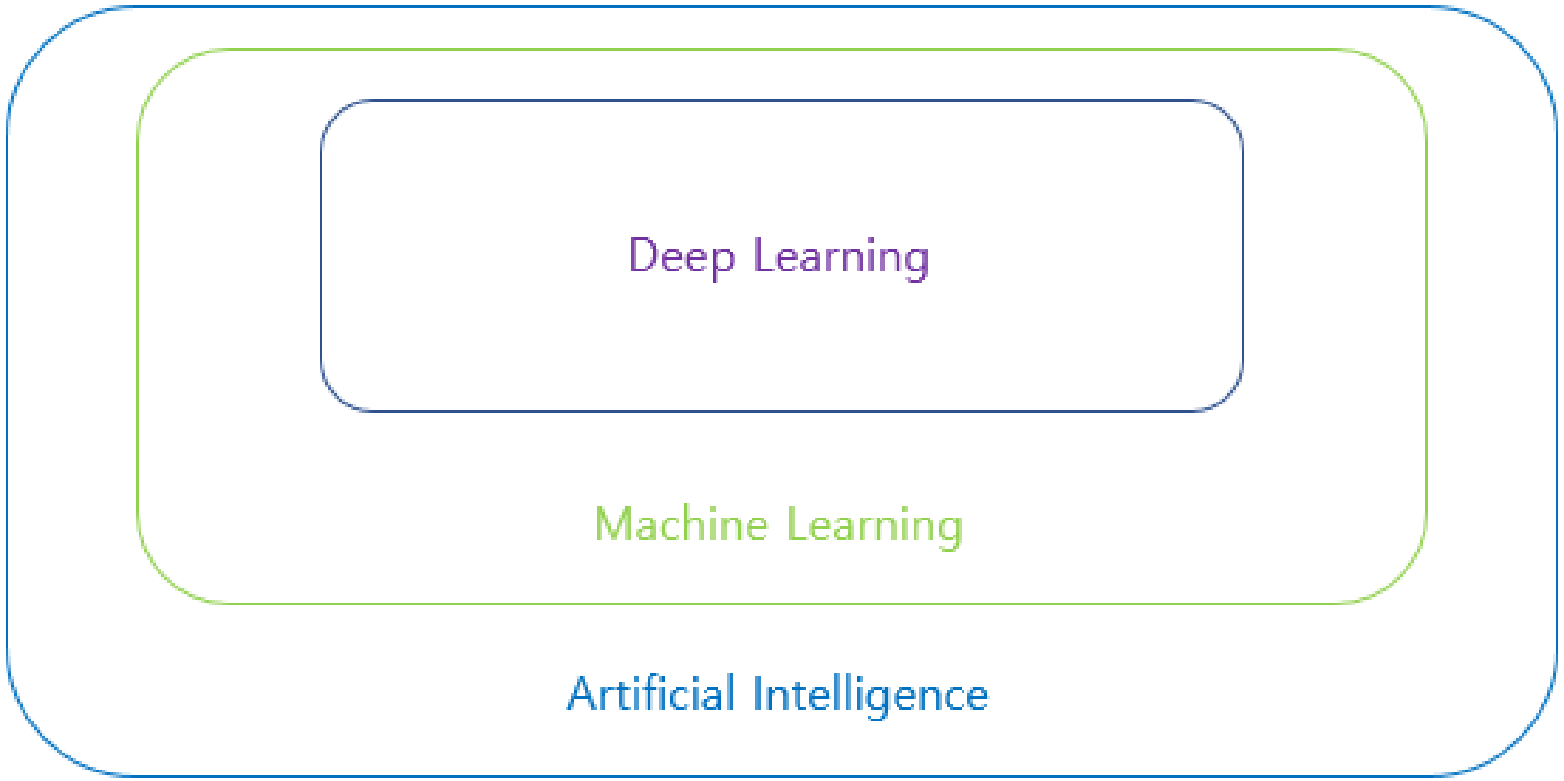
Deep Learning

17기 이현정

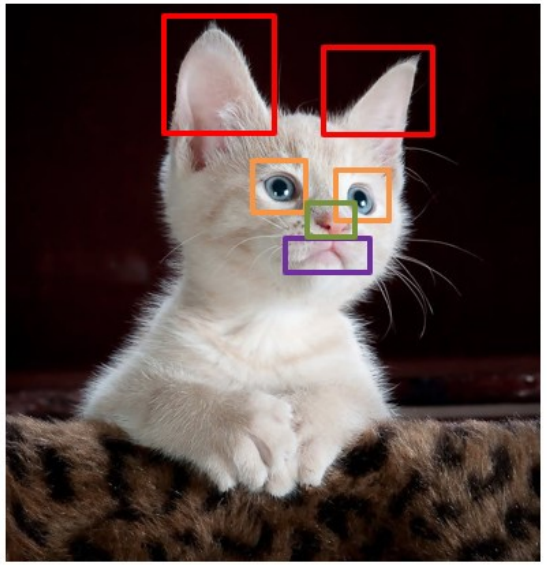
0. 딥러닝이란?

- 인공지능? 머신러닝? 딥러닝?



인공지능(人工知能, 영어: artificial Intelligence, **AI**)은 인간의 학습능력, 추론능력, 지각능력, 논증능력, 자연언어의 이해능력 등을 인공적으로 구현한 컴퓨터 프로그램 또는 이를 포함한 컴퓨터 시스템이다. 하나의 인프라 기술이기도 하다.[1][2] 인간을 포함한 동물이 갖고 있는 지능 즉, natural intelligence와는 다른 개념이다.

원본 사진



학습 : 특징 추출

고양이: 0.95
개: 0.05

분류

귀, 눈, 코, 입 등의
구조·색상·형태 파악

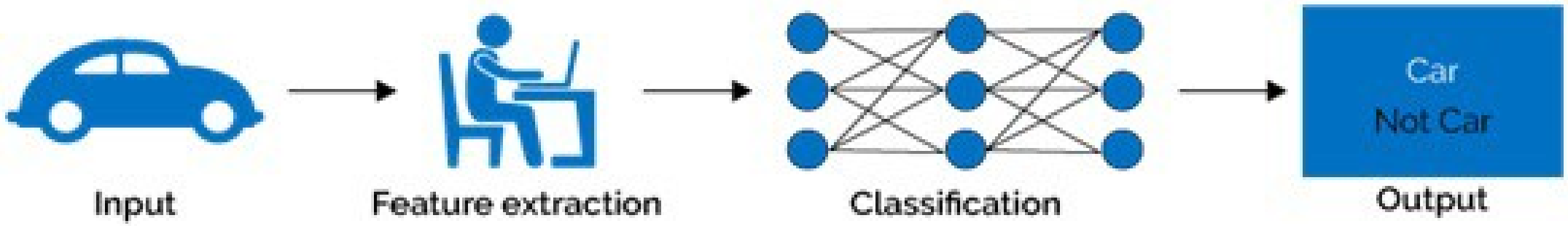
머신러닝 vs 딥러닝

- 고양이의 귀, 눈, 코, 입 등의 학습할 특징을 알려줌
- 즉, 학습을 위해 필요한 feature는 사람이 정함.

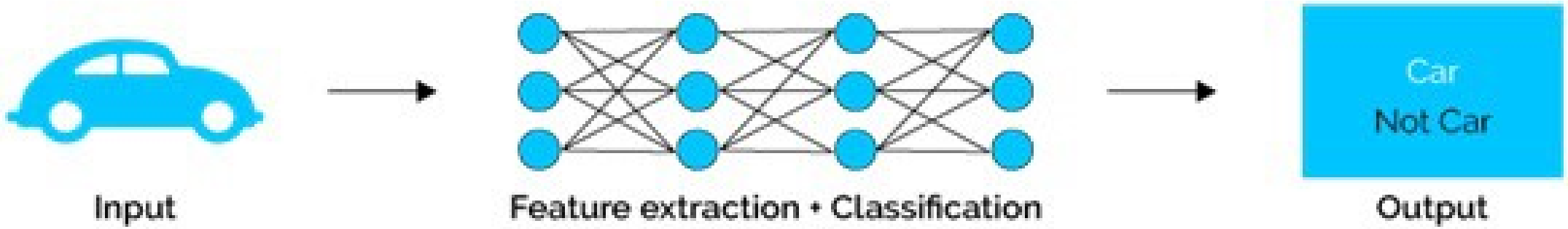
- 사진을 주고 고양이인지 아닌지를 알려줌
- 즉, 학습을 위해 필요한 feature 또한 기계 스스로 학습

딥러닝은
feature extraction을
스스로 하는
심층적인 학습을 한다!

Machine Learning

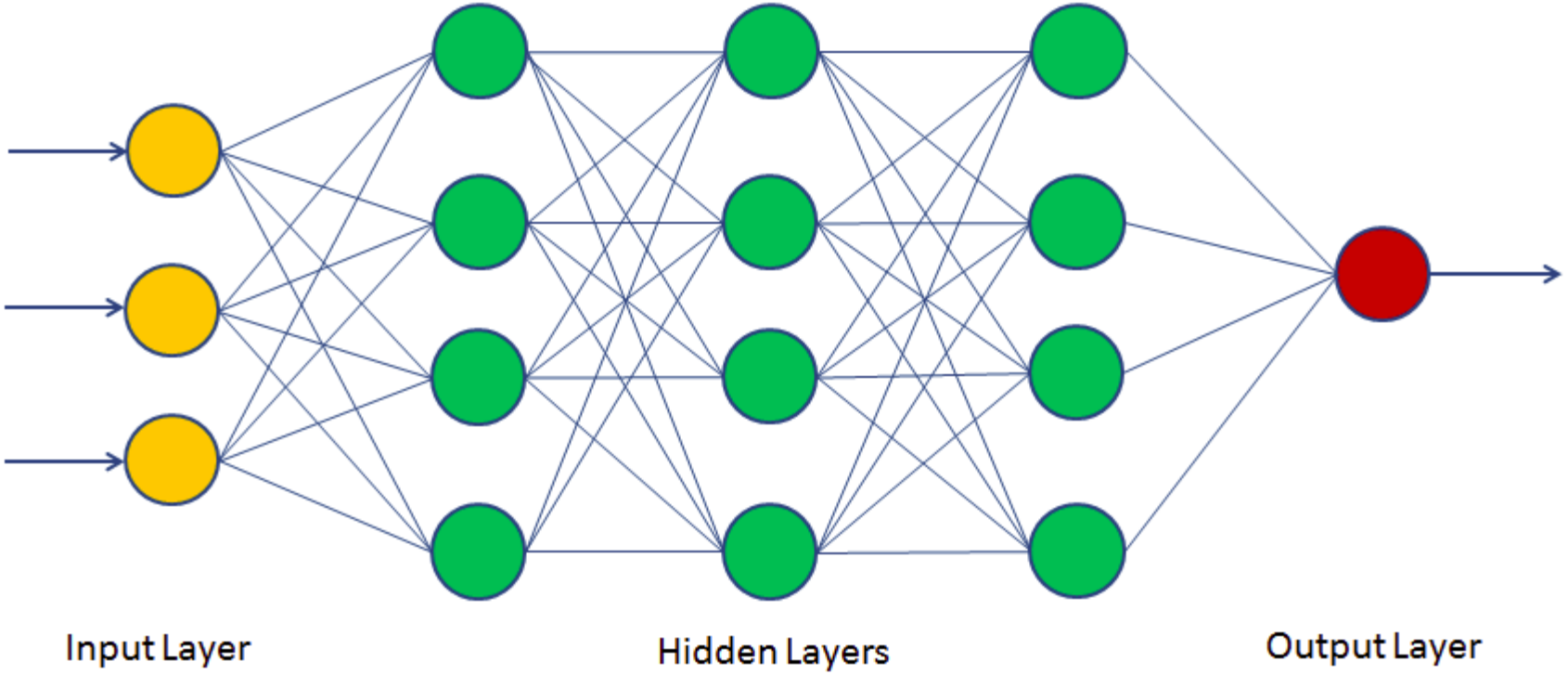


Deep Learning



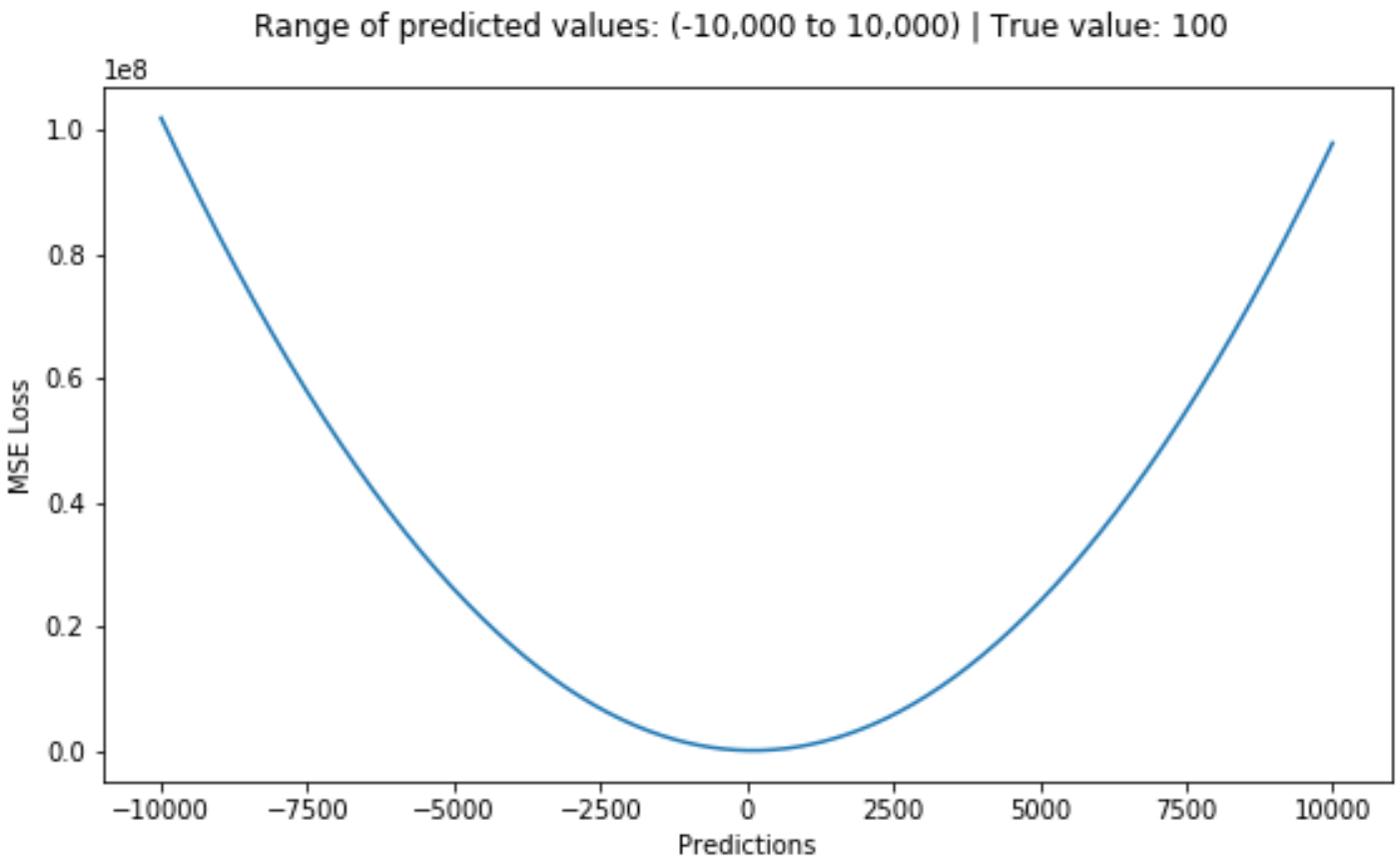
1. 딥러닝 워크플로

- Fundamental structure: 인간의 신경망을 본 따 만든 neural network 이용
input을 받아 신호를 전달하며 output을 연산하는 구조
어떤 신호를 얼마만큼 가중치를 주어 전달할 지를 딥러닝은 스스로 학습(feature extraction)



- loss function

실제값과 예측값의 차이로 나타나는 손실을 가중치들에 대한 손실 함수로 표현 가능함.



$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)}^{2}$$

The square of the difference between actual and predicted

$$Y_i = \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_K X_{iK} + \epsilon_i, 1 \leq i \leq n$$

loss function은 다양함
학습에 따라 적절한 loss function을 사용하여야 함, 회귀 vs 분류 …

-Cross entropy
-Log Likelihood
-KL divergence
-Mean Absolute Error
-Mean Squared Error

- optimize

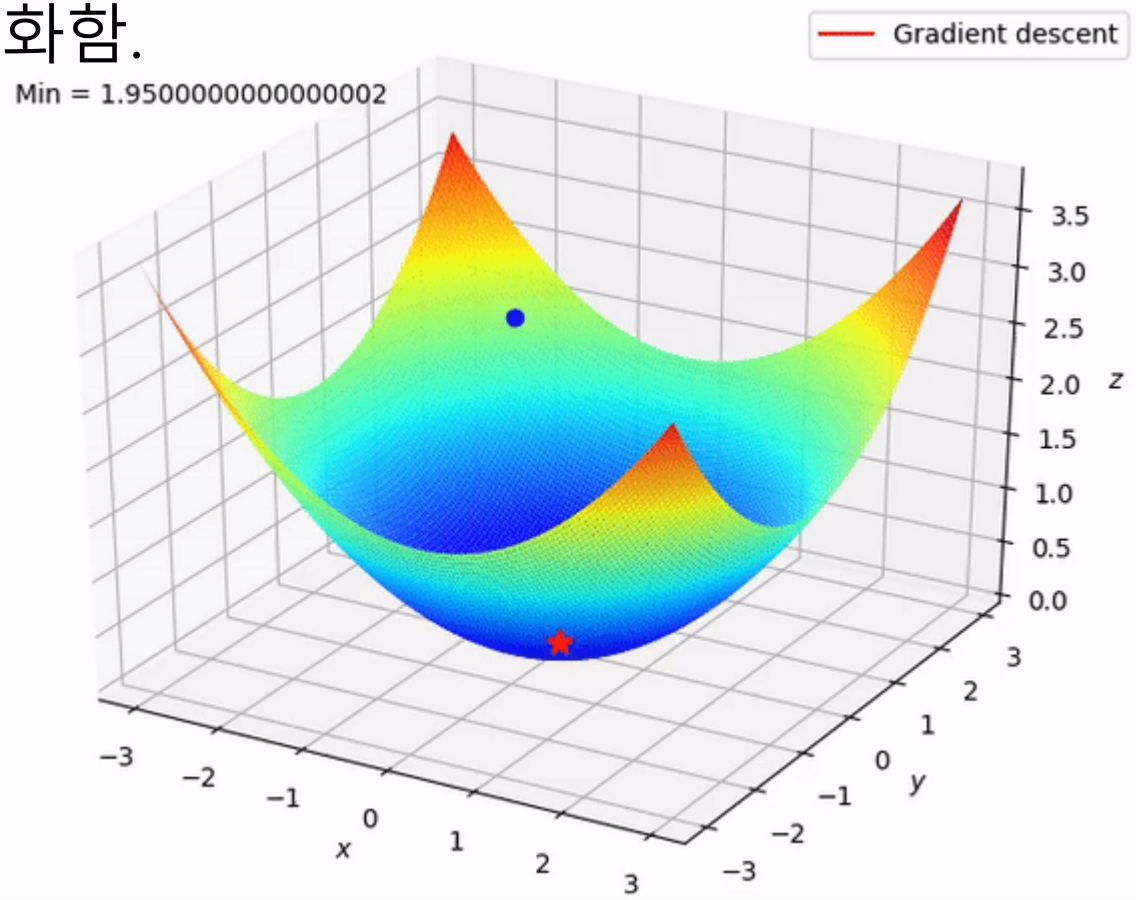
Gradient descent(경사하강법)

경사를 계산하여 점진적으로 밑으로 향해가며

Loss를 최소화하는 가중치를 최적화함.

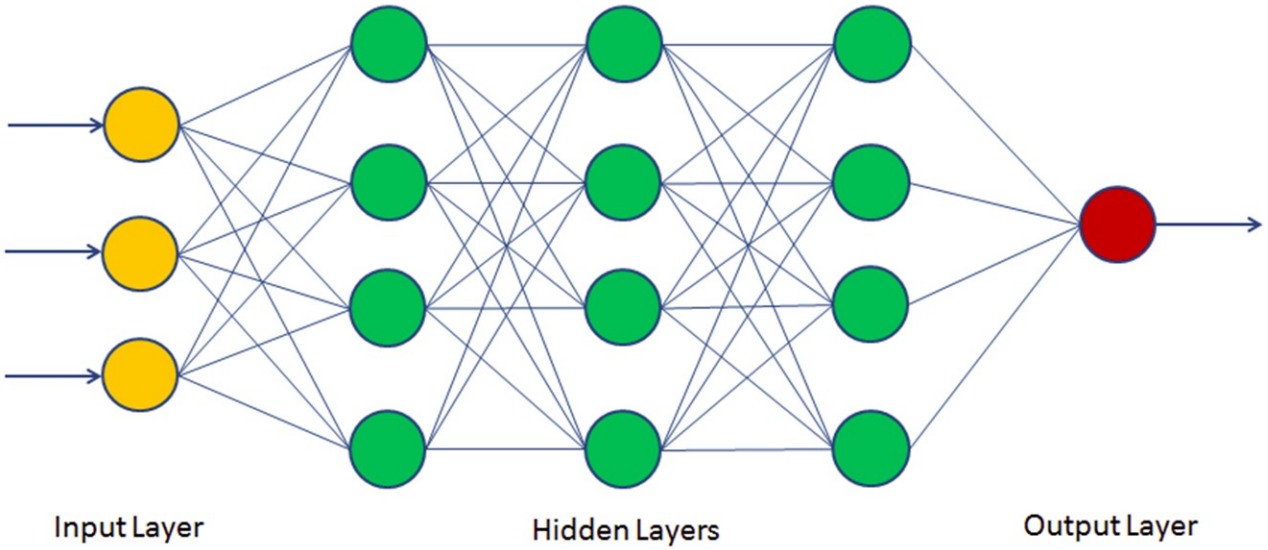
Optimizer 또한 다양하게 존재!

- Gradient Descent
- SGD
- Adam
- Rmsprop

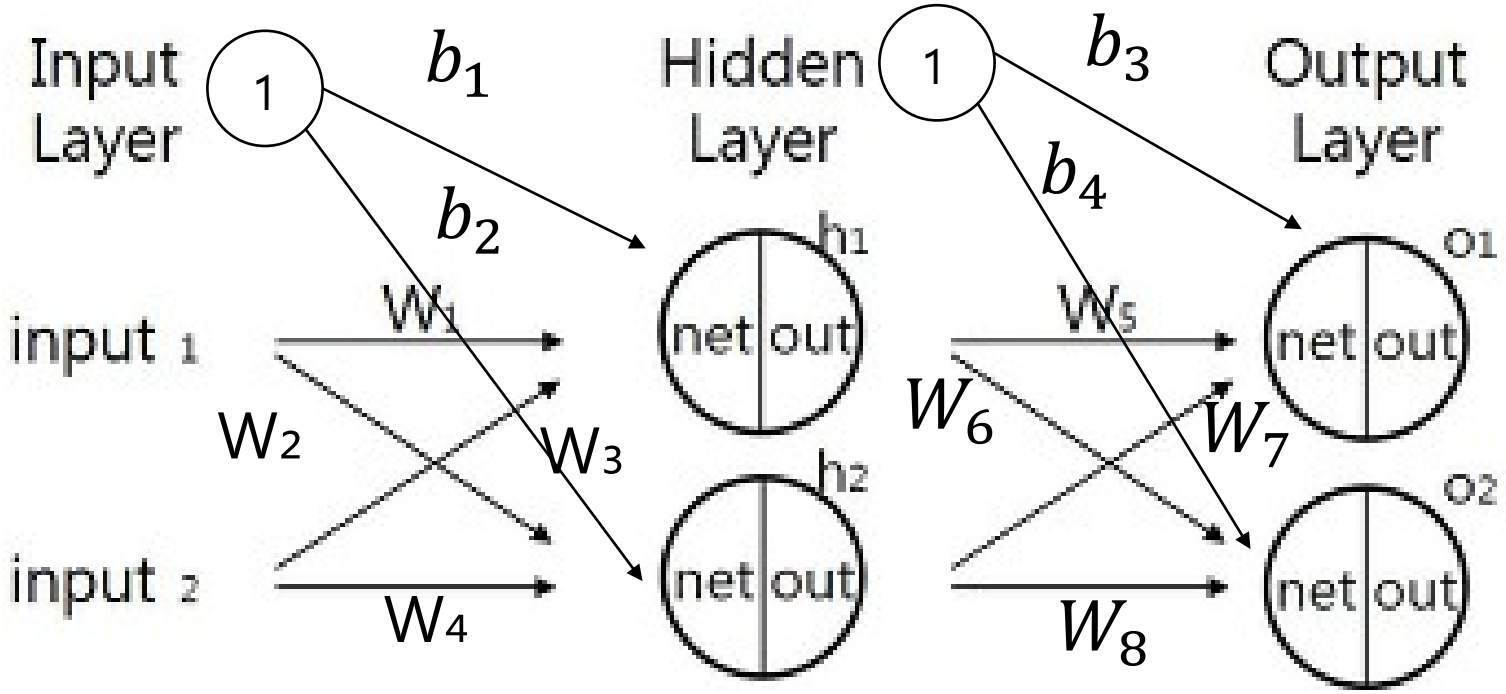


2. Forward propagation

- 가중치를 이용한 연산을 통해 실제 y 값에 대한 예측값을 연산하는 과정.
- Y에 대한 예측값을 통해 가중치에 대한 손실을 구할 수 있음.
- 가중치들에 대한 linear한 연산과정과 ***activation function***을 거쳐 y에 대한 예측값을 구함.



Caculation of the output



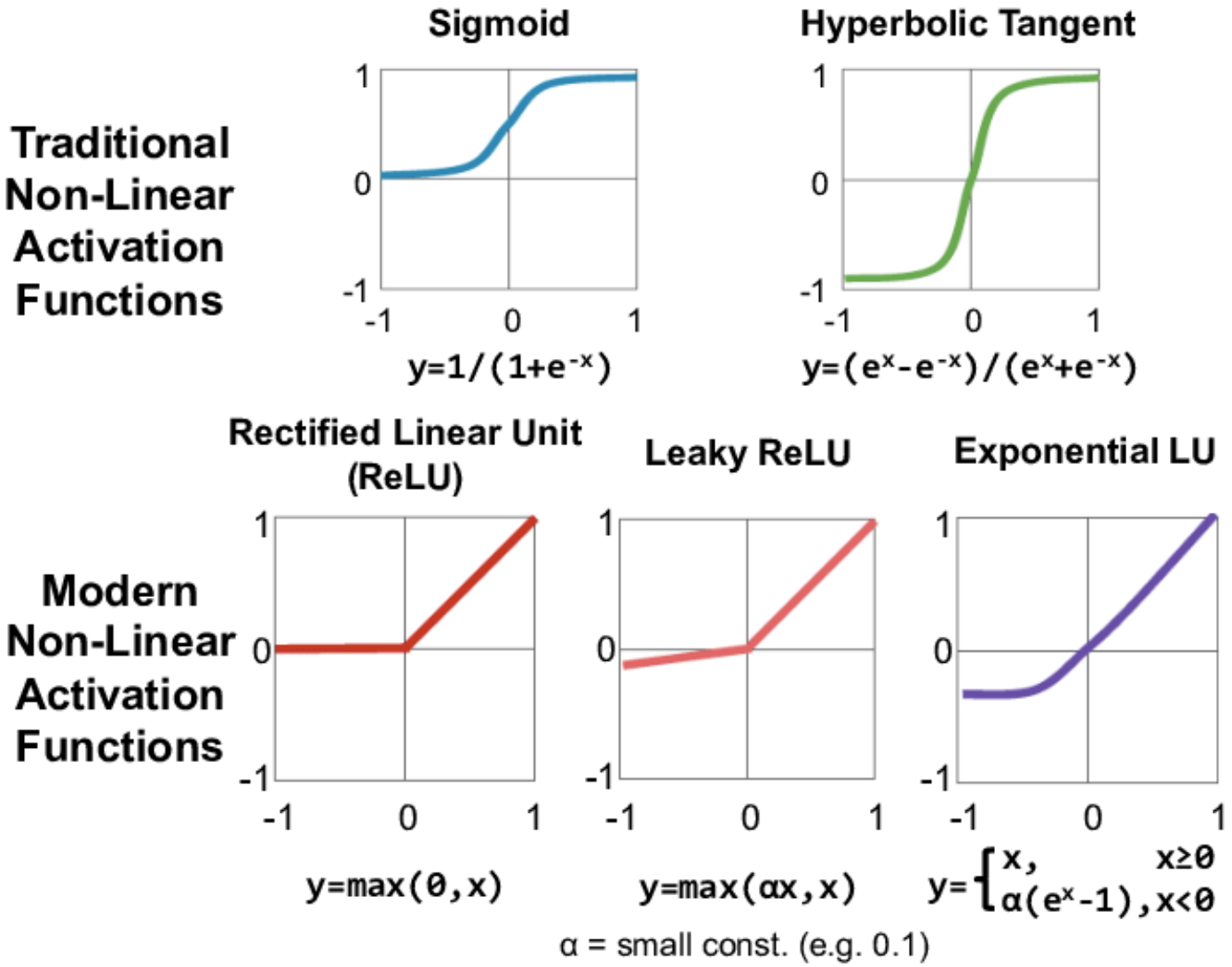
input: i1, i2
hidden layer: h1, h2
output: o1, o2

neth1 = w1 * i1 + w3 * i2 + b1 outh1 = sigmoid(neth1)
neth2 = w2 * i2 + w4 * i2 + b2 outh2 = sigmoid(neth2)

neto1 = w5 * outh1 + w6 * outh2 + b3 **outo1 = sigmoid(neto1)**
neto2 = w7 * outh1 + w8 * outh2 + b4 **outo2 = sigmoid(neto2)**

forward pass를 통해 예측값을 구함!

* Activation function(활성화 함수)



- Activation function을 사용하는 것은 선형성을 깨트리기 위해서임!
- hidden layer의 activation function으로 가장 많이 사용하는 것은 ReLU function
- Output layer의 activation function으로 이진분류시 sigmoid function, 다중분류시 softmax function을 주로 이용

* *Weight Initialization*

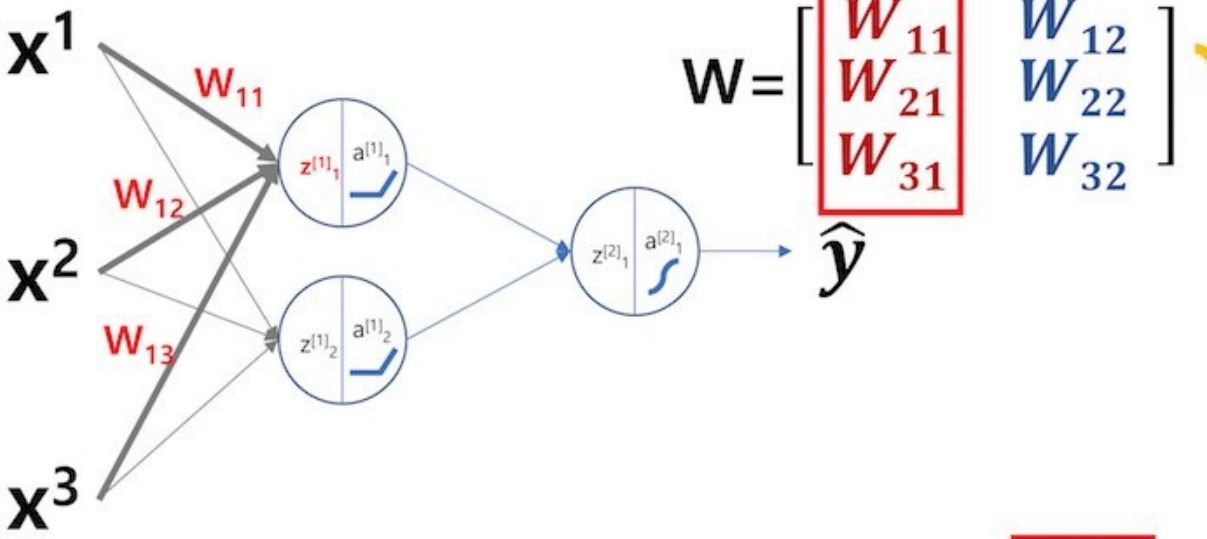
- random initialization: 가중치 값들을 처음에 random하게 initialization을 해주어 임의의 가중치 값에 대한 손실을 계산해서 최적의 가중치 값을 찾아 나감.

- 이 때 적절한 값으로 initialization을 해주는 것이 중요한데, 제일 많이 사용하는 방법은 xavier initialization, he initialization.

- bias는 통상적으로 0으로 initialization하는 것이 효율적.

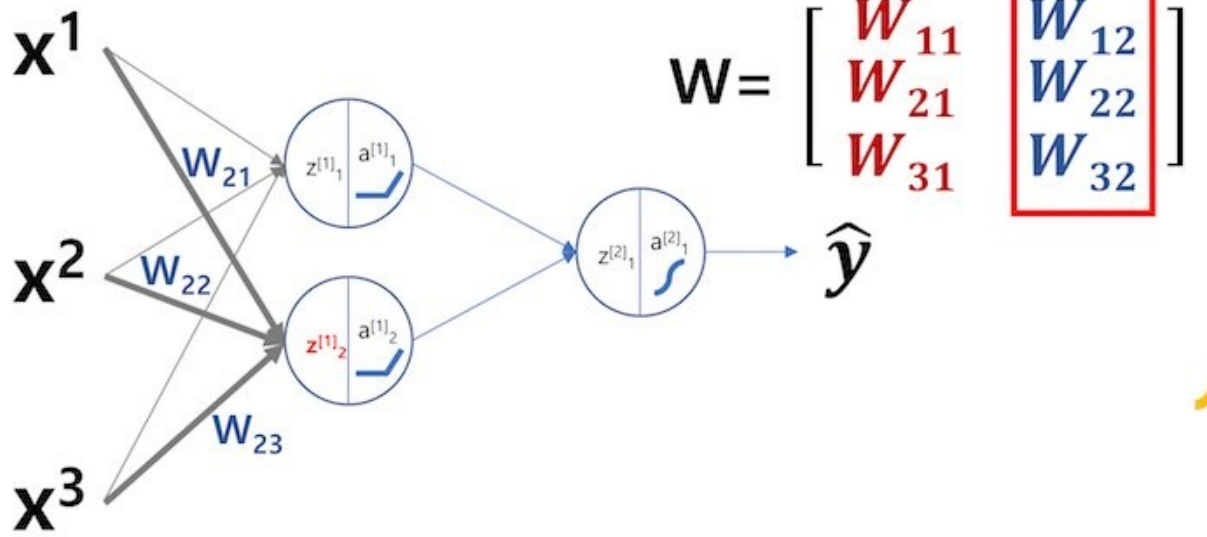
- Weights값은 0으로 초기화 해버리면 대칭성이 생겨버림(똑같은 feature를 뽑아낸다고 생각하면 됨) -> random initialization이 중요!!

Vectorization



$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix}$$

$\hat{y}$



$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix}$$

$\hat{y}$

$$= \begin{bmatrix} Z^{[1]}_1 \\ Z^{[1]}_2 \end{bmatrix}$$

$$= \begin{bmatrix} W_{11}X_1 + W_{21}X_2 + W_{31}X_3 + b \\ W_{12}X_1 + W_{22}X_2 + W_{32}X_3 + b \end{bmatrix}$$

$$= \begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} + b$$

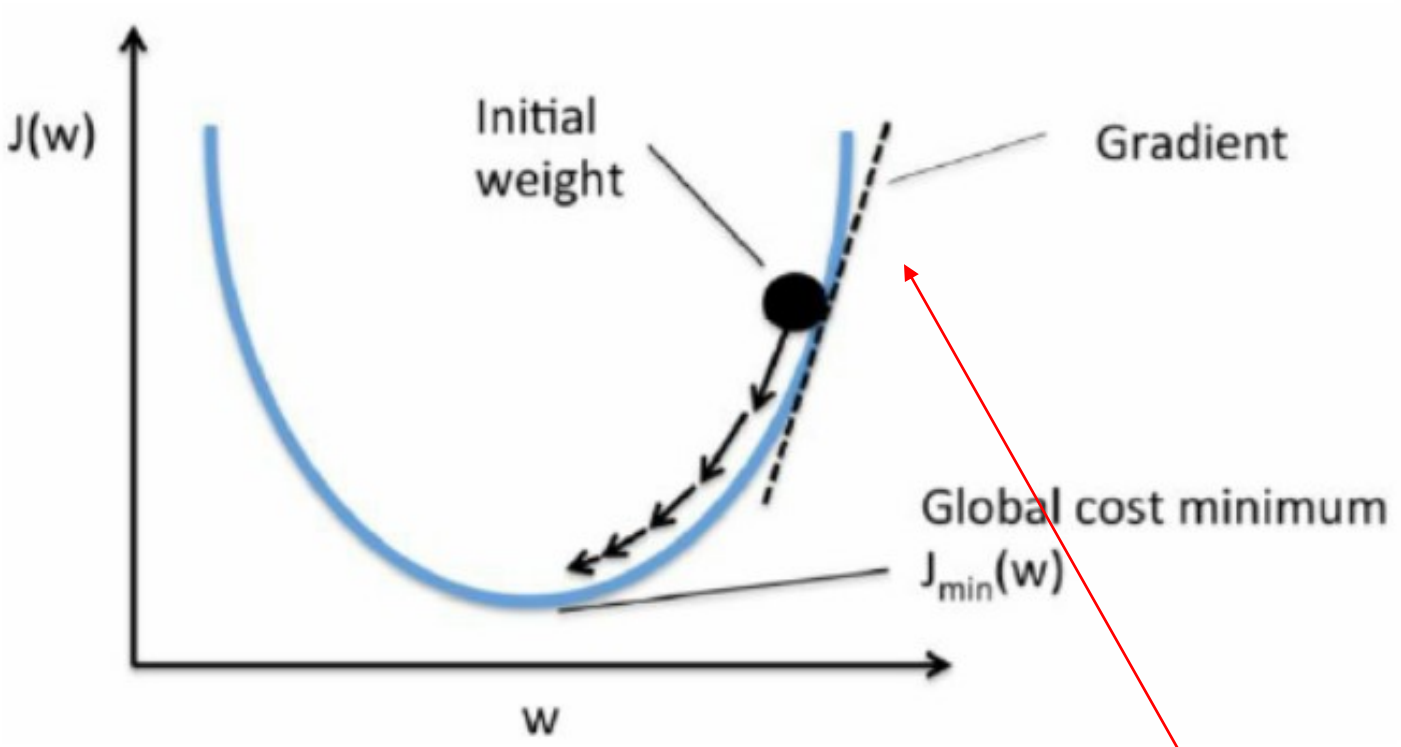
$$= W^T X + b$$

Caculation of the loss

Mean squared error를 loss function으로 이용(예시일 뿐, 학습에 따라 다름!)

Loss = $\frac{1}{2}$ (targeto1 $-$ outo1)^2 + $\frac{1}{2}$ (targeto2 $-$ outo2)^2
→ 하나의 데이터에 대한 Loss

Total Loss =$\sum_{i=1}^{n}\{\frac{1}{2}$ (targetoi1 $-$ outoi1)^2 + $\frac{1}{2}$ (targetoi2 $-$ outoi2)$^2\}$

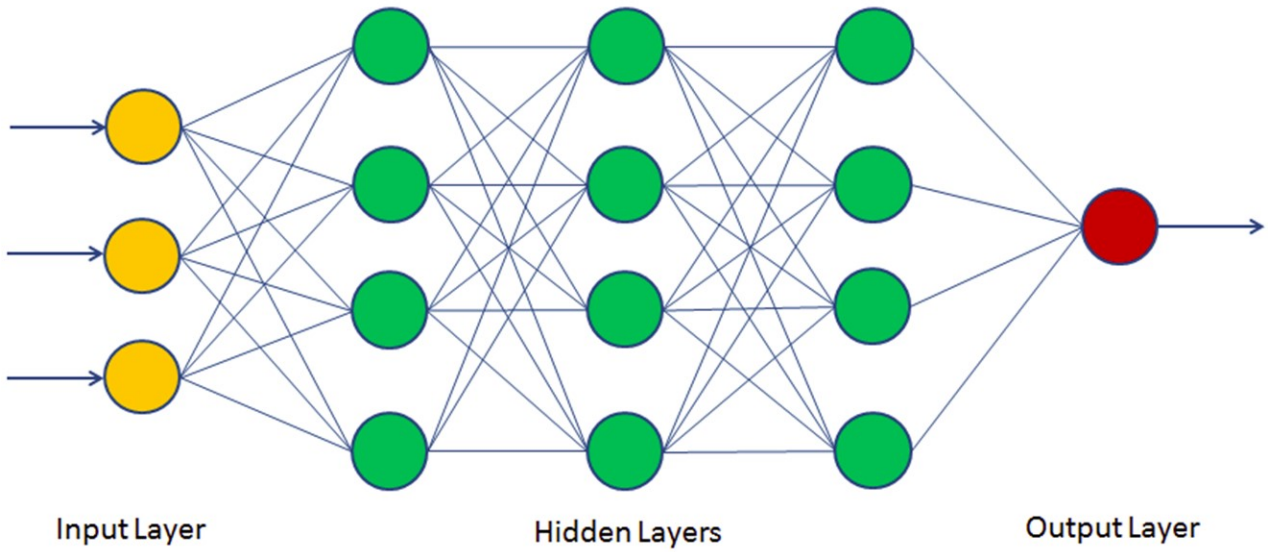


Forward propagation 을 이용해 Loss function에서 초기의 가중치를 이용한 loss 를 구해 보았음.
-> loss를 minimize 하는 가중치의 update 과정을 거쳐 최적의 가중치를 찾아야함.

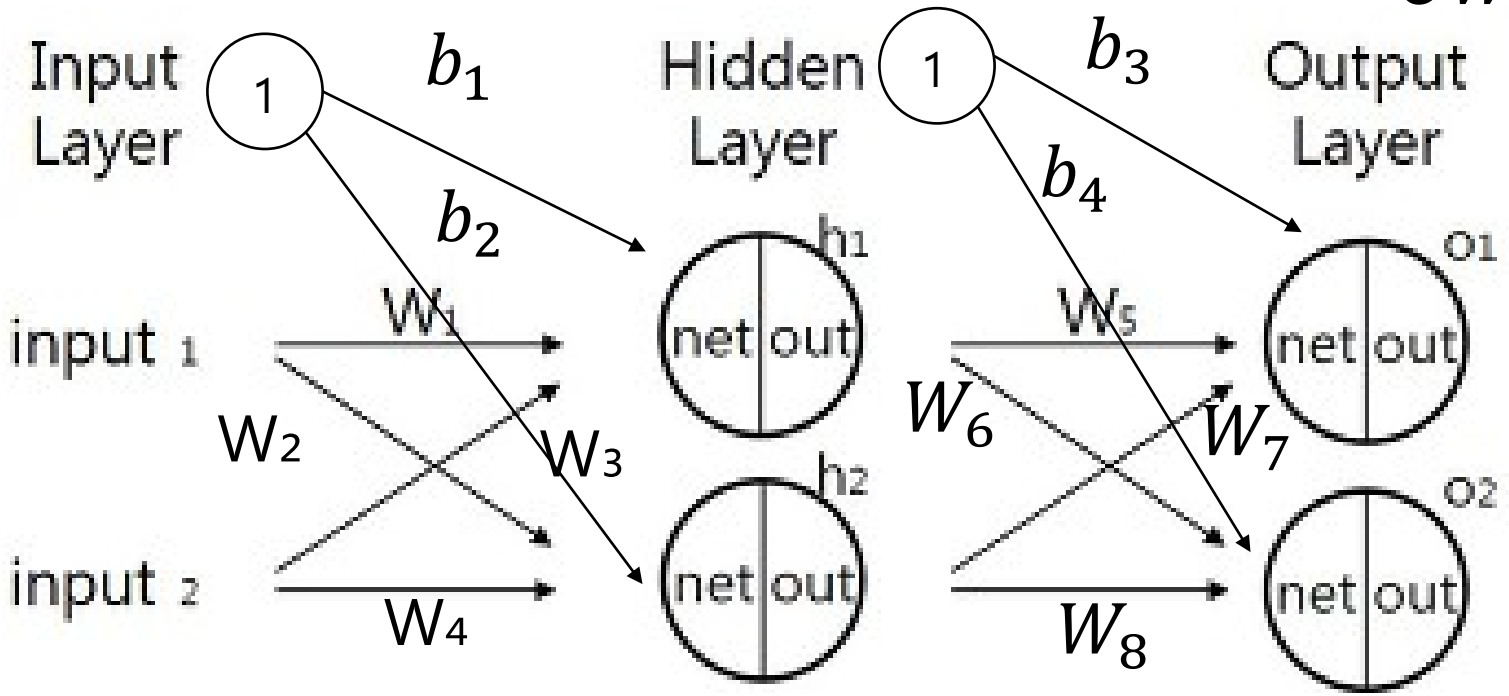
가중치의 update 방향을 나타내는 경사도를 계산하는 Backward propagation 연산이 필요함!!!!!!

2. Backward propagation

- 각 가중치 값들에 대한 loss의 미분값을 연산하는 과정.
- 가중치에 대한 미분값을 구하기 위해서 chain rule을 이용함!



Caculation of the Gradient of $W_6(\frac{\partial Loss}{\partial W_6})$



neto1 = w5 * outh1 + w6 * outh2 + b3

outo1 = sigmoid(neto1)

Loss = $\frac{1}{2}$ (targeto1 − outo1)² +

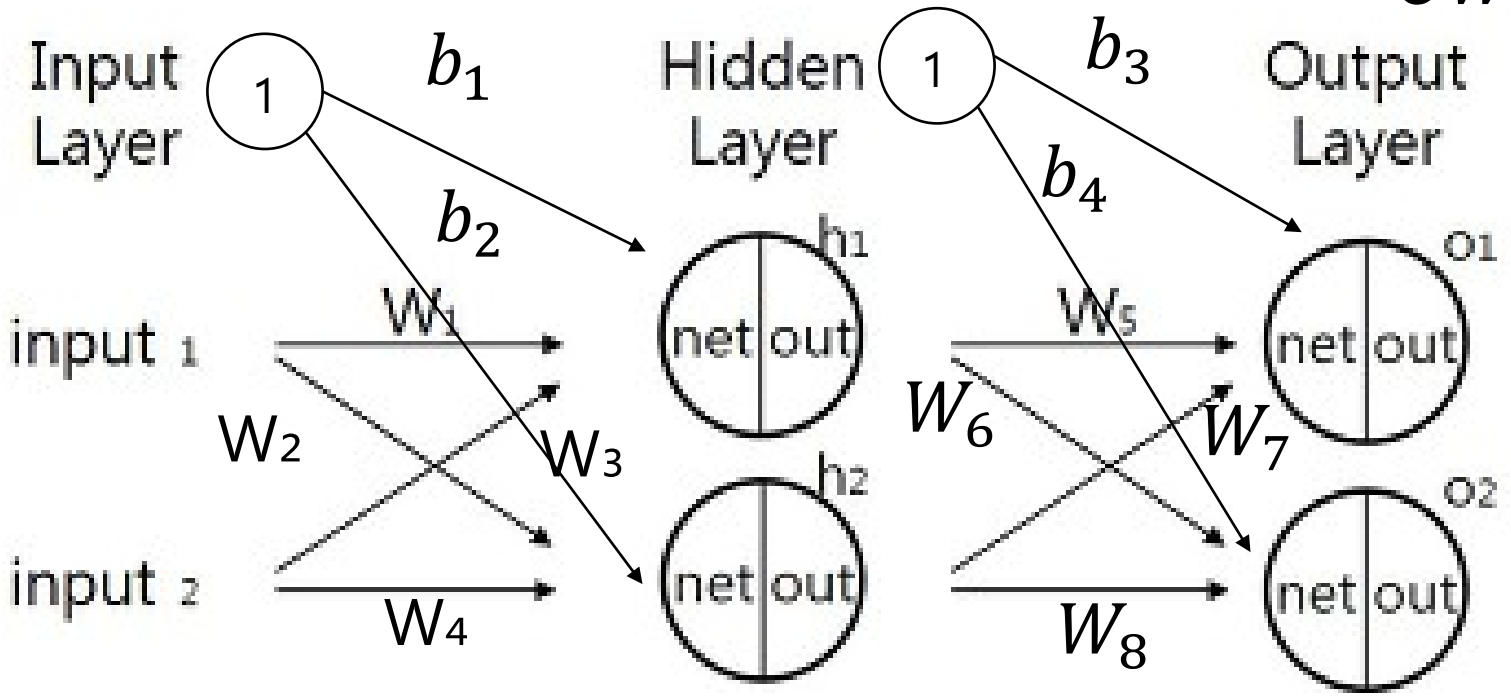
$\frac{1}{2}$ (targeto2 − outo2)²

$$\frac{\partial Loss}{\partial W_6}$$

$$= \frac{\partial Loss}{\partial outo1} * \frac{\partial outo1}{\partial neto1} * \frac{\partial neto1}{\partial W_6}$$

= {-(targeto1 − outo1)} * {outo1*(1-outo1)} * outh2

Caculation of the Gradient of $W_1(\frac{\partial Loss}{\partial W_1})$



neth1 = w1 * i1 + w3 * i2 + b1 outh1 = sigmoid(neth1)

neto1 = w5 * outh1 + w6 * outh2 + b3 outo1 = sigmoid(neto1)

neto2 = w7 * outh1 + w8 * outh2 + b4 outo2 = sigmoid(neto2)

$$\frac{\partial Loss}{\partial W_1}$$

$$= \frac{\partial Loss}{\partial outo1} * \frac{\partial outo1}{\partial neto1} * \frac{\partial neto1}{\partial outh1} * \frac{\partial outh1}{\partial neth1} * \frac{\partial neth1}{\partial W_1} + \frac{\partial Loss}{\partial outo2} * \frac{\partial outo2}{\partial neto2} * \frac{\partial neto2}{\partial outh1} * \frac{\partial outh1}{\partial neth1} * \frac{\partial neth1}{\partial W_1}$$

3. Update weights until convergence

$$w_{new} = w - \alpha * \frac{\delta L}{\delta w}$$

$$b_{new} = b - \alpha * \frac{\delta L}{\delta b}$$

$$\text{Repeat Until Convergence } \{$$

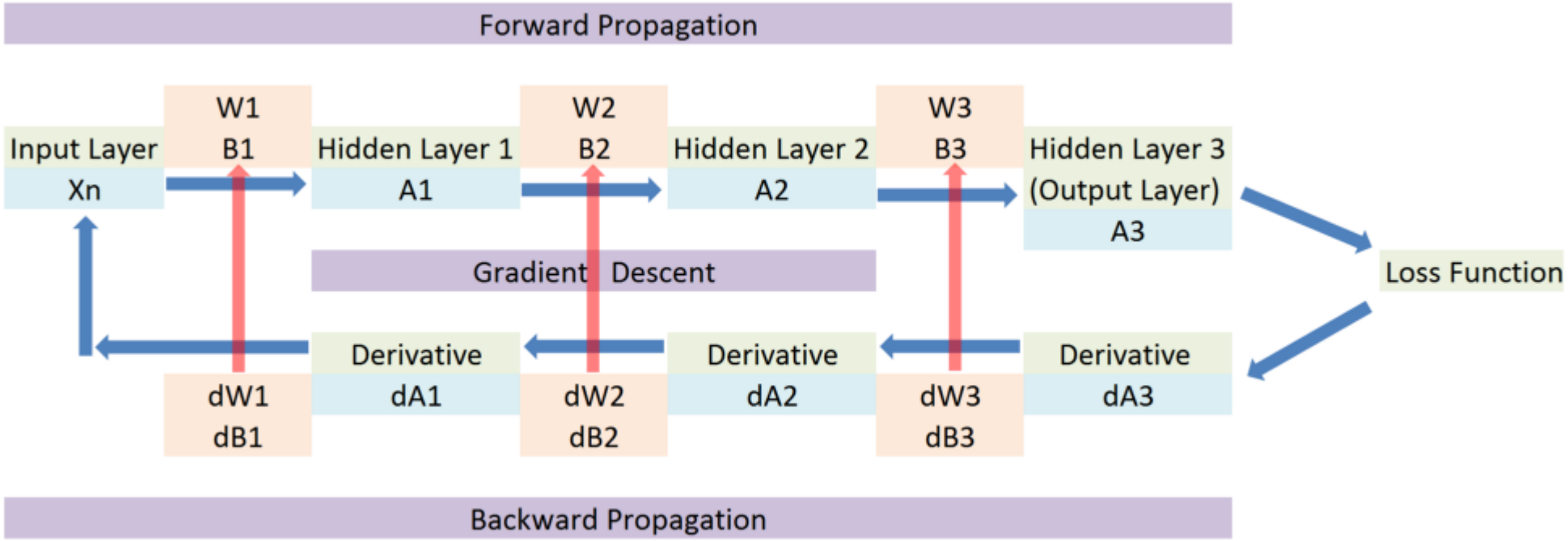
$$\quad \text{for i} = 1...\text{m } \{$$

$$\quad\quad \omega \leftarrow \omega - \alpha * \nabla_w L_m(w)$$

$$\quad \}$$

$$\}$$

4. 전체 summary



1. 가중치와 편향을 초기화
2. Input 값을 이용하여 forward pass를 통해 예측값과 loss 계산
3. 주어진 loss에 대한 gradients를 backward pass를 이용해 계산
4. Gradients들을 이용하여 가중치와 편향을 업데이트
5. Loss가 수렴할 때까지 2-4 반복

5. 알아두면 좋을 것들

- Neural network를 이용한 학습에서 과적합 방지 목적으로 ***dropout**을 이용함

- Neural network 구조에서 hidden layer의 수, 뉴런의 개수, 가중치 업데이트 시 사용할 learning rate 모델링 시 직접 정해야 하는 ***hyper parameter**라고 부름.

- 딥러닝 training에서는 일반적으로 데이터를 분할하여 학습하는 ***batch trainig**을 실시함

더 공부하고 싶다면!

- Coursera: <deeplearnig.ai> Neural Networks and Deep Learning
- 모두를 위한 딥러닝 (유튜브, sung kim)
- 연세대학교 박재우 교수님 딥러닝 강좌(강추!)