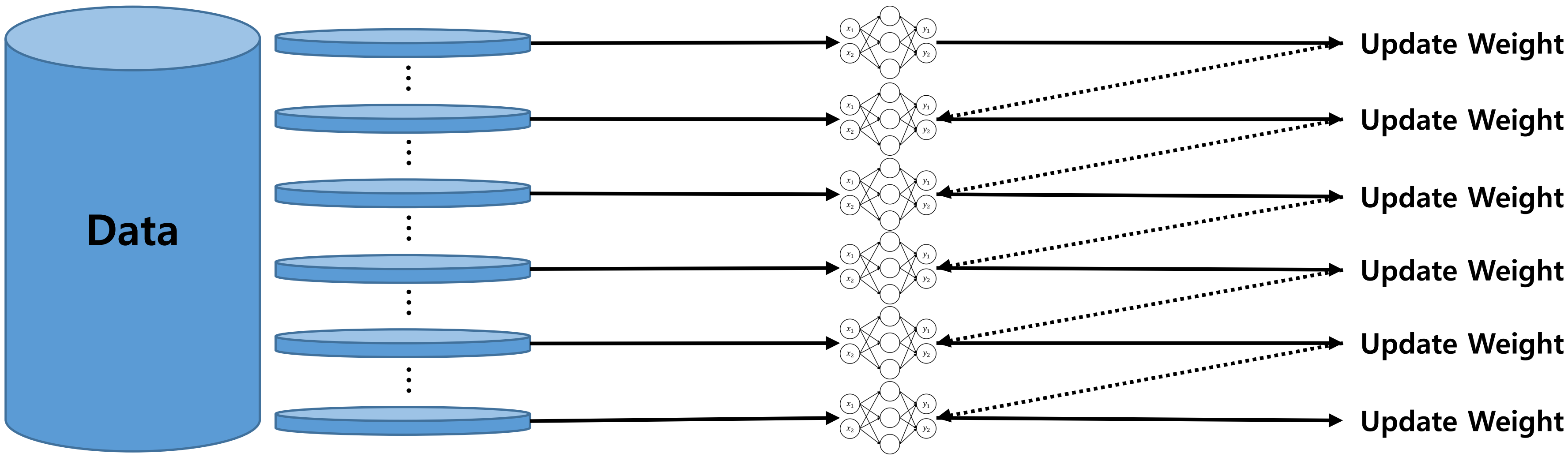
Neural Network 학습 시 고려사항(1)

- **Weight 업데이트 주기**
  - **방법 1. Batch Gradient Descent(BGD)**
    - 전체 데이터를 하나의 batch로 간주하여, weight를 학습시키는 방법
    - Weight들을 고정시키고 모든 데이터들에 대해 loss를 산출한 뒤, 이로부터 gradient 계산



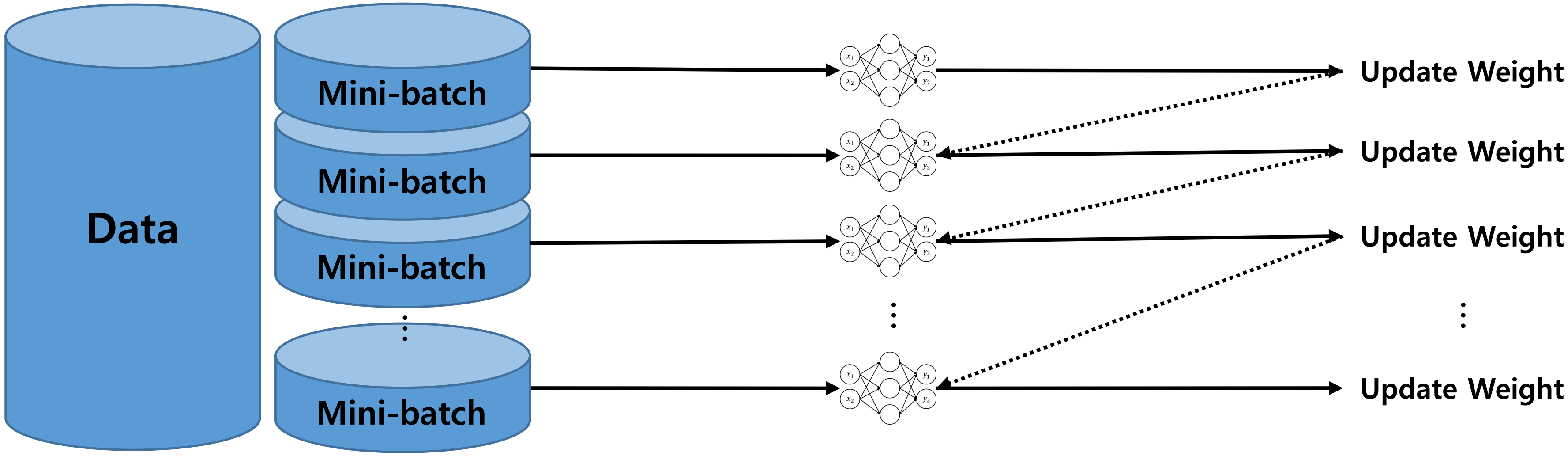
Neural Network 학습 시 고려사항(1)

- **Weight 업데이트 주기**
  - **방법 2. Stochastic Gradient Descent(SGD)**
    - 개별 데이터를 하나의 batch로 간주하여, weight를 학습시키는 방법
    - Single data로 weight를 업데이트 한 뒤, 새로운 data는 업데이트된 network에 넣어 다시 update



Neural Network 학습 시 고려사항(1)

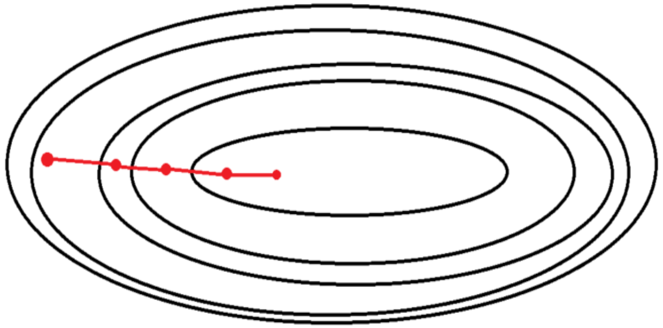
- **Weight 업데이트 주기**
  - **방법 3. Mini-batch Stochastic Gradient Descent(MSGD)**
    - 데이터의 일부를 하나의 batch로 묶고, batch 단위로 weight를 학습시키는 방법
    - Mini-batch로 weight를 업데이트 한 뒤, 새로운 Mini-batch를 업데이트된 network에 넣어 update
    - (Pytorch등과 같은 딥러닝 패키지에서 일반적으로 SGD는 MSGD를 의미)



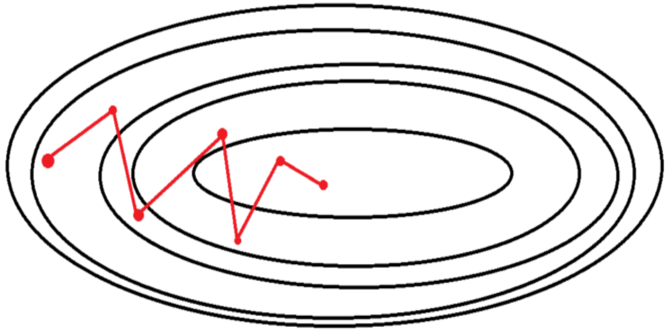
Neural Network 학습 시 고려사항(1)

- **Weight Update 주기**

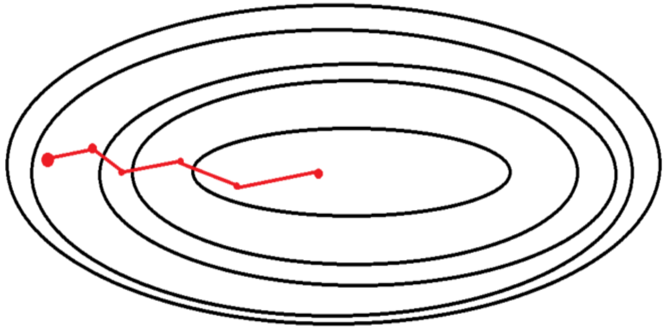
Stochastic Gradient Descent

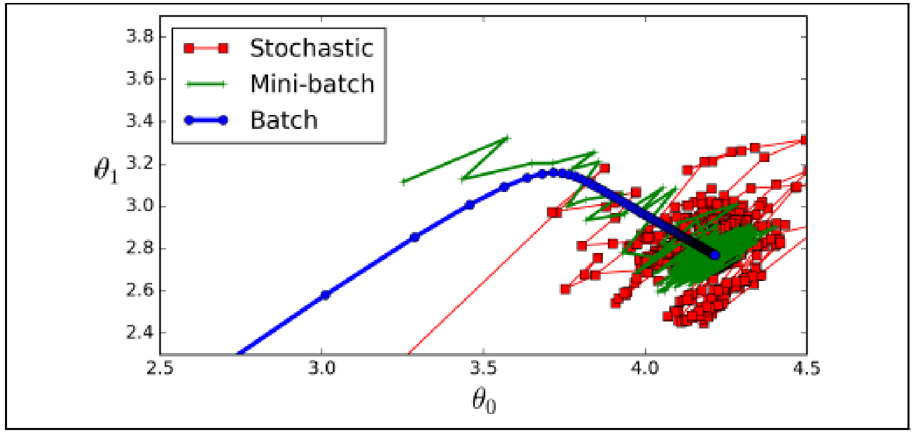


Batch Gradient Descent



Mini-batch Stochastic Gradient Descent



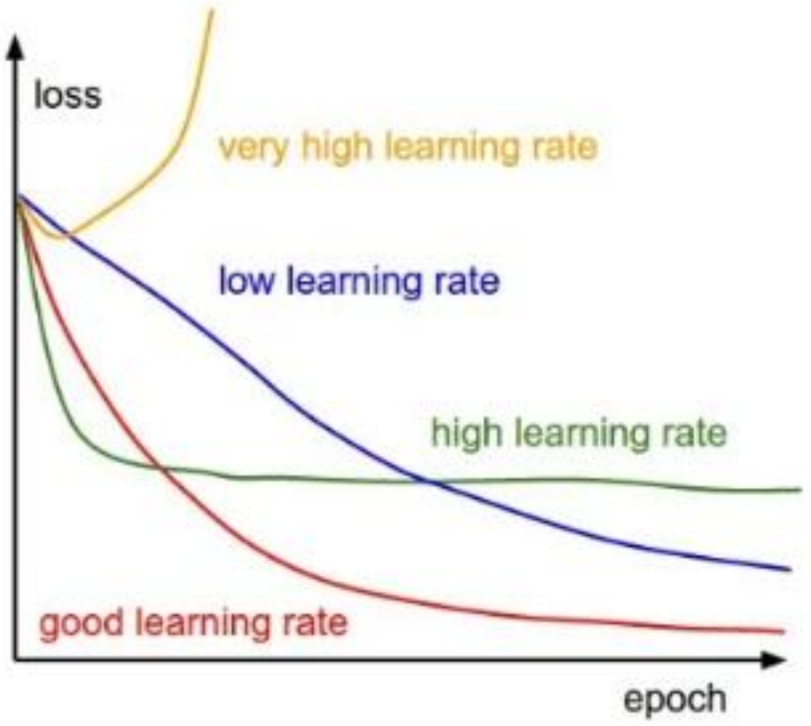


(출처: https://skyil.tistory.com/68)

Neural Network 학습 시 고려사항(2)

- **Learning Rate Policy**

$$w^{new} = w^{prev} - \alpha \cdot grad(E(w))$$



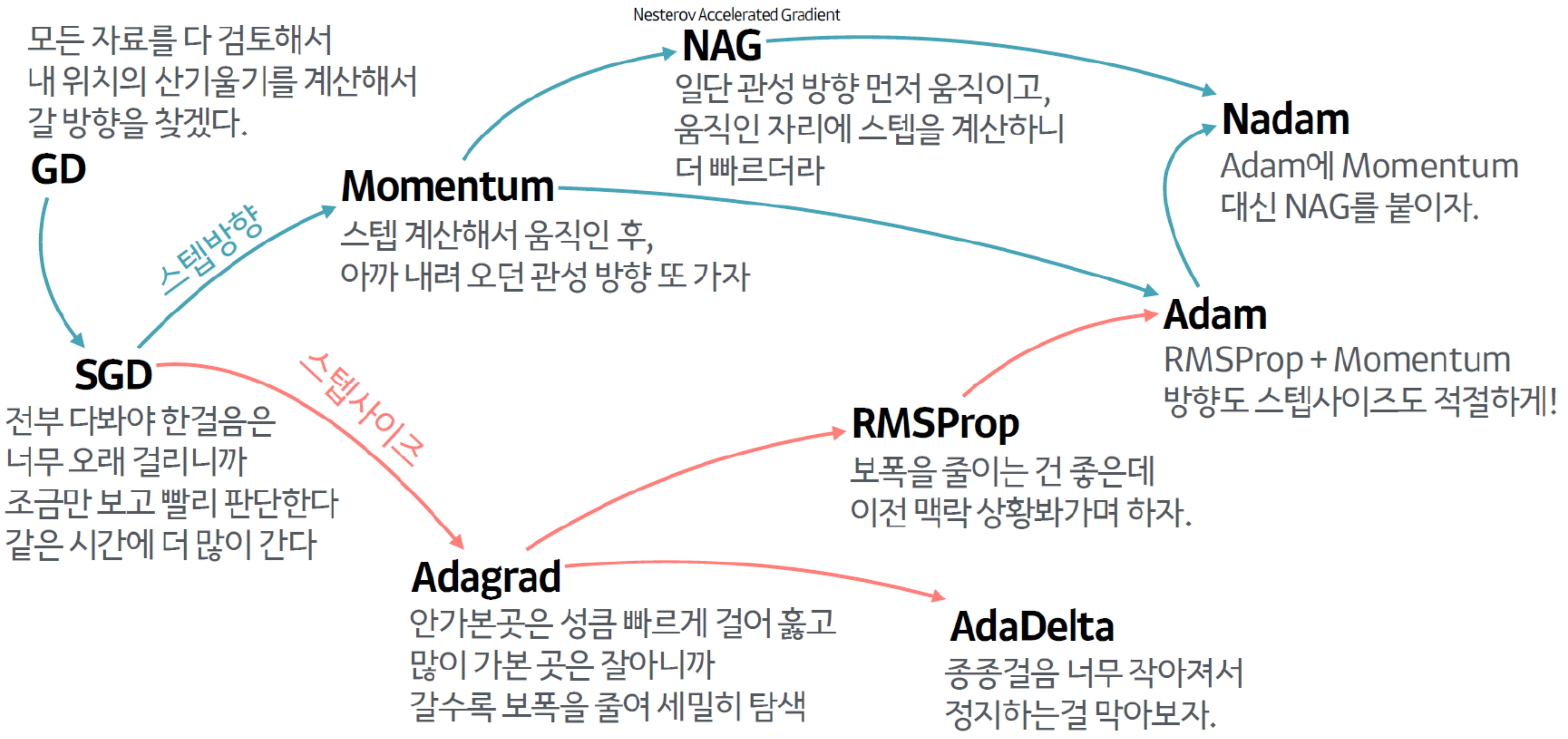
보폭이 너무 작으면 오래 헤매고(파란라인)

보폭이 너무 크면, 오솔길을 지나친다(녹색라인)

(출처: https://www.slideshare.net/yongho/ss-79607172)

Neural Network 학습 시 고려사항(2)

- **Learning Rate Policy**



(출처: https://www.slideshare.net/yongho/ss-79607172)

Summary

- **Artificial Neural Network**
  - 생물학적 신경망에서 영감을 얻은 통계학적 학습 알고리즘
  - 다양한 종류의 ANN이 존재
  - Feed-Forward Neural Network: 데이터가 input → output방향으로만 흐르는 NN
    - 입력층(input layer), 은닉층(hidden layer), 출력층(output layer)이 존재

- **Perceptron**
  - 다수의 신호를 입력받아 하나의 신호를 출력하는 알고리즘
  - 간단한 Perceptron을 여러 층을 쌓아서 복잡한 함수도 approximately하게 표현 가능

Summary

- **Activation Function**
  - 뉴런에 들어오는 입력 signal의 총합을 output으로 변환하는 함수
  - 이 activation function은 비선형 함수여야 함 (그렇지 않으면 층을 쌓는 이유가 없음)

- **Forward Propagation**
  - 데이터를 NN에 입력하였을 때, output layer에서 출력값이 나오는 과정

- **Training neural network**
  - 원하는 함수를 approximately하게 나타내기 위해 적절한 weight를 나타내는 과정
  - Loss Function을 최소화하는 weight를 찾는게 목표이며, Gradient Descent로 탐색
    - Loss Function: ML모델 성능의 '나쁨'을 나타내는 지표
    - Gradient Descent: Weight를 Gradient방향으로 이동시키면서 점진적으로 학습하는 방법

End of the documents