학습 관련 기술

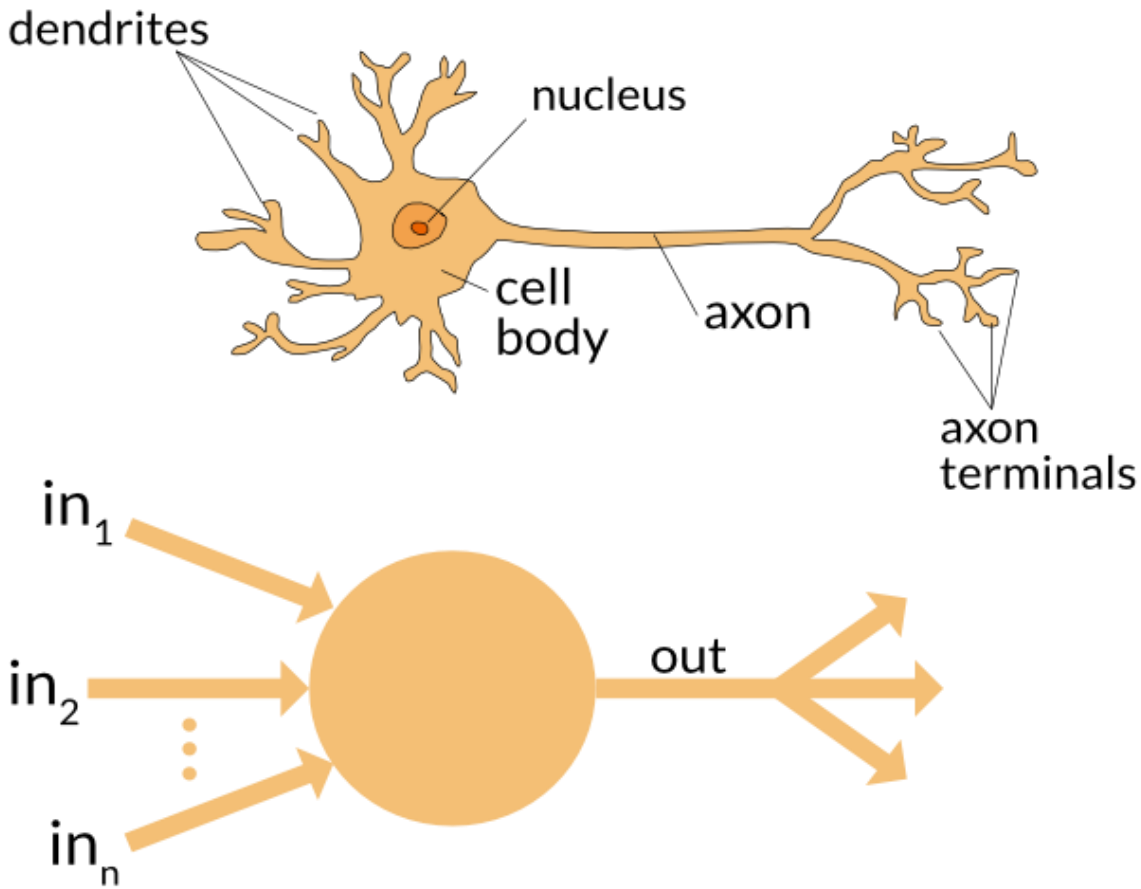
HYU AILAB SEMINAR SEASON11

석사3기 조충현

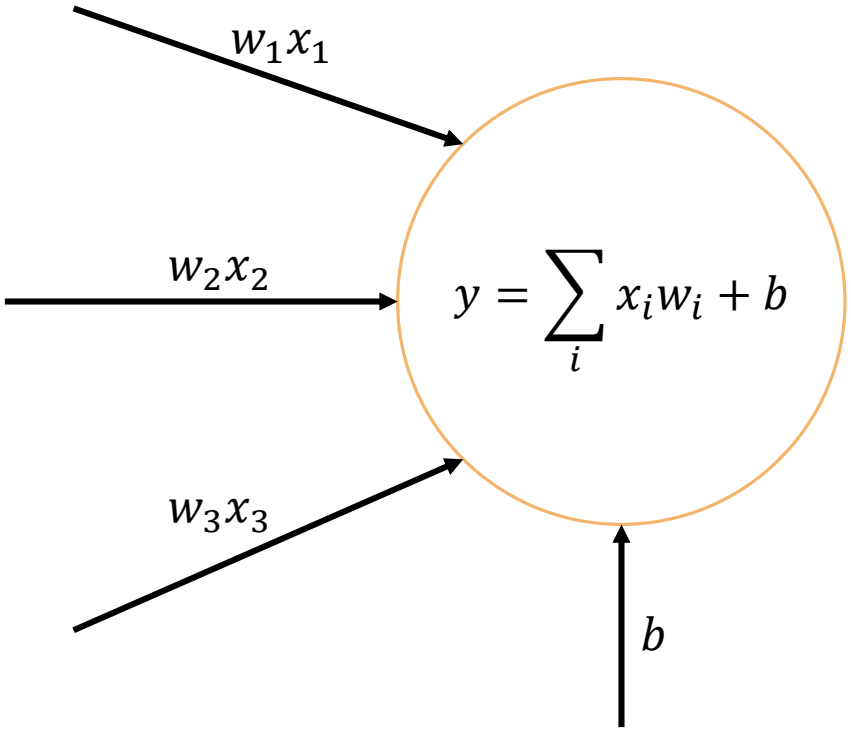
목차

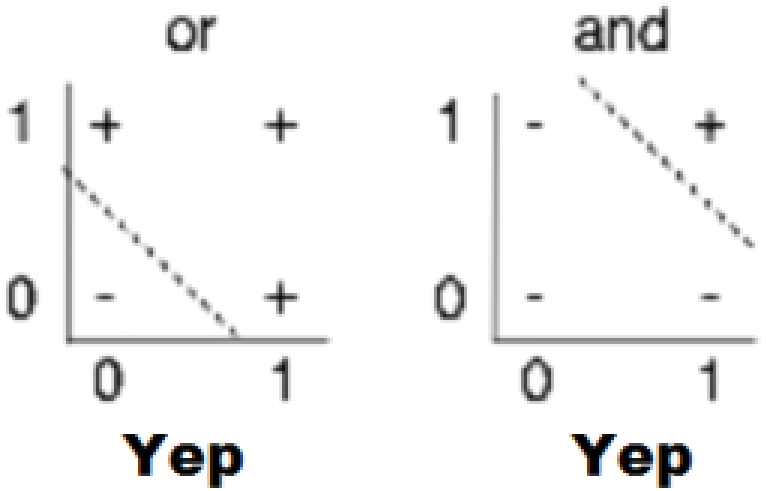
퍼셉트론

뇌의 뉴런을 모방한 구조



단일 퍼셉트론

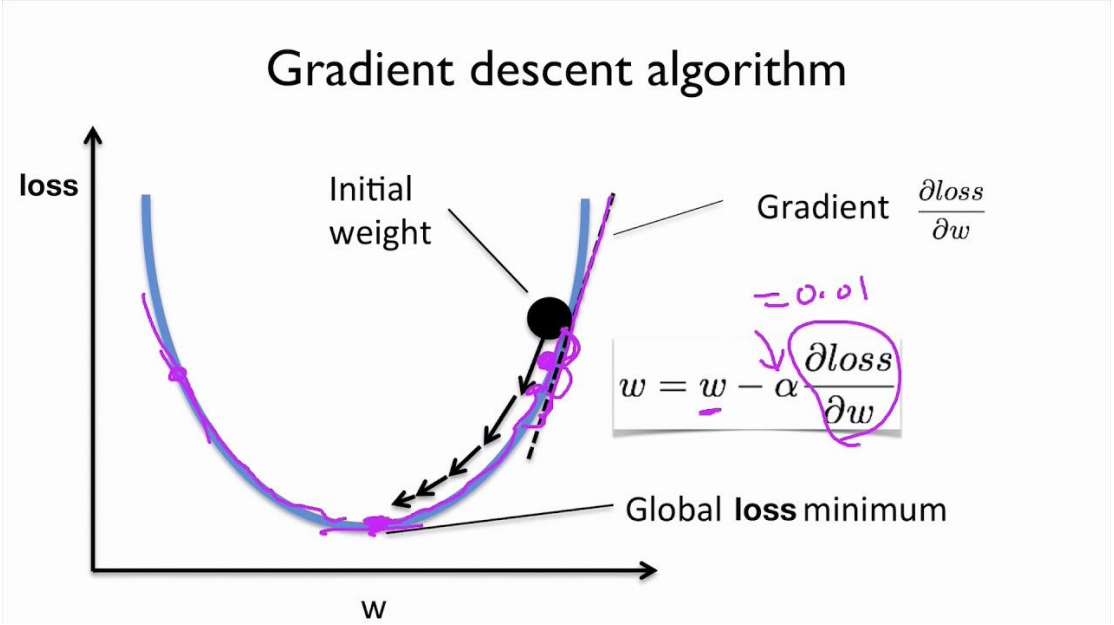




학습 알고리즘 – Delta Rule

출력 값이 원하는 목표 값(d)과 가까워지도록 가중치를 Gradient Descent(경사하강법)를 이용하여 조정

1. W(가중치)와 b(바이어스)를 임의의 값으로 초기화
2. 하나의 학습 벡터에 대한 출력 값 계산
3. 목표 값과 비교해사 허용 오차보다 크면 학습



파라미터(W, b) 조정

출력 값과 목표 값의 차이로 W(가중치)와 b(bias) 조정

$W_i(t+1) = W_i(t) + \alpha[d(t) - y(t)] \cdot X_i(t)$

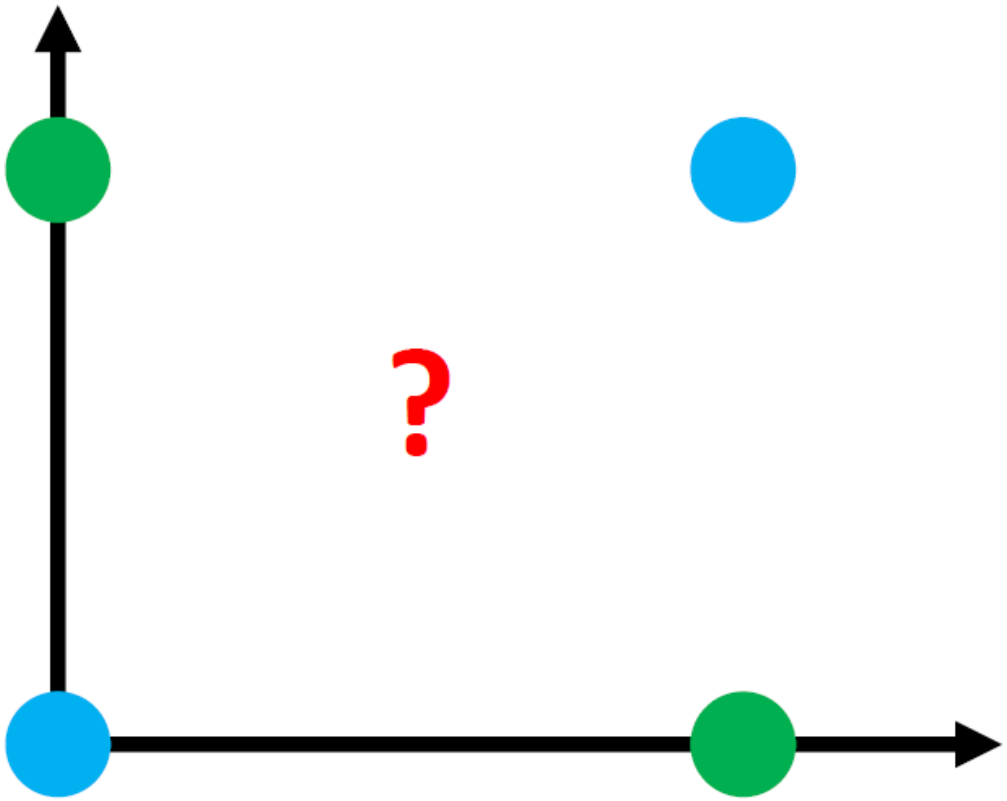
$b = b + \alpha[d(t) - y(t)] \cdot X_i(t)$

$\alpha =$ learning rate

$d(t) =$ 목표 값

$y(t) =$ 결과 값

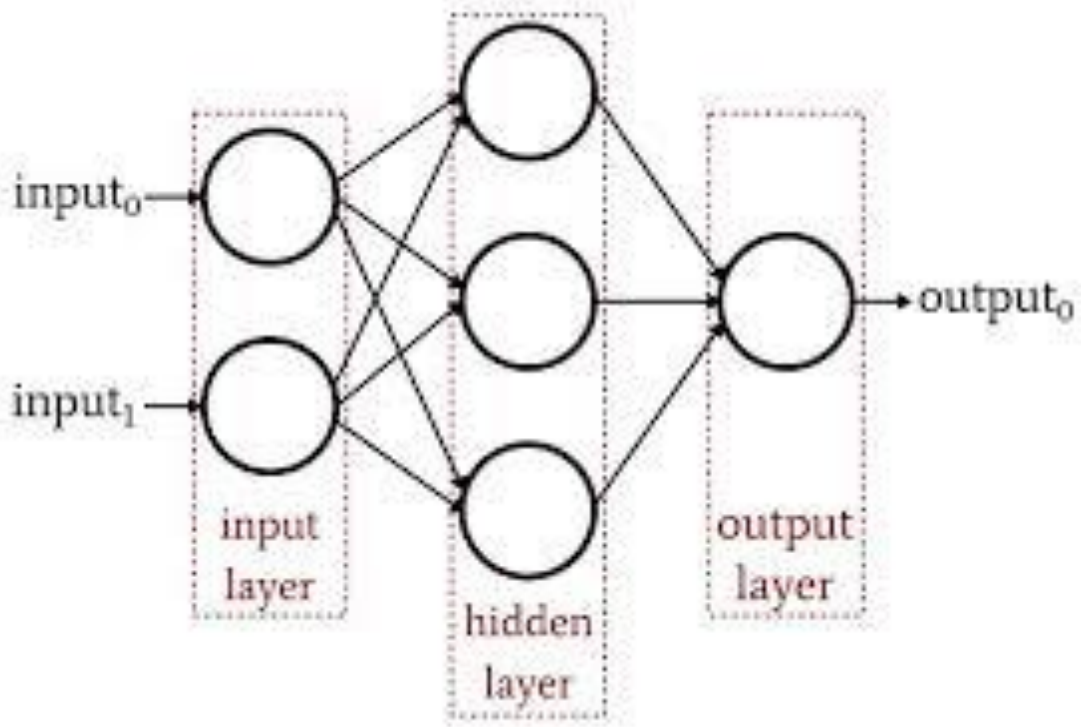
XOR?



다층 퍼셉트론

입력층과 출력층 사이에 1개 이상의 은닉층(hidden layer) 추가

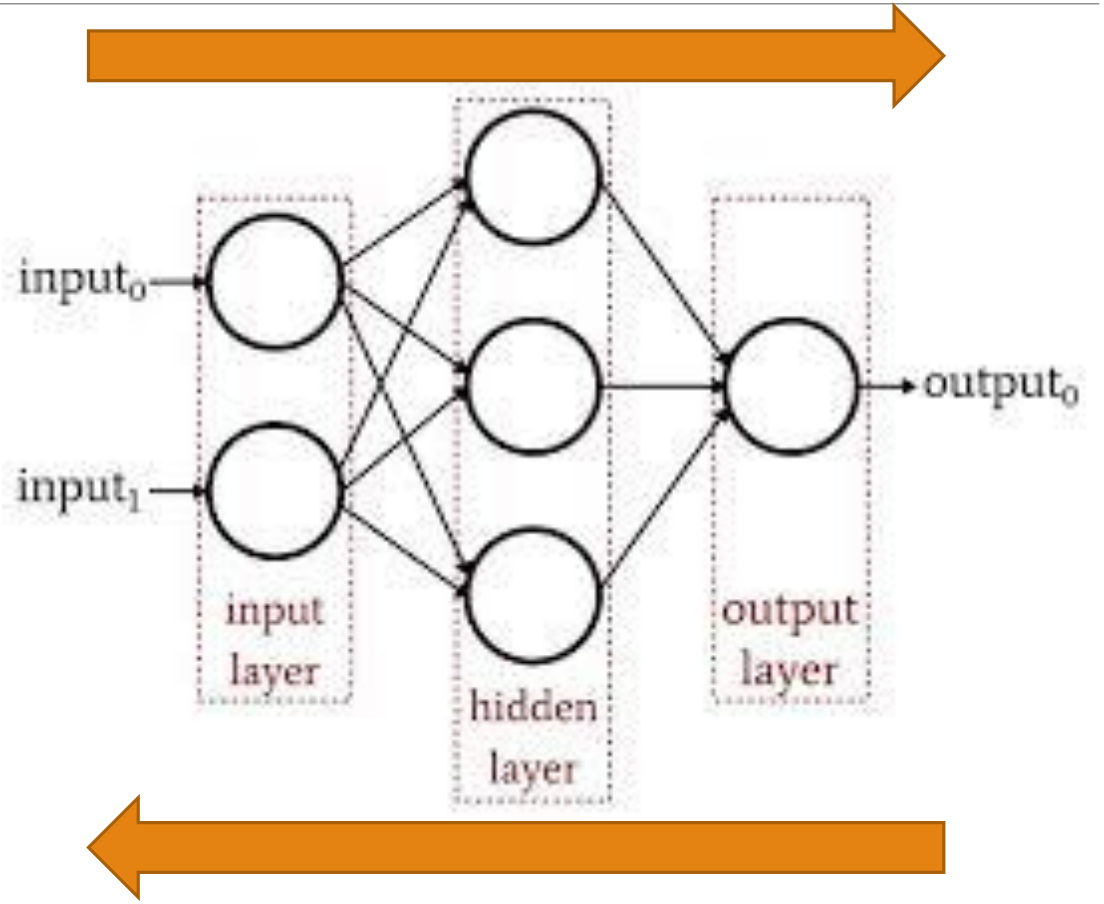
역전파 알고리즘을 이용하여 학습



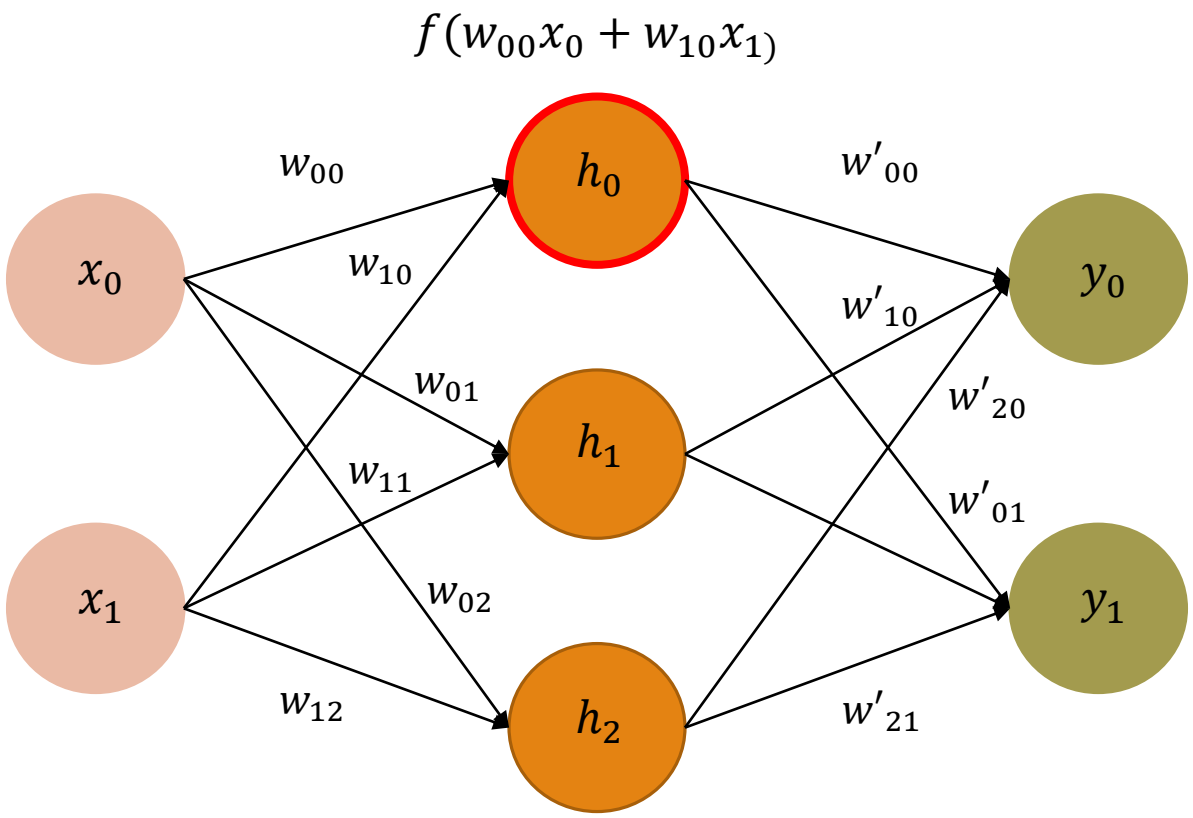
학습 알고리즘

1. 초기화 (initialization)
2. 순전파 (forward propagation)
3. 역전파 (backward propagation)

-> 2,3 을 오차 허용 범위까지 반복



순전파(forward propagation)



역전파 (backpropagation)

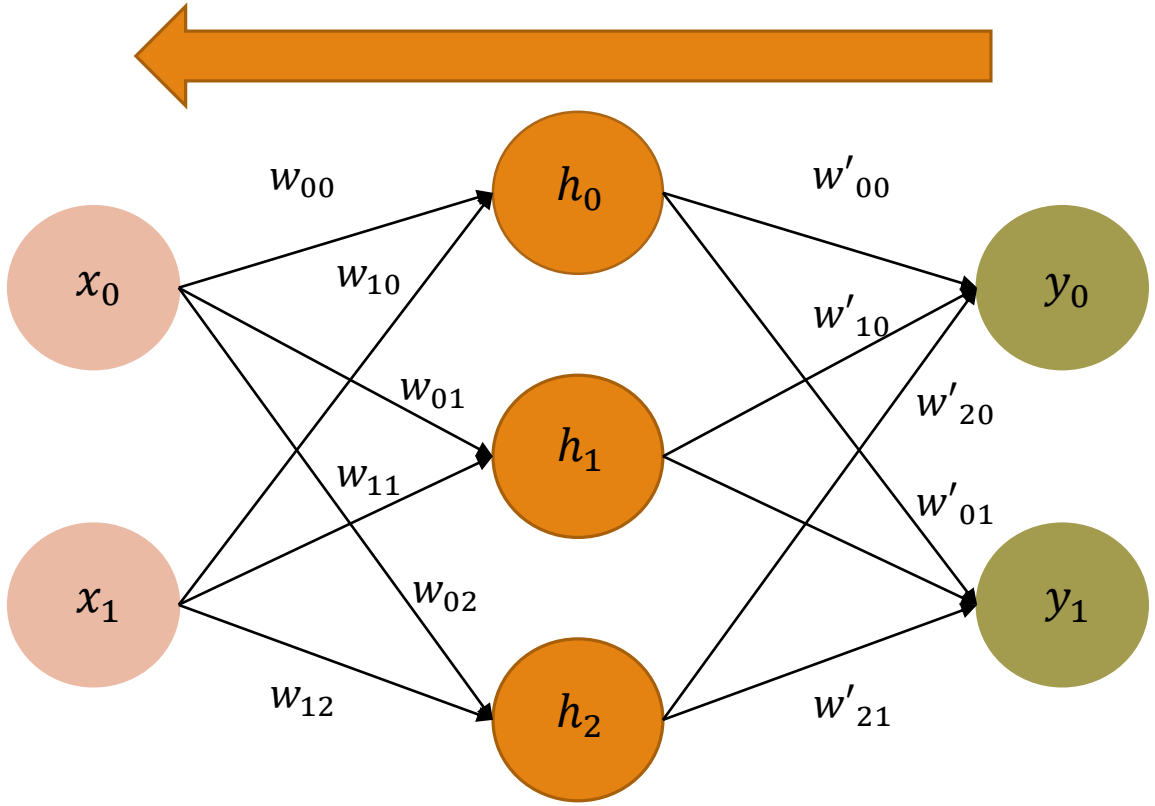
목표 : error을 줄이는 것 (loss function)

각 layer간의 weigh에 대한 loss functio의 gradient 를 weight 반대 방향으로 더해준다.

$$w = w - \eta * \frac{\partial E}{\partial w}$$

η : learning rate

$\frac{\partial E}{\partial w}$: w의 변화에 따른 E의 변화



역전파 (backpropagation)

$$E = \frac{1}{2}(y - y')2 \qquad w_2 = w_2 - \eta * \frac{\partial E}{\partial w_2}$$

$y' = desired\ value\ (training\ data)$

$$\frac{\partial E}{\partial w_2} = (y - y')\frac{\partial y}{\partial w_2}$$

$y = sig(w_2 h)$

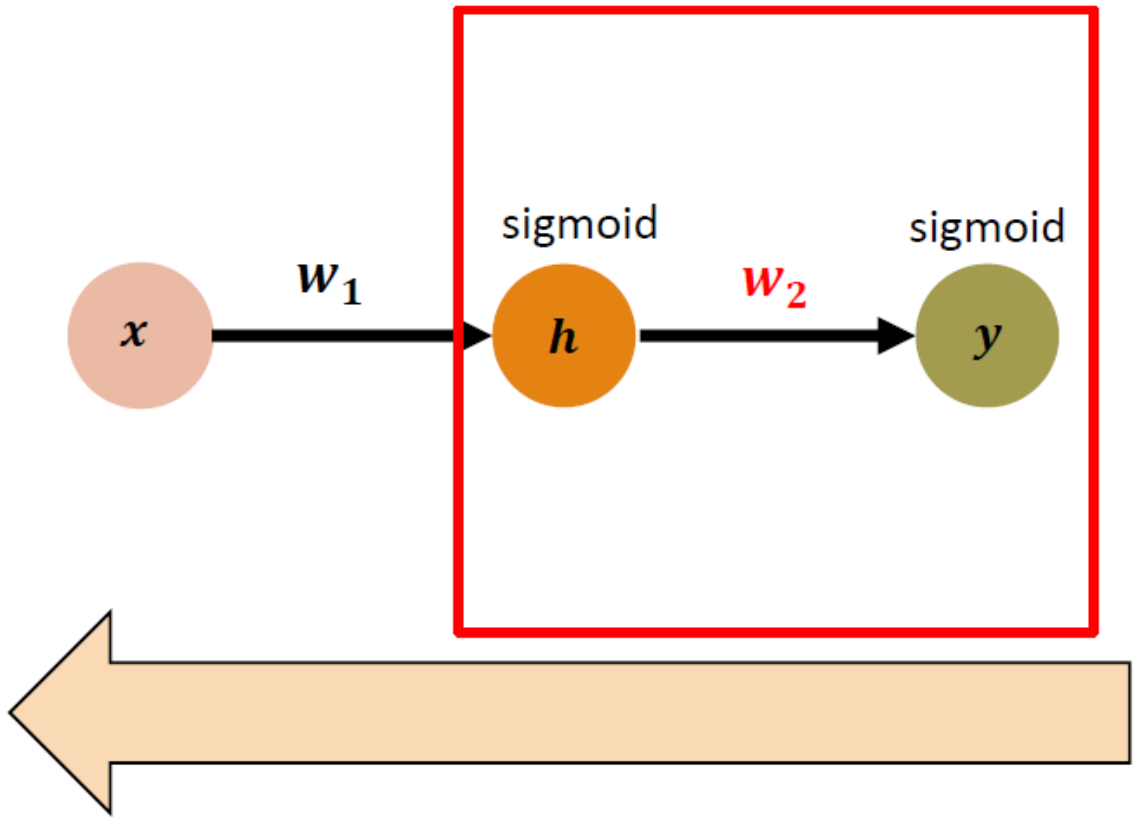
$$= (y - y')\frac{\partial}{\partial w_2} sig(w_2 h)$$

$sig(x)' = sig(x)(1 - sig(x))$

$$= (y - y')\, sig(w_2 h)(1 - sig(w_2 h))\frac{\partial w_2 h}{\partial w_2}$$

$$= (y - y')\, y(1 - y)\frac{\partial w_2 h}{\partial w_2}$$

$$= (y - y')y(1 - y)h$$



역전파 (backpropagation)

$$E = \frac{1}{2}(y - y')2 \qquad w = w_1 - \eta * \frac{\partial E}{\partial w_1}$$

$y' = desired\ value$

$$\frac{\partial E}{\partial w_1} = (y - y')\frac{\partial y}{\partial w_1}$$

$y = sig(w_2 h)$

$$= (y - y')\frac{\partial}{\partial w_1} sig(w_2 h)$$

$sig(x)' = sig(x)(1 - sig(x))$

$$= (y - y')\, y(1 - y)\frac{\partial w_2 h}{\partial w_1}$$

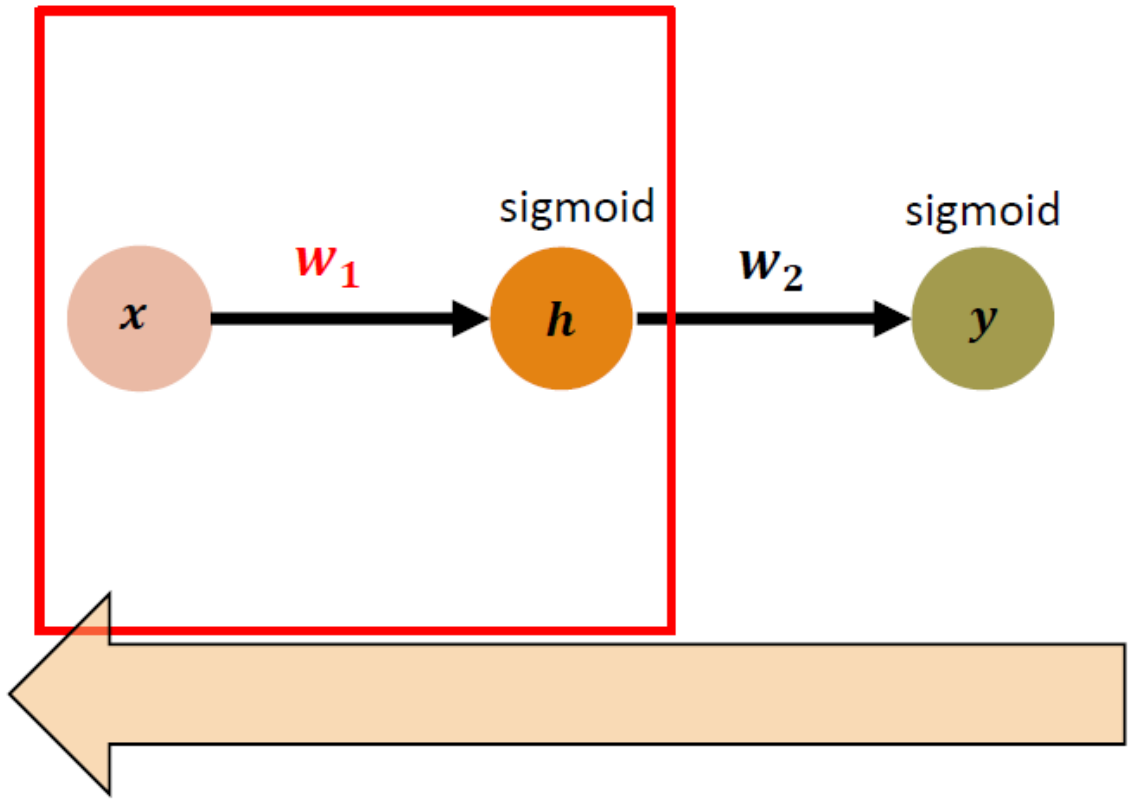
w_2 는 w_1 입장에서 상수

$$= (y - y')y(1 - y)w_2\frac{\partial h}{\partial w_1}$$

$h = sig(w_1 x)$

$$= (y - y')y(1 - y)w_2\frac{\partial}{\partial w_1} sig(w_1 x)$$

$$= (y - y')y(1 - y)w_2 h(1 - h)x$$



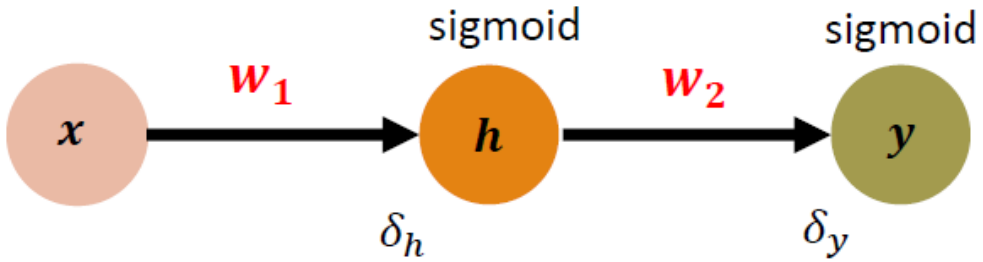
역전파 (backpropagation)

$$E = \frac{1}{2}(y - y')2 \qquad w = w - \eta * \frac{\partial E}{\partial w}$$

$$\frac{\partial E}{\partial w_2} = \underbrace{(y - y')y(1-y)}_{\delta_y} h$$

$$\frac{\partial E}{\partial w_1} = \underbrace{(y - y')y(1-y)w_2 h(1-h)}_{\delta_h} x$$

$$\delta_h = \delta_y \; sig(w_1 x)' \; w_2$$

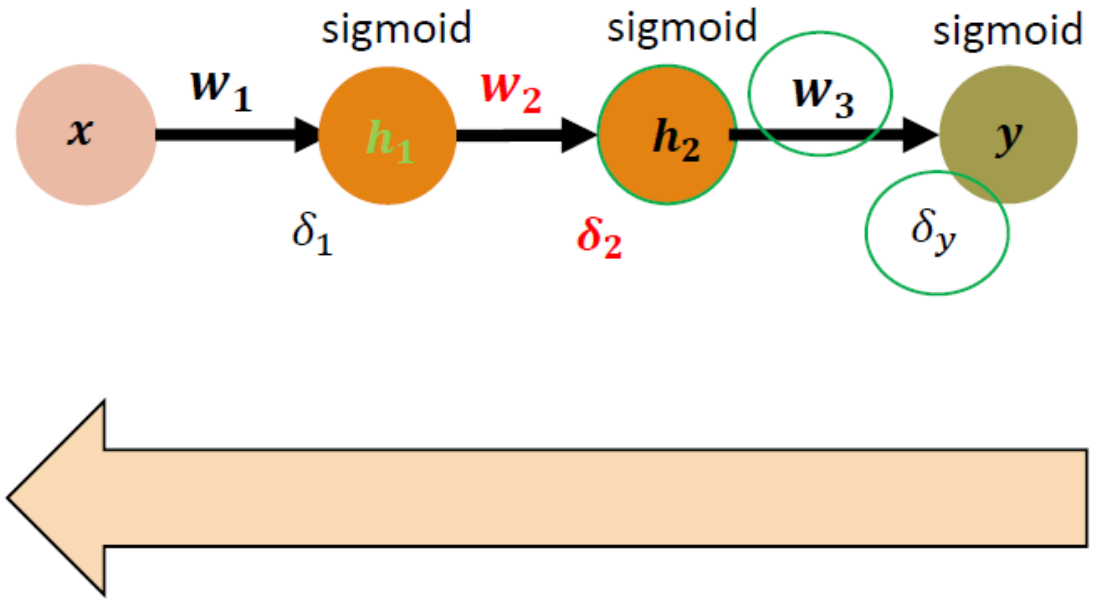


역전파 (backpropagation)

$\delta_2 = \delta_y \;\; sig(w_2 h_1)' \;\; w_3$

$\frac{\partial E}{\partial w_2} = \delta_2 h_1$

$w_2 = w_2 - \eta * (\delta_2 h_1)$



활성 함수 (Activation function)

네트워크에 비선형성을 추가하기 위해 사용됨

Why?

- H(X) = f(g(..(X))) -> H(X) = AX -> 아무리 층이 많아도 선형함수

Activation function 추가

- 각 층의 출력 결과를 비선형화 하고 다음 층으로 넘겨준다

단층 퍼셉트론에서는 출력값 영역을 나누는 선이 직선 형태로 표현되기 때문에 XOR을 표현할 수 없다.

그래서 층을 하나 더 쌓아서 `(input -> NAND,OR -> AND -> output)` XOR을 표현하게 되는데, 이게 가능했던 이유는 활성화 함수로 비선형 함수인 계단 함수를 사용해 결과값의 영역을 나누는 선이 곡선으로 표현될 수 있었기 때문이다.

이렇게 기본적인 연산을 조합해 더욱 복잡한 연산을 표현하기 위해서는 비선형 함수가 필요하다.

활성 함수 종류

활성 함수에 따라 학습 성능이 달라진다

step function

sigmoid

hyperbolic tangent

ReLU

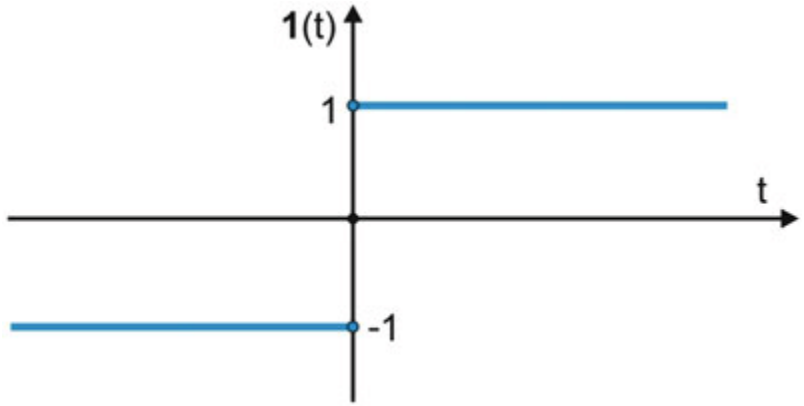
softmax

…

활성 함수 – step function

$$step(x) = \begin{cases} 1 & x \geq 0 \\ 0 & otherwise \end{cases}$$

임계 값 기준으로 활성화 되거나 혹은 비활성화 되는 형태

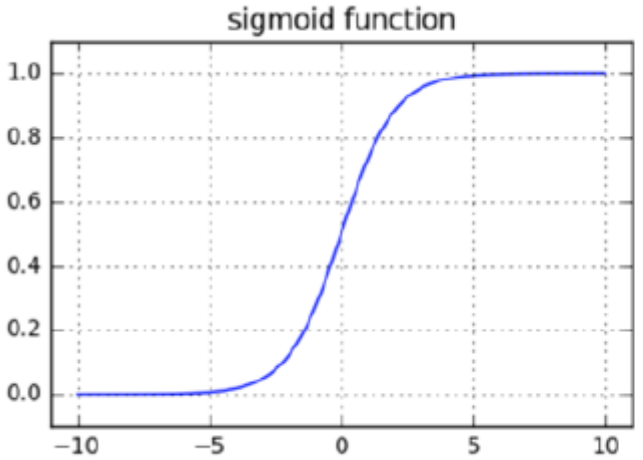


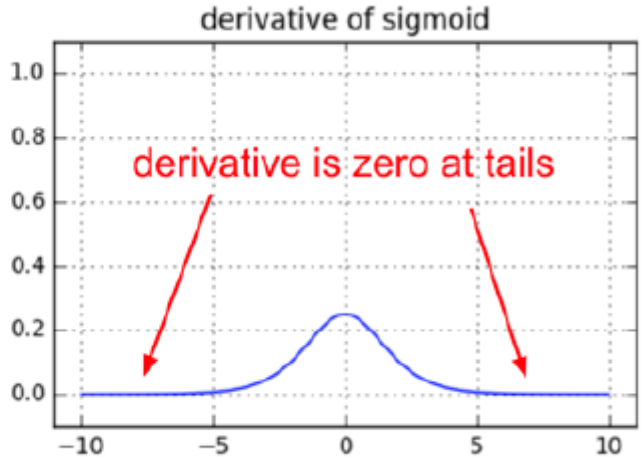
활성 함수 - sigmoid

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

결과값이 [0,1] 사이에서 나타남

뇌의 뉴런과 유사하여 많이 쓰였다





단점

- 계산의 복잡성
- 중심값이 0이 아니라 0.5이다
- gradient vanishing 문제
  - 역전파 될때 gradient가 0에 가까워 지는 현상

활성 함수 – hyperbolic tangent

$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} = 2sigmoid(2x) - 1$

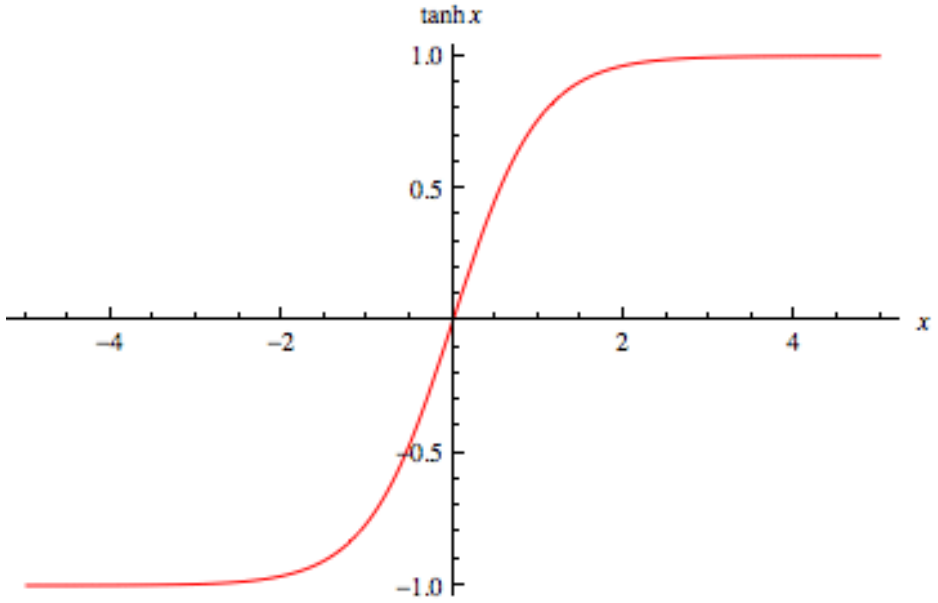
결과값 범위 (-1, 1)

결과값 중심이 0이다.

sigmoid 확장 버전

sigmoid 보다 더 빨리 수렴

(미분계수 최대값이 sigmoid의 4배)



단점

- 동일한 Gradient Vanishing 문제

활성 함수 - ReLU

$$ReLU(x) = \begin{cases} x & x \geq 0 \\ 0 & otherwise \end{cases}$$

0에서 꺾이기 때문에 비선형

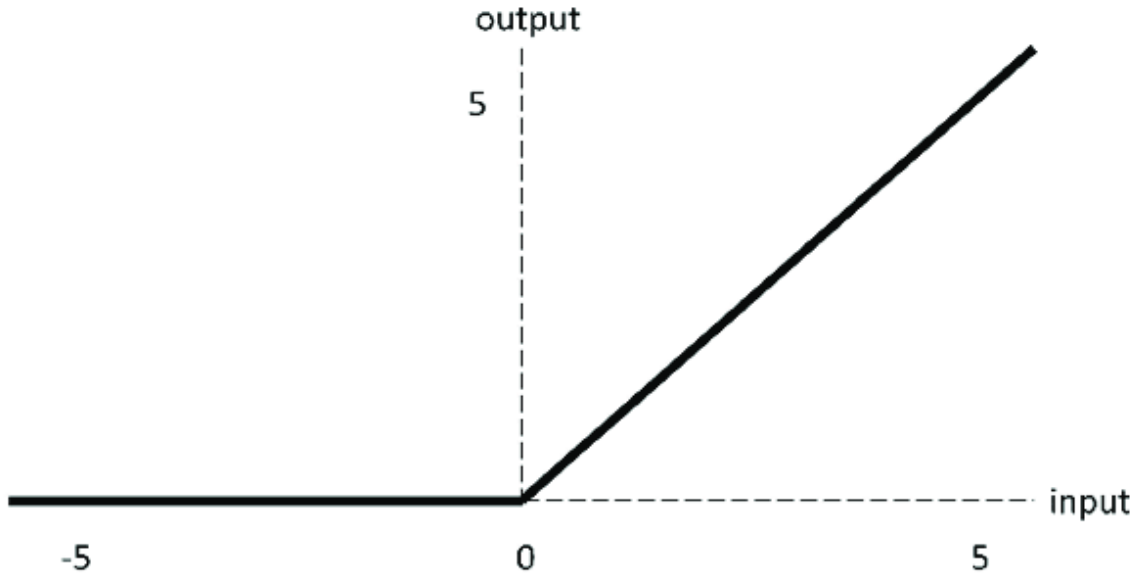
계산이 매우 효율적

sigmoid 보다 계산 속도가 빠르다

But. 음수가 나온 노드는 학습이 불가능

-> 간단한 네트워크에서는 성능 저하

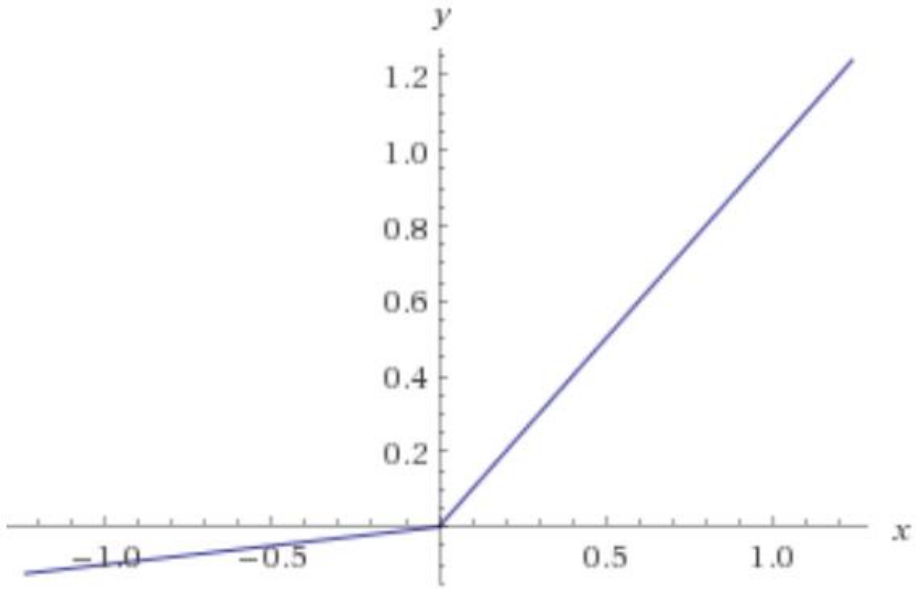
-> 0 에서 미분 불가능



활성 함수 - Leaky ReLU

$$Leaky\ ReLU(x) = \begin{cases} x, & x \geq 0 \\ 0.01x, & otherwise \end{cases}$$

ReLU 함수의 변형으로 음수에 대해서도 학습 가능



활성 함수 - softmax

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad j = 1,2,\dots,K$$

입력 받은 값을 출력으로 0~1 사이의 정규화 된 값

출력 값들의 총합은 항상 1

주로 출력 노드에서 사용

분류, 강화 학습에 사용된다

softmax는 다른 뉴런의 출력 값들 과의 상대적인 비교를 통해 출력이 결정

Loss function

Backpropagation의 기본 개념은 정답과 현재 예측 값의 차이를 통해 기울기를 계산, network의 학습에 사용하는 것

Loss function은 그 차이를 어떻게 계산할 것인지 결정하는 함수

의미로는 model이 현재 실제 값에 비해 얼마나 잘 예측하는지를 수치화한 것

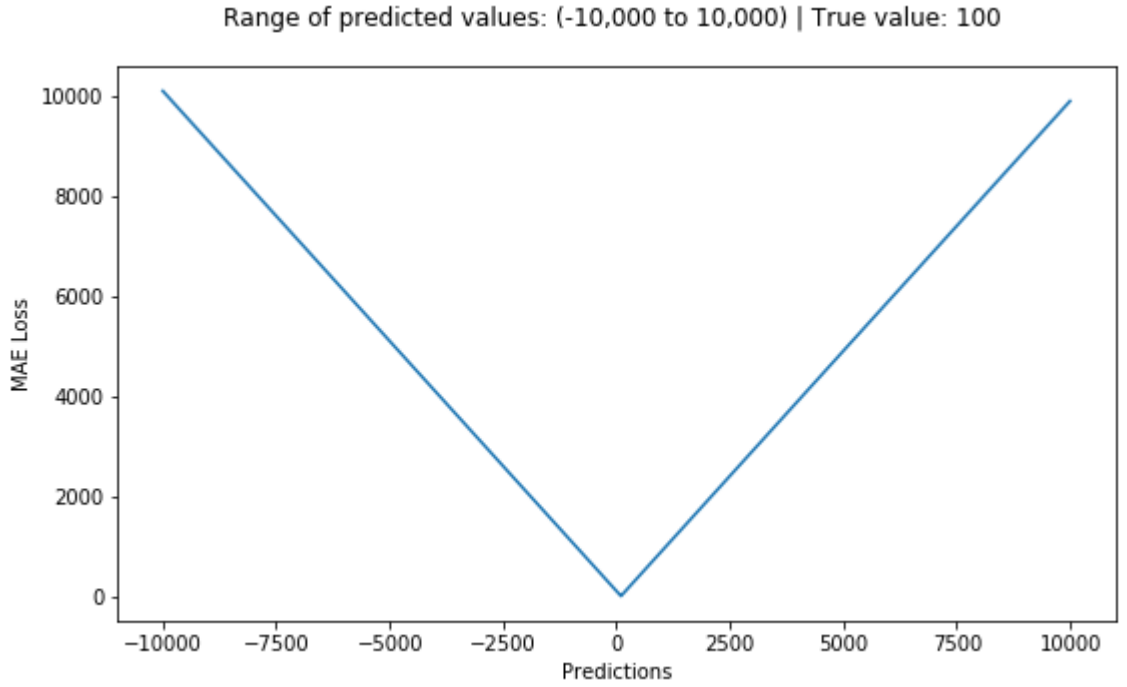
Ex) 실제 값: 500 / 모델이 예측한 값 : 350 … Loss? 500 – 350 = 150

Common Loss functions

Mean Absolute Error (MAE)

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$

- 실제 값에서 예측 값을 뺀 것의 절대값의 평균

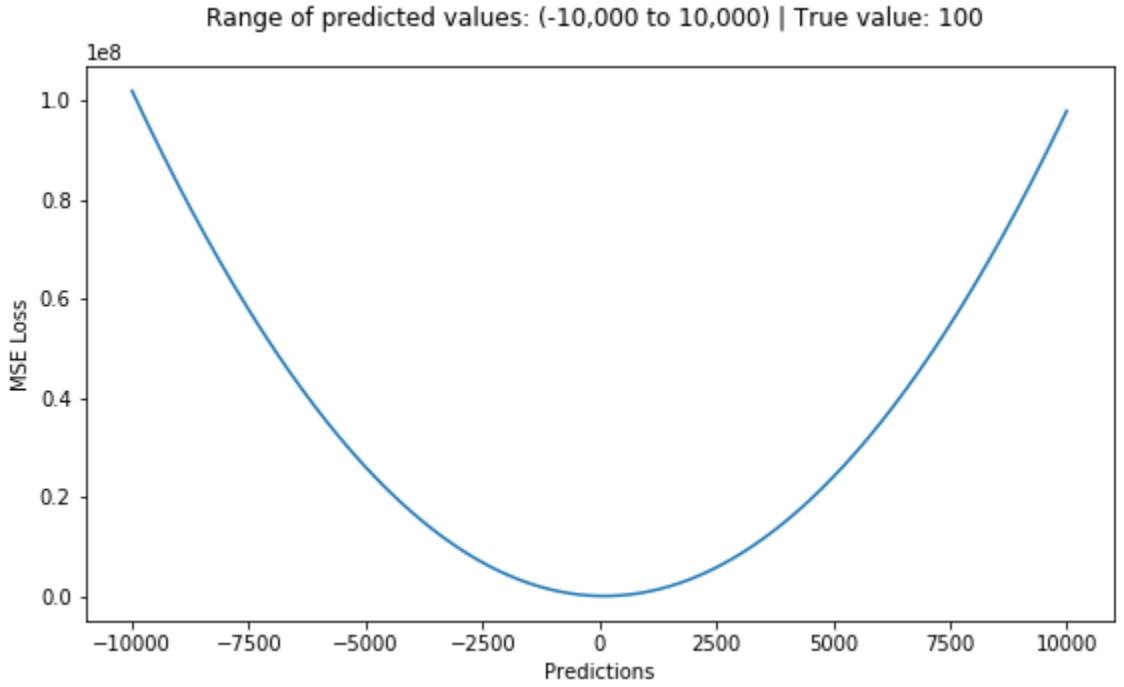


Common Loss functions

Mean Square Error (MSE)

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_i^p)^2}{n}$$

- 실제 값에서 예측 값을 뺀 것의 제곱의 평균

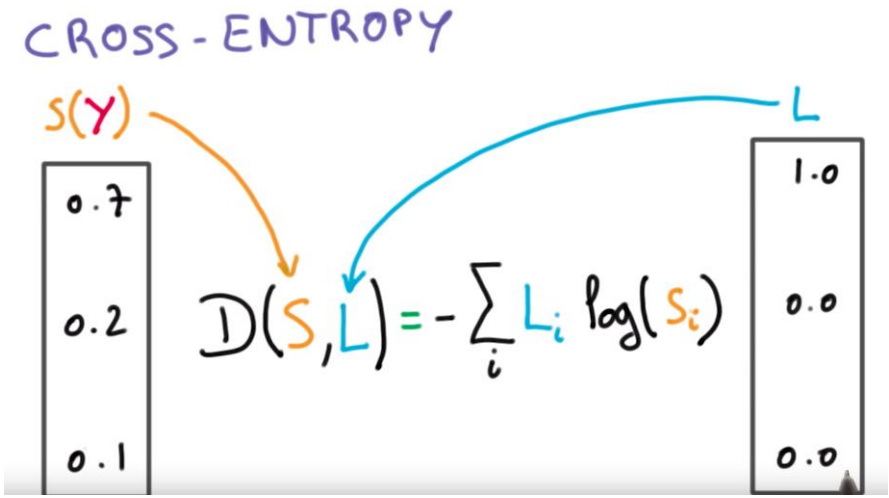


Common Loss functions

Cross entropy error

- 두 확률 분포 사이의 차이가 얼마나 되는지 수치화

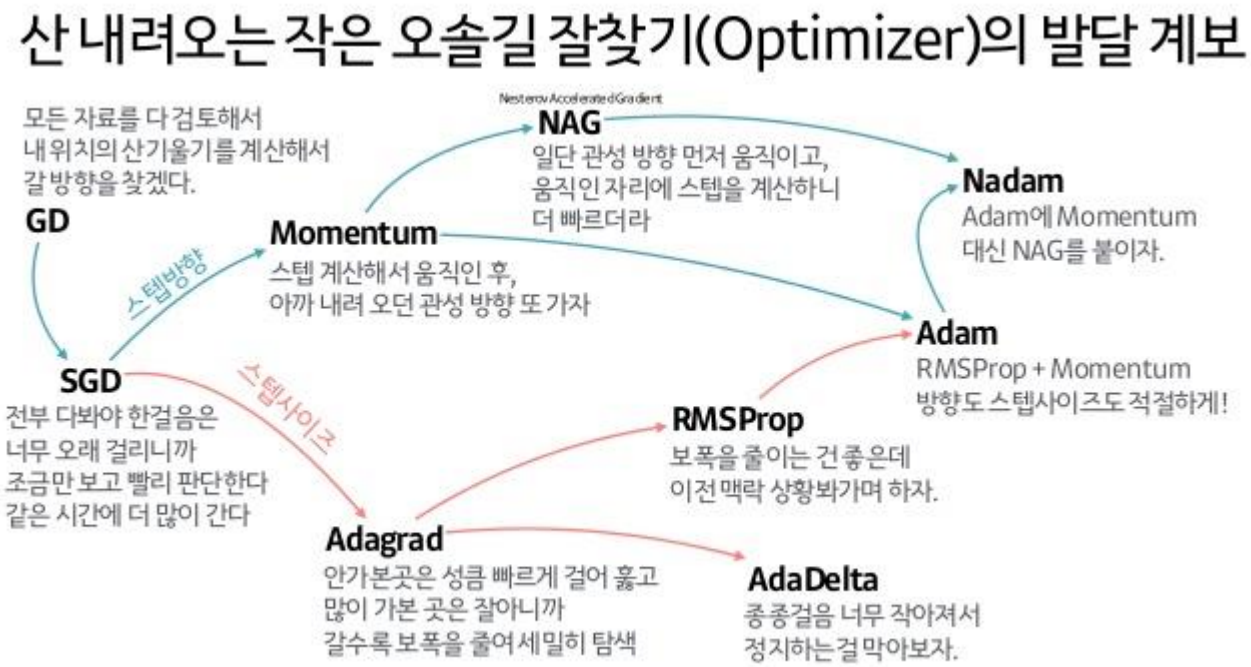
$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}$$



Optimizer

Loss function 을 통해 구한 차이를 사용해 기울기를 구하고

Network의 parameter(W, b) 의 학습에 어떻게 반영할 것인지를 결정하는 방법

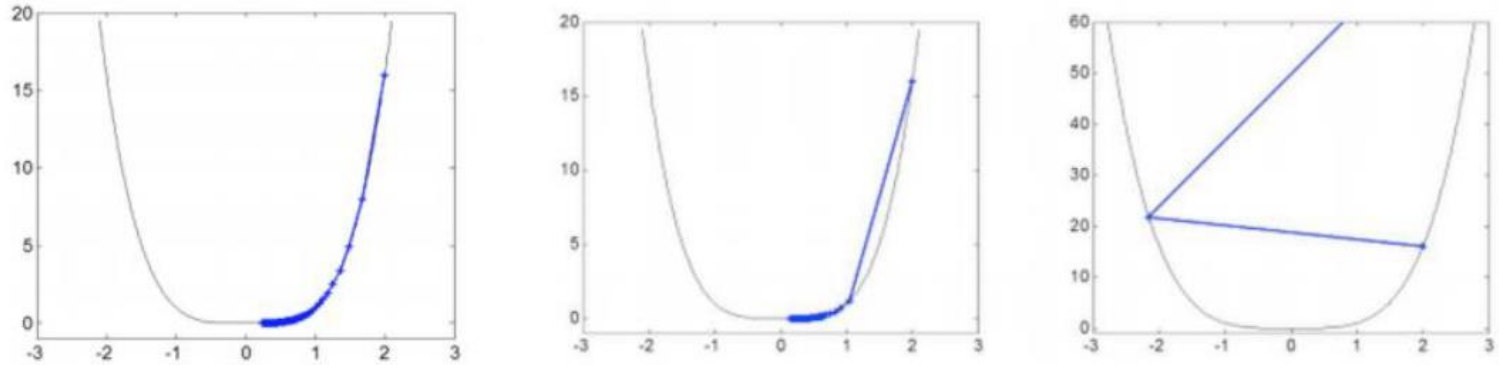


Gradient Descent

1회의 학습 step 시, 현재 모델의 모든 data에 대해서

예측 값에 대한 loss 의 미분을 learning rate 만큼 보정해서 반영

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta)$$



$\eta = 0.01$ $\eta = 0.03$ $\eta = 0.13$

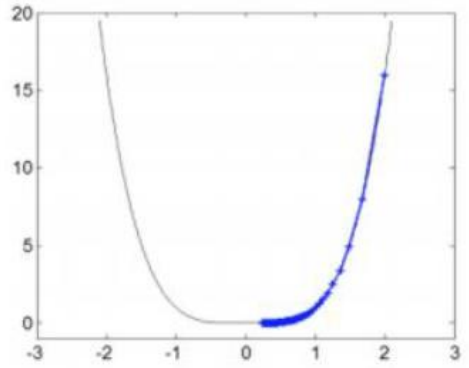
Stochastic Gradient Descent

1회의 학습 step 마다 모든 data 의 계산이 너무 느리기 때문에

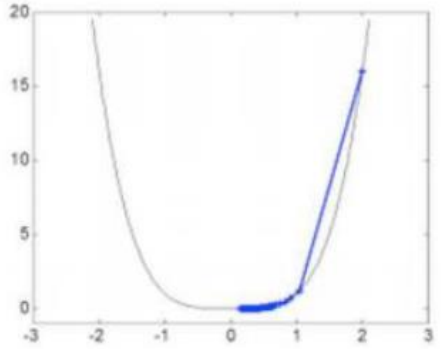
적은 수의 data sample 이 전체 set의 gradient 와 유사할 것이라고 가정하고 계산

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta)$$

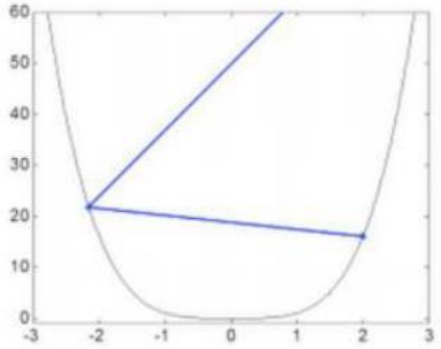
$\boldsymbol{\theta}: \boldsymbol{x}^{(i)}; \boldsymbol{y}^{(i)}$



$\eta = 0.01$



$\eta = 0.03$



$\eta = 0.13$

Momentum

이전 step의 방향(=관성)과 현재 상태의 gradient를 더해 현재 학습할 방향과 크기를 정함

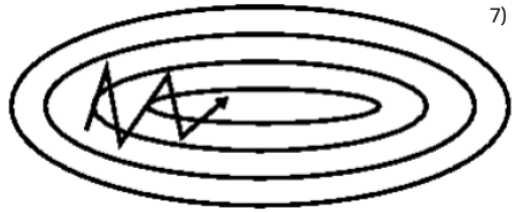
Local minima를 빠져 나올 수 있다

- SGD의 Oscilation 현상 해결

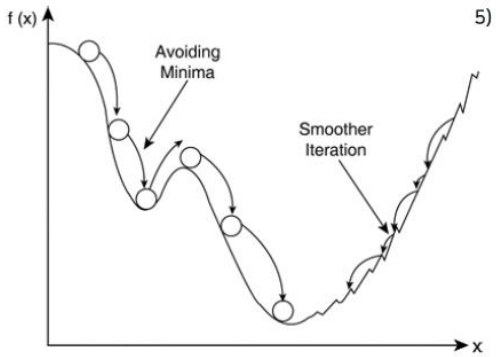
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta = \theta - v_t$$

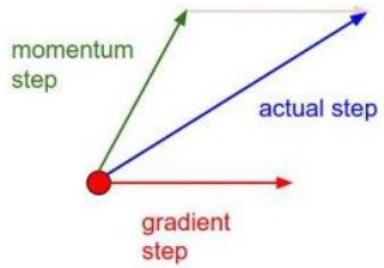
7)



5)



Momentum update 6)

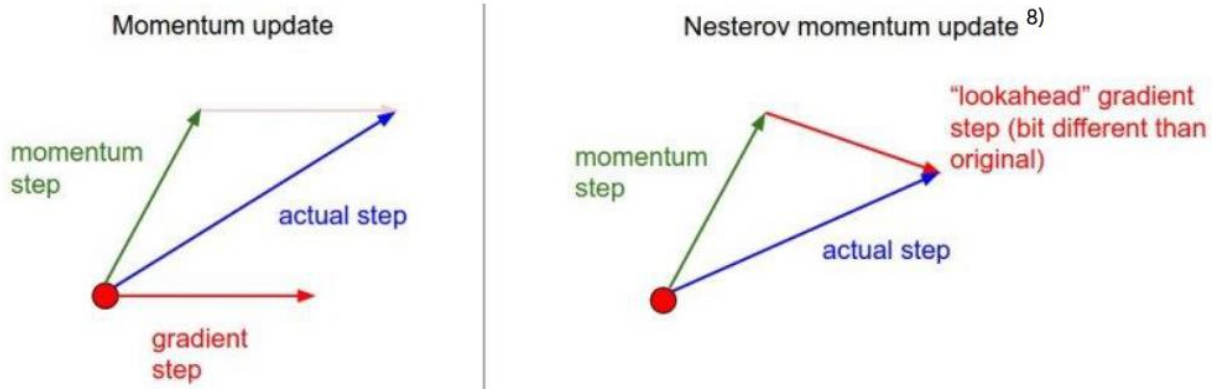


Nesterov Accelerated Gradient(NAG)

Momentum 과 비슷한 개념, 이전까지의 moment step 과 moment step 만큼 이동한 위치에서의 gradient를 더함

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$



Adagrad (Adaptive Gradient)

Parameter 별로 gradient 를 다르게 주는 방식

Network의 parameter 개수가 k 일때, G_t는 time step t 까지 각 변수가 이동한 gradient의 sum of squares를 저장하는 k x k dimension diagonal matrix

많이 변화한 변수들은 G에 저장된 값이 커지기 때문에 step size가 작은 상태로, 적게 변화한 변수들은 상대적으로 step size가 큰 상태로 학습에 반영

학습이 오래 진행되는 경우 G값이 너무 커져서 학습이 제대로 되지 않는다

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(G_t+\epsilon)}} \cdot \nabla_\theta J(\theta_t)$$

RMSProp

학습이 오래 진행되면 step size가 너무 작아지는 adagrad의 단점을 보완하기 위한 방법

각 변수에 대한 gradient의 제곱을 계속 더하는 것이 아니라, 지수평균으로 바꾸어 G값이 무한정 커지지 않도록 방지하면서 변화량의 상대적인 크기 차이를 유지

$$G = \gamma G + (1 - \gamma)(\nabla_\theta J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

AdaDelta (Adaptive Delta)

RMSProp 과 유사한 점은 G에 저장할 때 지수평균을 사용

차이점은 parameter update 시에 step size의 변화값의 제곱을 가지고 지수평균 값을 구하여 learning rate 대체

$$G = \gamma G + (1-\gamma)(\nabla_\theta J(\theta_t))^2$$

$$\Delta_\theta = \frac{\sqrt{s+\epsilon}}{\sqrt{G+\epsilon}} \cdot \nabla_\theta J(\theta_t)$$

$$\theta = \theta - \Delta_\theta$$

$$s = \gamma s + (1-\gamma)\Delta_\theta^2$$

Adam (Adaptive Moment Estimation)

Momentum 방식과 유사하게 지금까지 계산해온 기울기의 지수평균을 저장

RMSProp 과 유사하게 기울기의 제곱값의 지수평균을 저장

학습 초반부에 m과 v가 0에 가깝게 bias 되어 있을 것이라고 판단해 unbiased 작업을 거친 후에 계산을 진행

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta))^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

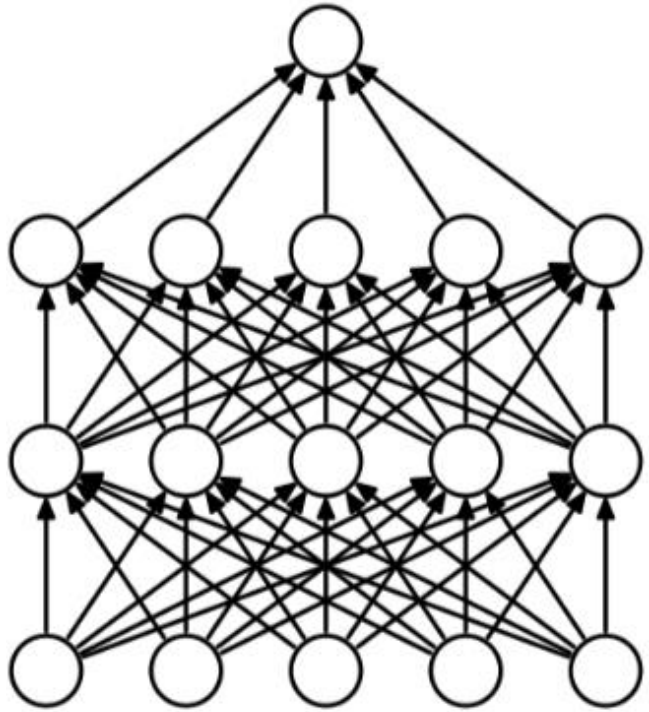
$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

Dropout

왜 필요한가?

- Network의 크기가 커짐에 따라 overfitting에 빠질 가능성이 높아지고 학습 시간도 길어지는 문제가 있으며, 적절한 결과를 도출하려면 training data의 양이 많이 필요

- Overfitting 문제를 완화시킬 수 있는 방법 중 하나

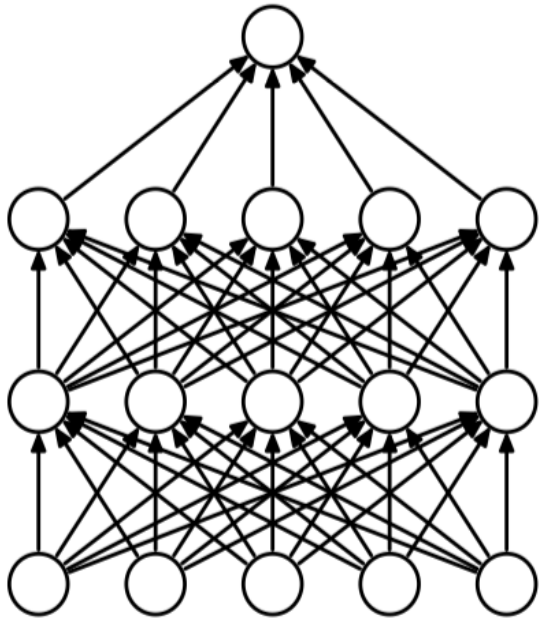


Dropout

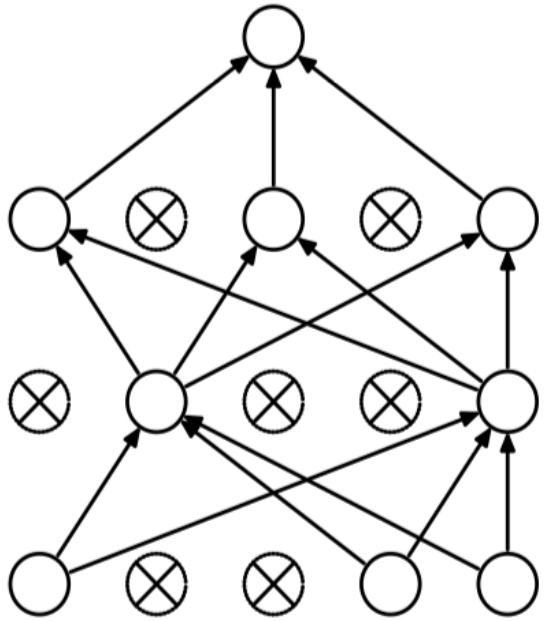
Layer에 포함된 weight 중에서 일부만 참여시킴

Random 하게 일부 뉴런을 0으로 만드는 방법

일정한 mini-batch 구간 동안 dropout 된 망에 대한 학습을 끝내면, 다시 무작위로 다른 뉴런 들을 dropout 하면서 반복적으로 학습



(a) Standard Neural Net



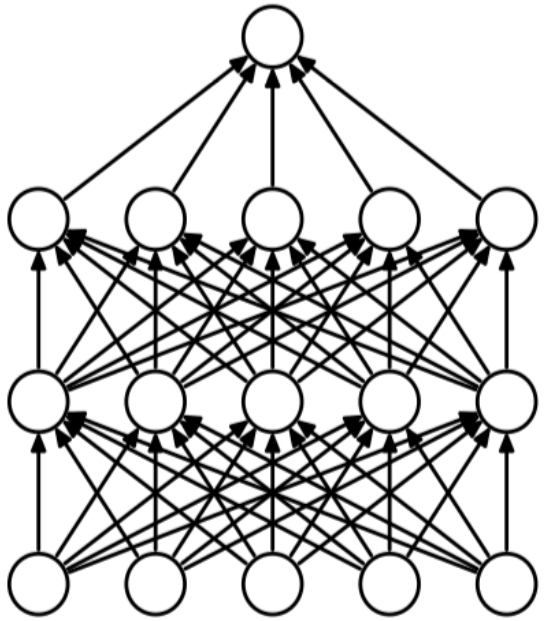
(b) After applying dropout.

Dropout

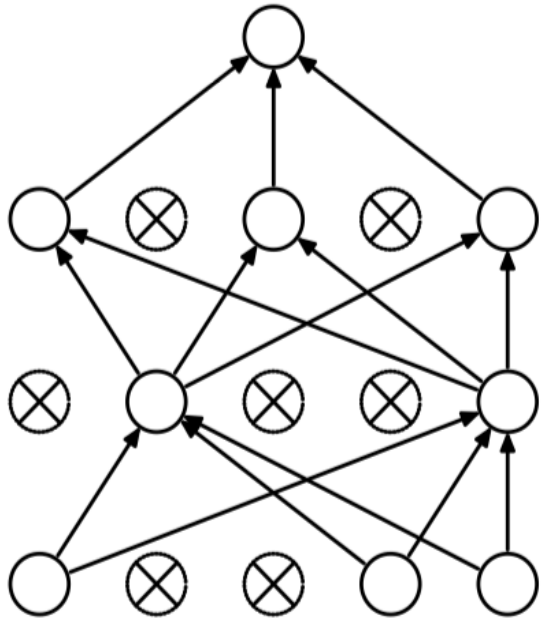
효과

1. Voting 효과
   - mini batch 구간 동안 dropout 된 각자의 망에 fitting이 되면서 평균 효과를 얻을 수 있음

2. Co-adaptation 효과
   - 특정 뉴런의 바이어스나 가중치가 큰 값을 갖게 되면 그것의 영향이 커지면서 다른 뉴런들의 학습 속도가 느려지거나 학습이 제대로 진행 되지 못하는데 dropout을 하면 결과적으로 이러한 뉴런의 가중치나 바이어스에 영향을 받지 않기 때문에 뉴런들이 서로 동조화 되는 것을 피할 수 있다



(a) Standard Neural Net

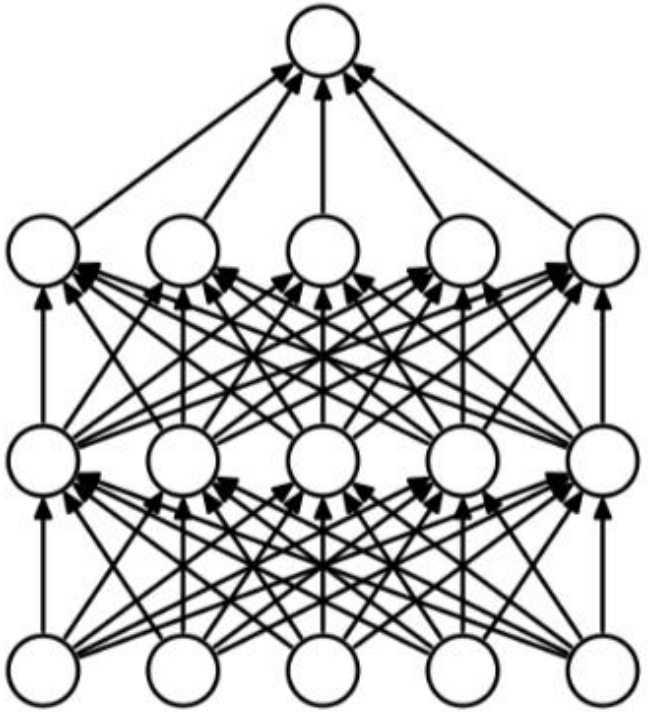


(b) After applying dropout.

Batch-Normalization

왜 필요한가?

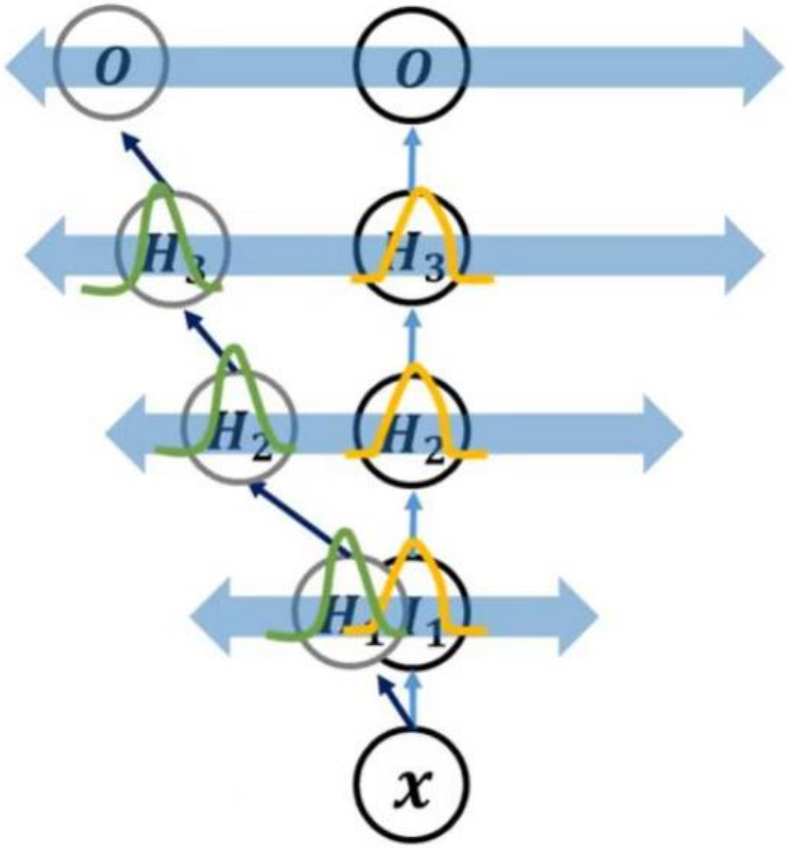
- 학습시켜야 하는 Parameter가 많음
- layer가 많이 쌓일 수록 vanishing/exploding gradient 문제 발생
- Network가 깊어짐에 따라 이전 layer에서의 작은 파라미터 변화가 증폭되어 큰 영향을 끼치게 됨
- 활성 함수나 dropout으로 해결 했지만 본질적인 해결책은 아님



Batch-Normalization

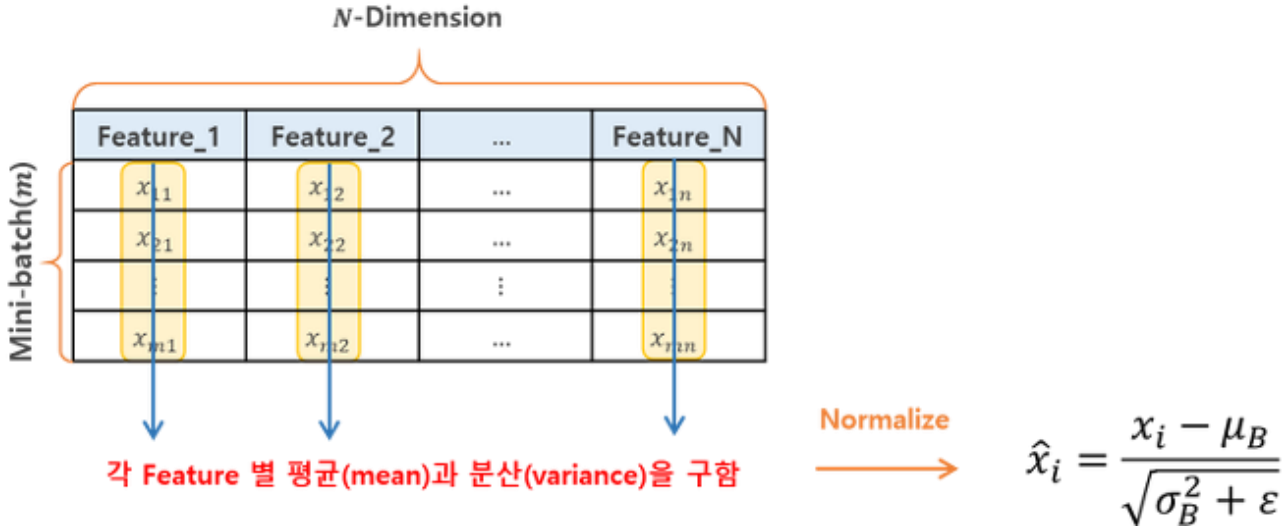
Internal Covariate Shift (내부 공변량 변화)

- 학습하는 도중에 이전 layer의 파라미터 변화로 인해 현재 layer의 입력의 분포가 바뀌는 현상
- layer 상단으로 갈수록 분포 변화가 더 심해짐
- 결과적으로 학습이 힘듦

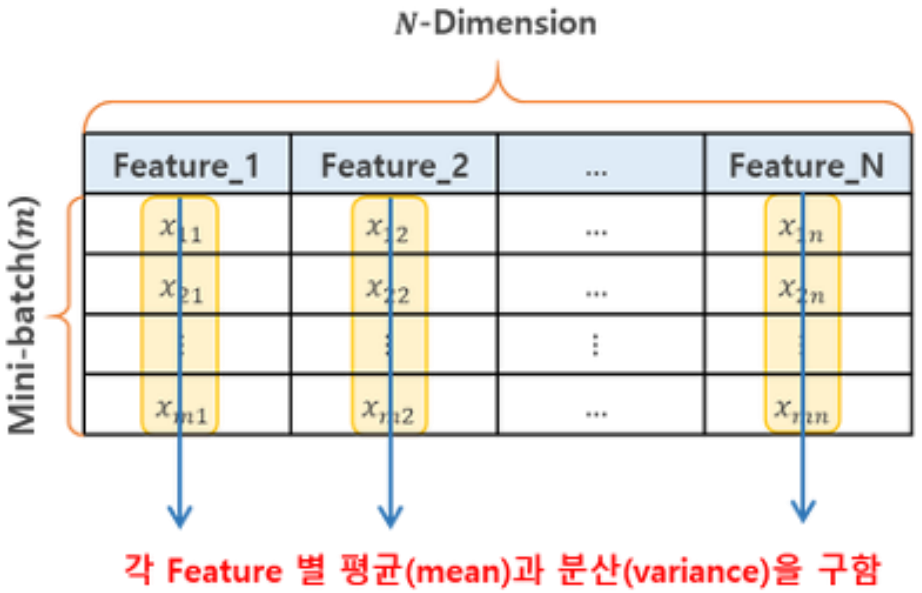


Batch-Normalization

- hyperbolic tangent, sigmoid 같은 활성 함수에 대해 vanishing gradient 문제가 감소
- Parameter 초기화에 덜 민감
- Learning rate 를 크게 잡아도 잘 수렴
- Overfitting 억제, dropout 필요성 감소



Batch-Normalization



Normalize

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{//mini-batch mean}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \qquad \text{//mini-batch variance}$$

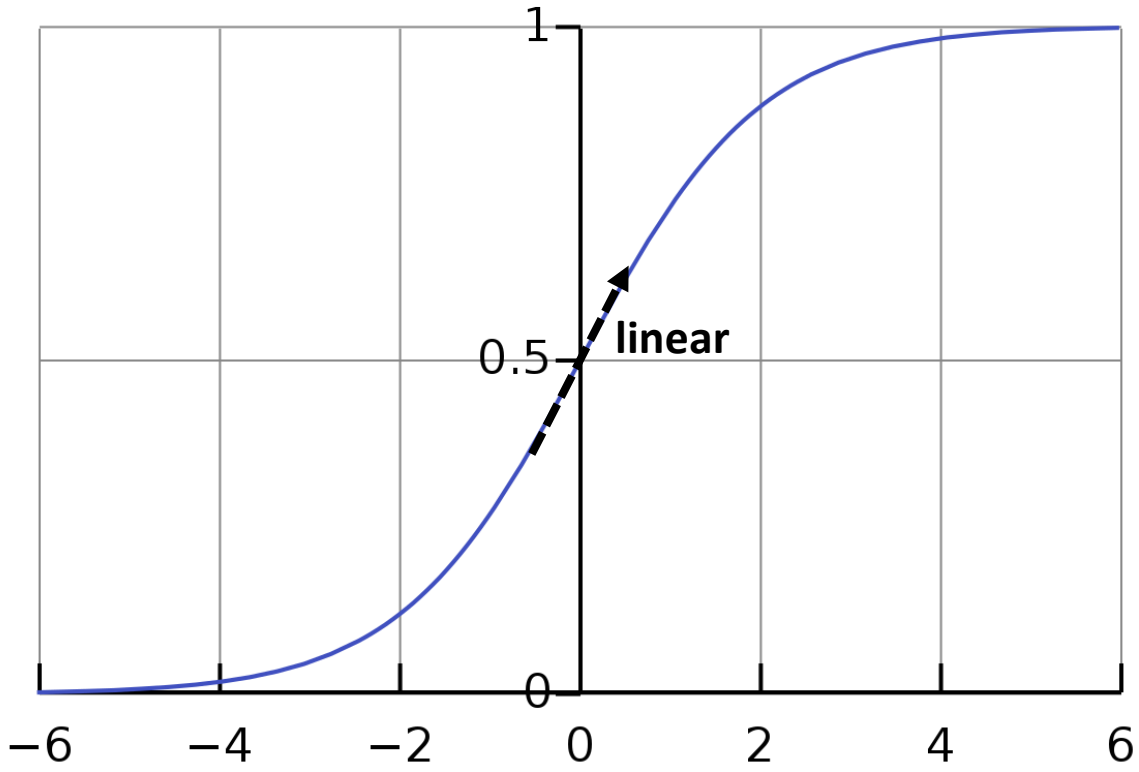
$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \qquad \text{//normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \qquad \text{//scale and shift}$$

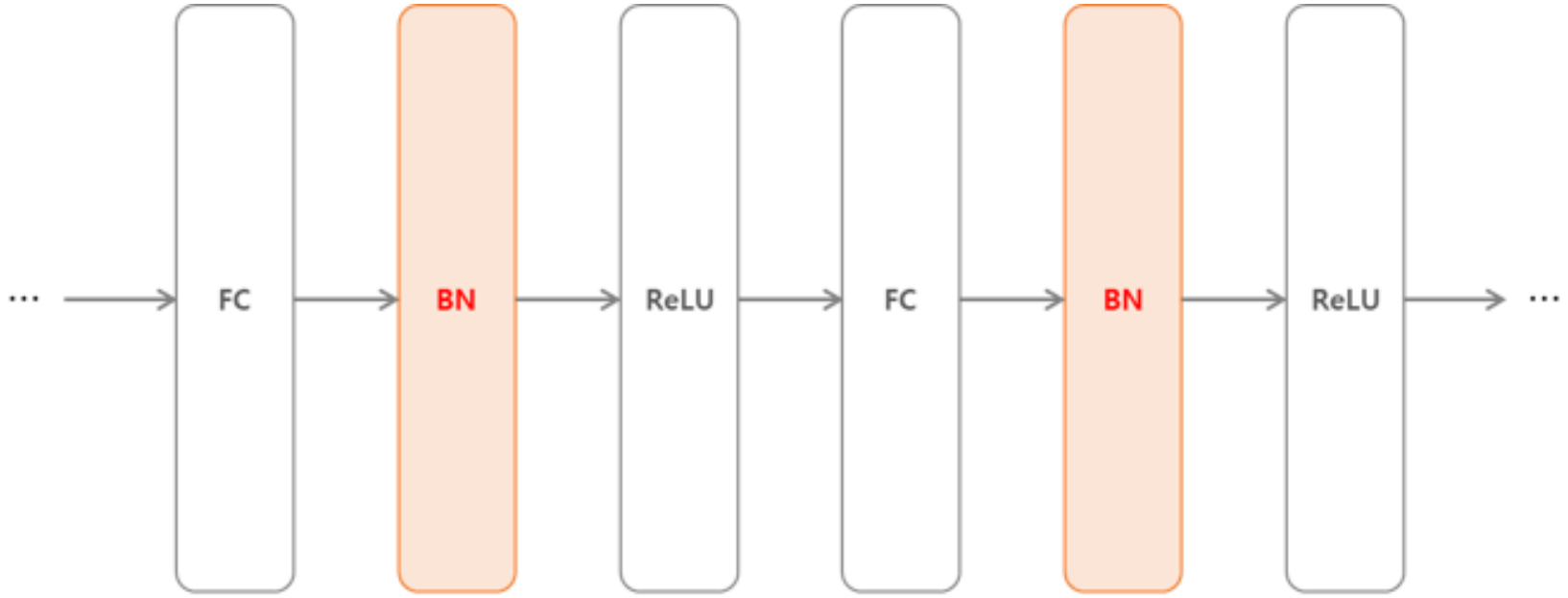
Batch-Normalization

$y_i \leftarrow \gamma \hat{x}_i + \beta$ //scale and shift

입력 데이터에 대해 normalization을 하게 되면 대부분 0에 가까운 값이 된다.
그러면 비선형 함수의 선형 구간에 빠지게 되는데 이러한 문제를 해결하기 위해 scaling과 shifting을 해준다.



Batch-Normalization

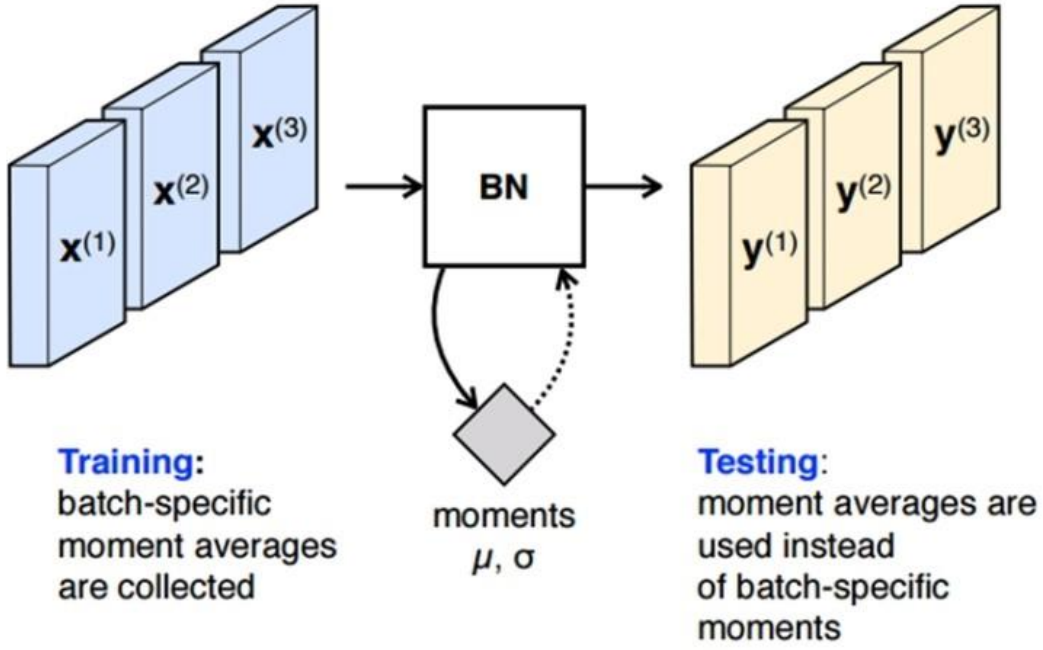


[BN을 사용한 신경망 예]

Batch-Normalization

Test 단계

- 데이터가 하나씩 들어가기 때문에 mean, variance를 계산할 mini-batch가 없다. 그래서 전체 Training Set의 mean, variance를 사용
- 계산량이 많은 경우 training 동안 구한 mean, variance들의 평균으로 사용




Q
A