

캡스톤 디자인

2020.09.18

고지형 김양기 박진수 안나민

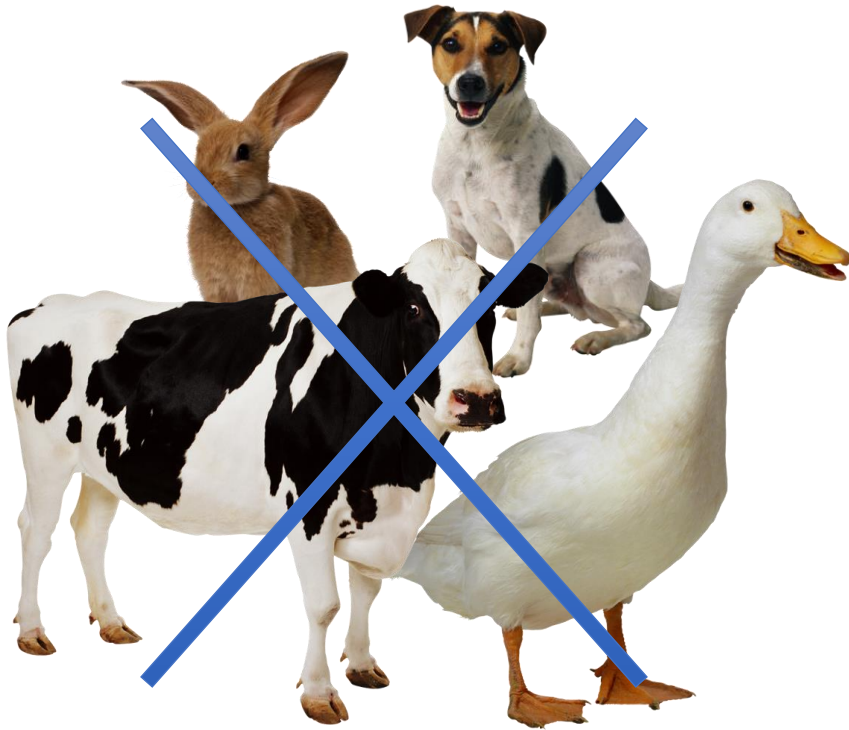
Intro to Deep Learning

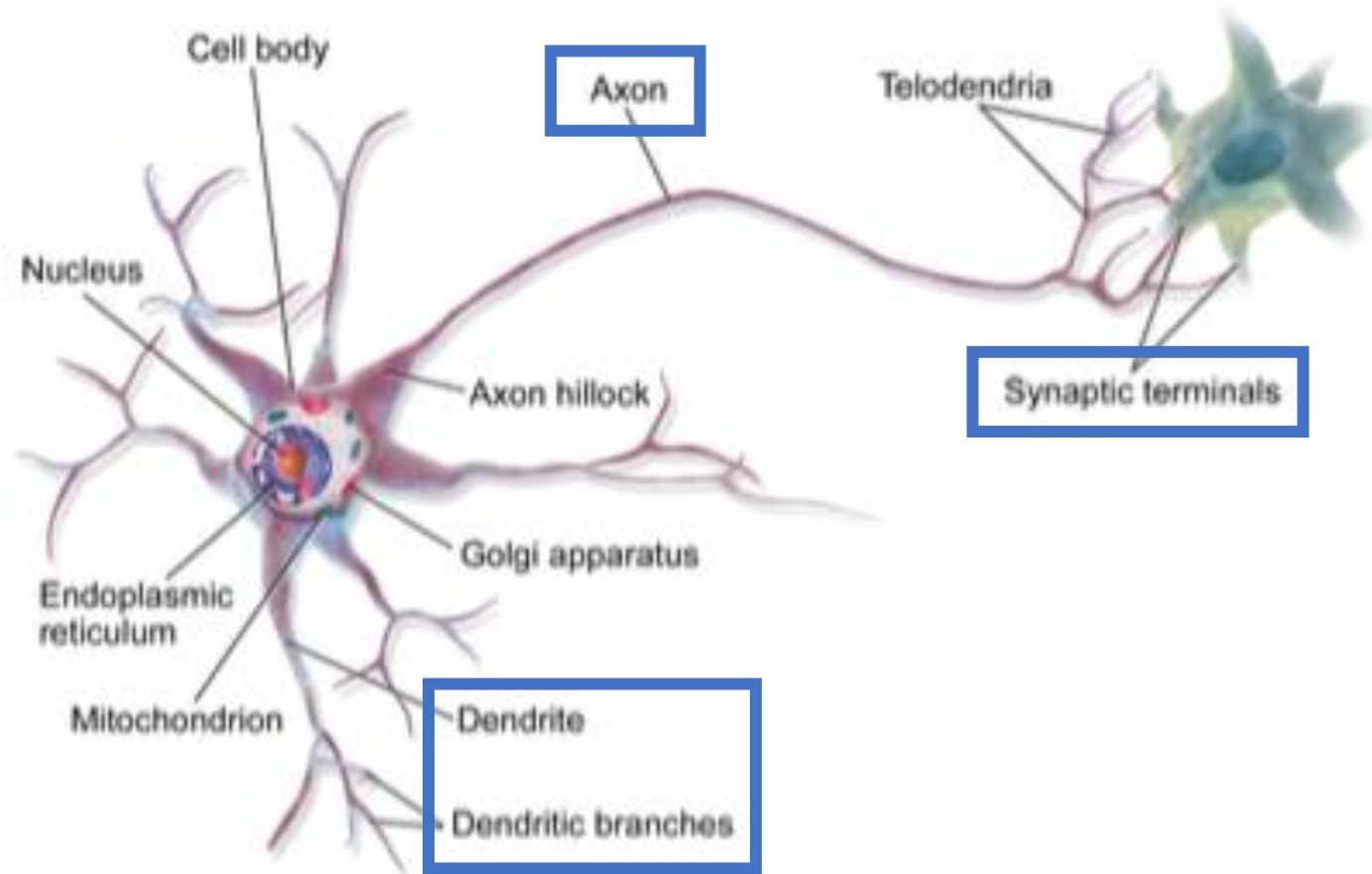


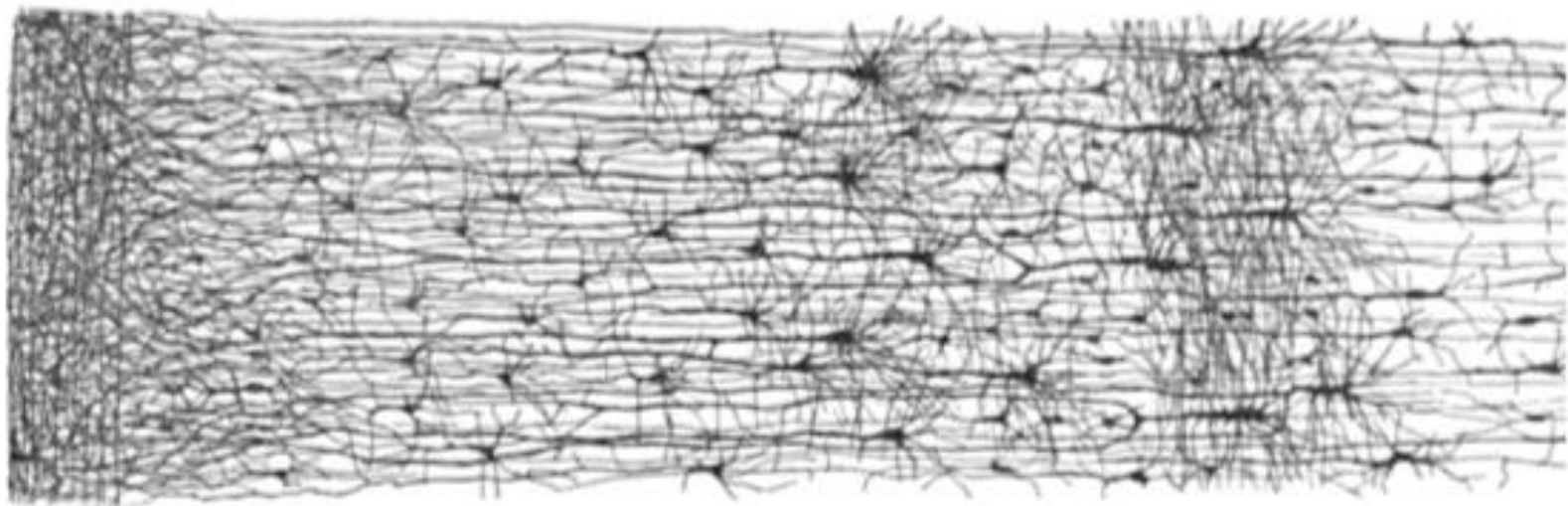
딥러닝의 정의와 예시

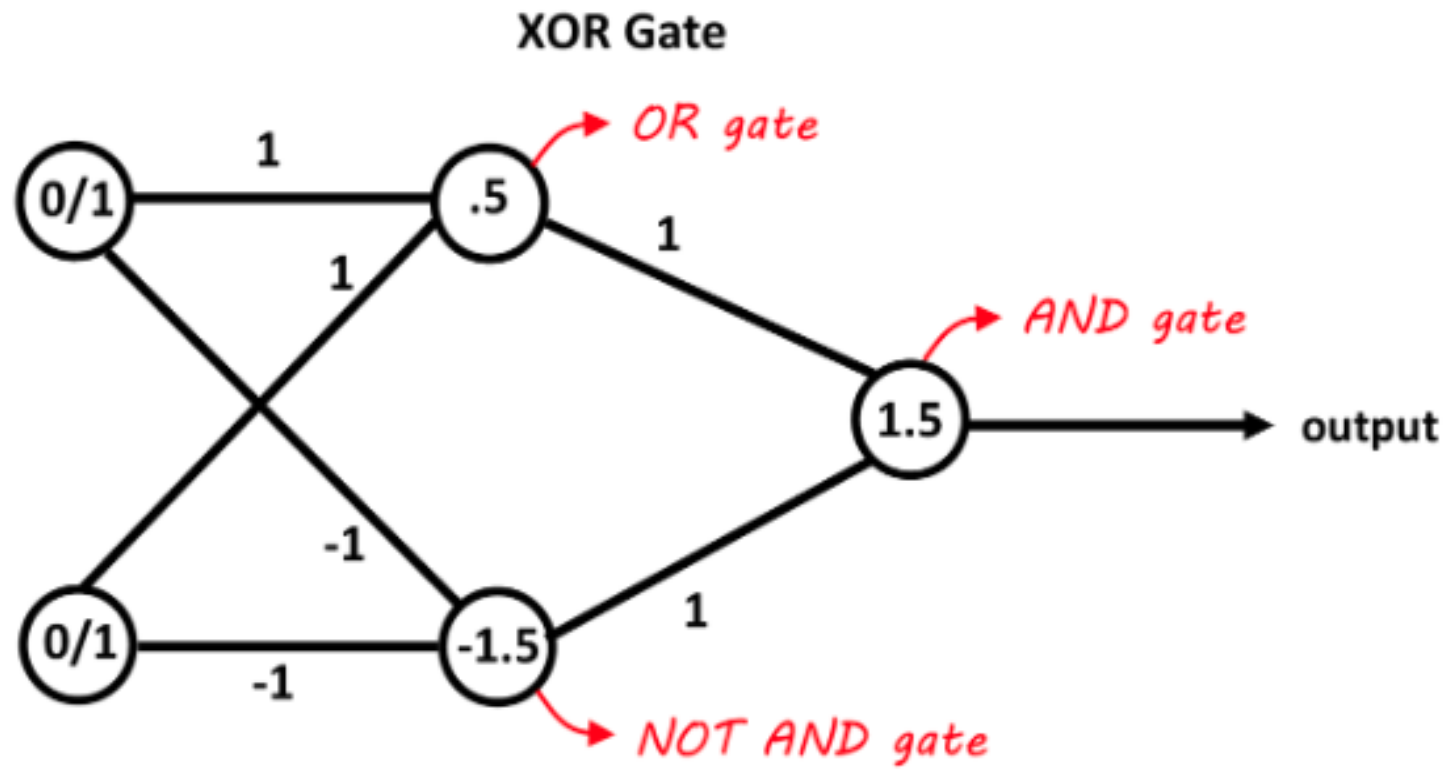
딥러닝의 구조 및 단계 (역전파 함수 전까지)

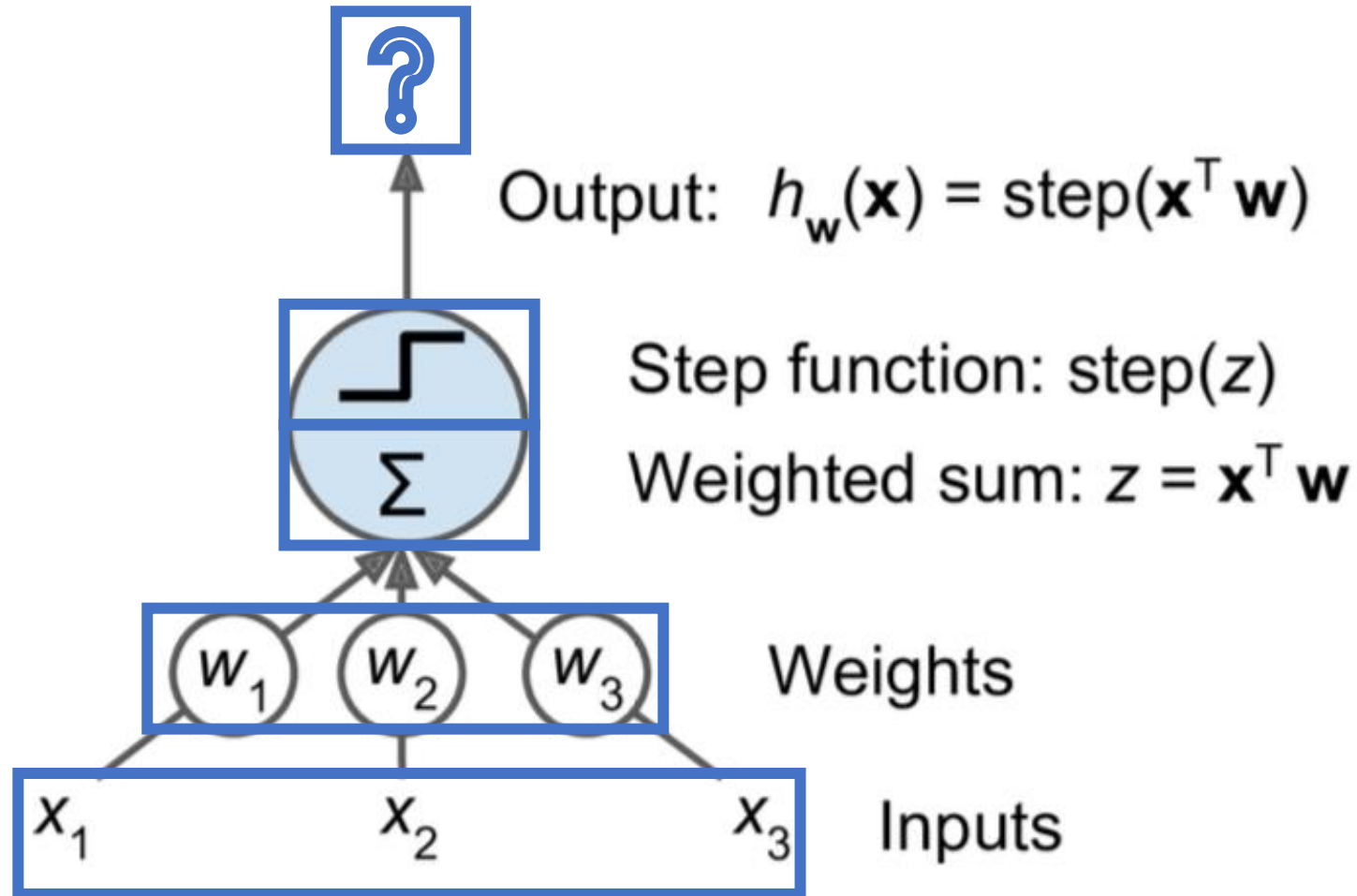
뉴럴 네트워크를 사용하여 데이터에서 패턴을 추출하는 것





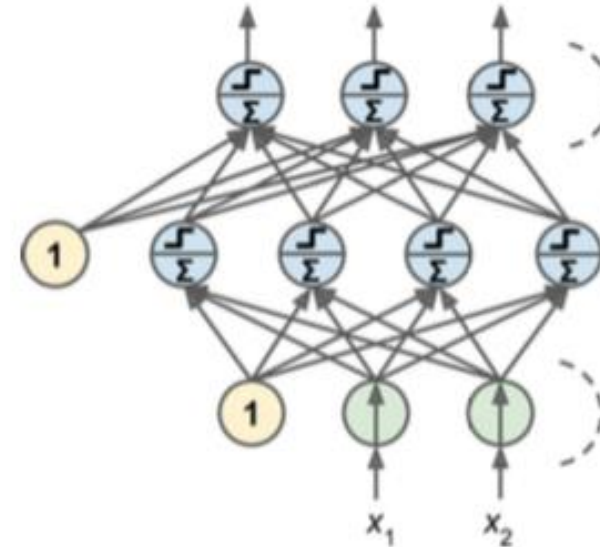
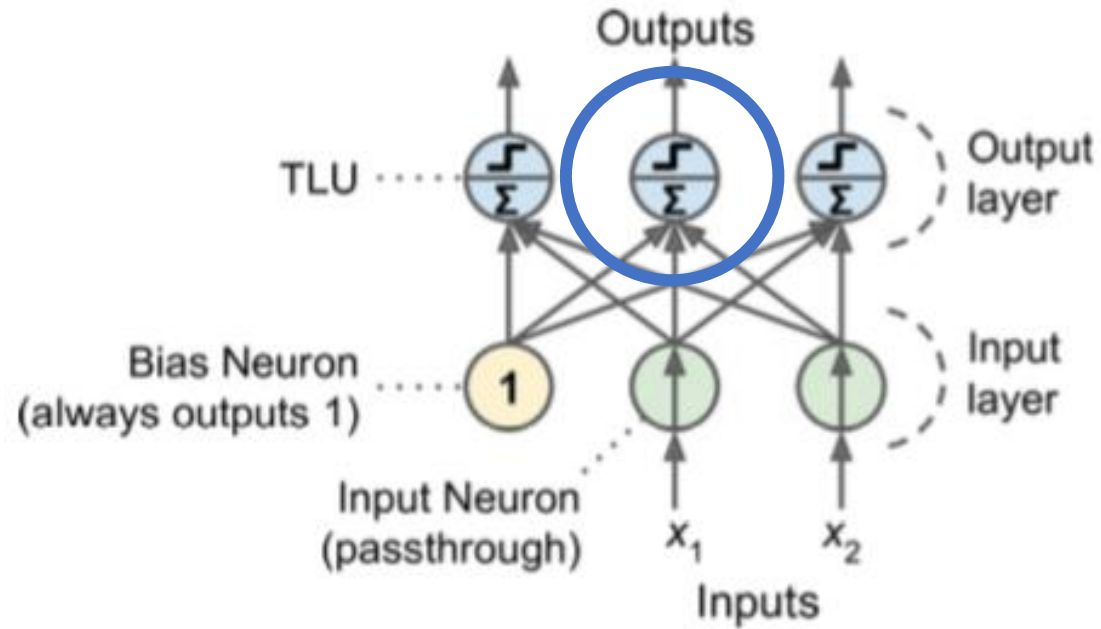




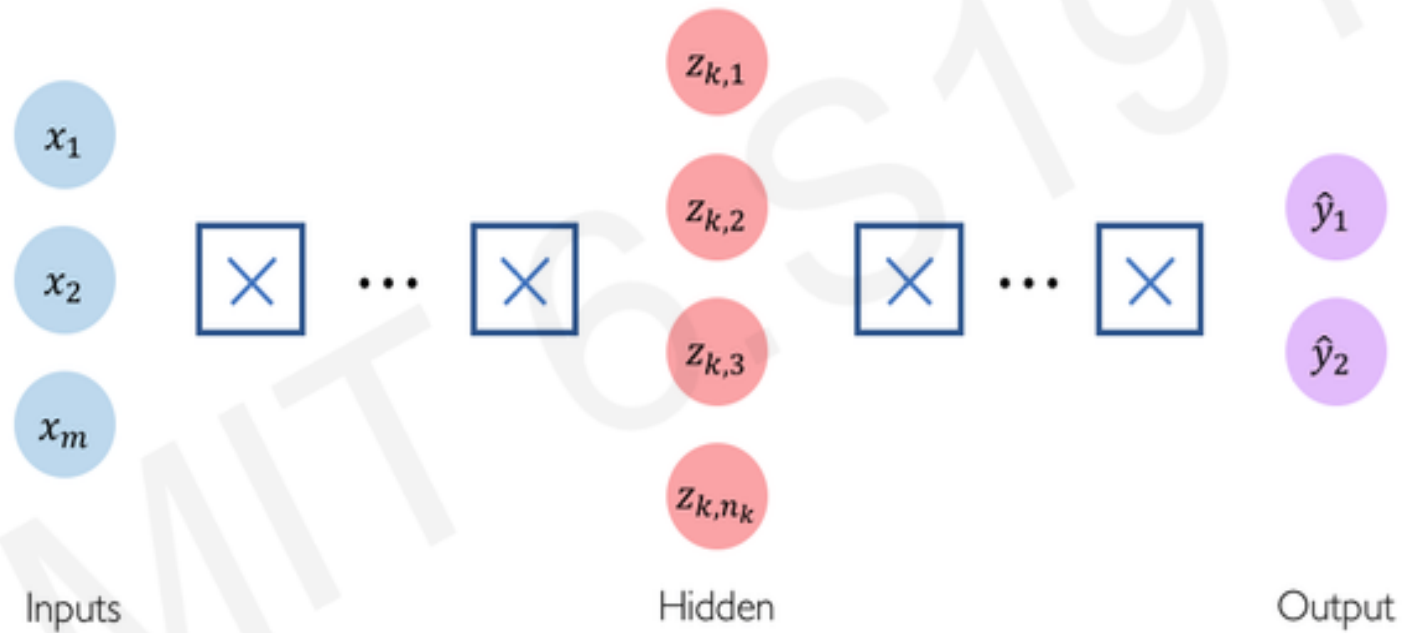


Inputs
Weights
Sum
Non-Linearity
Output

퍼셉트론

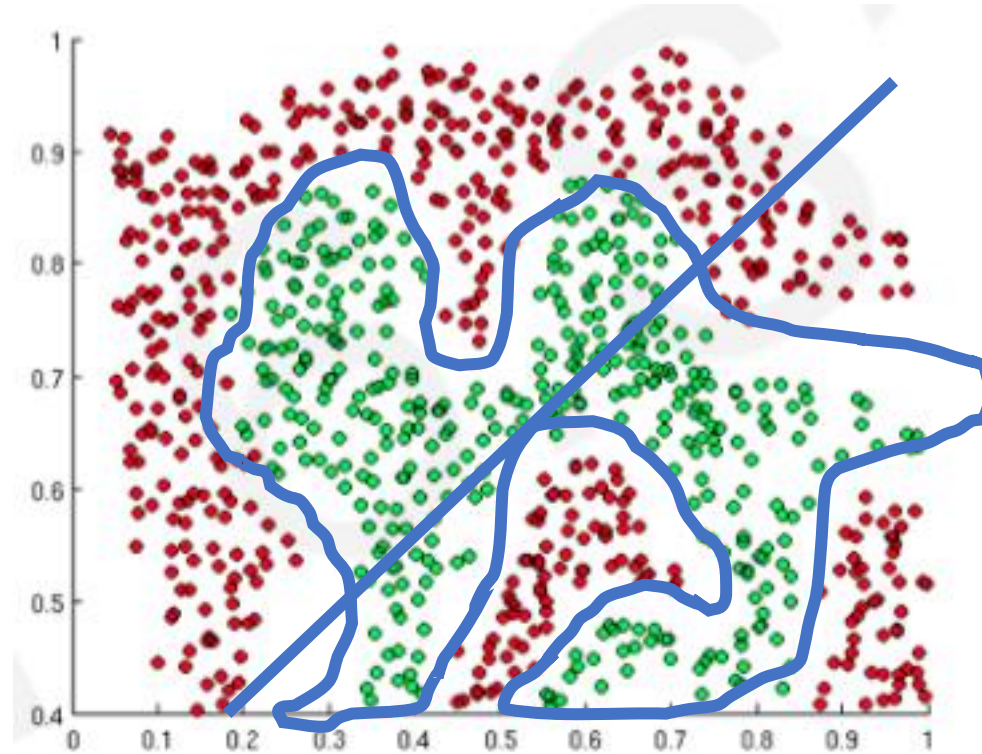


Inputs
Weights
Sum
Non-Linearity
Output

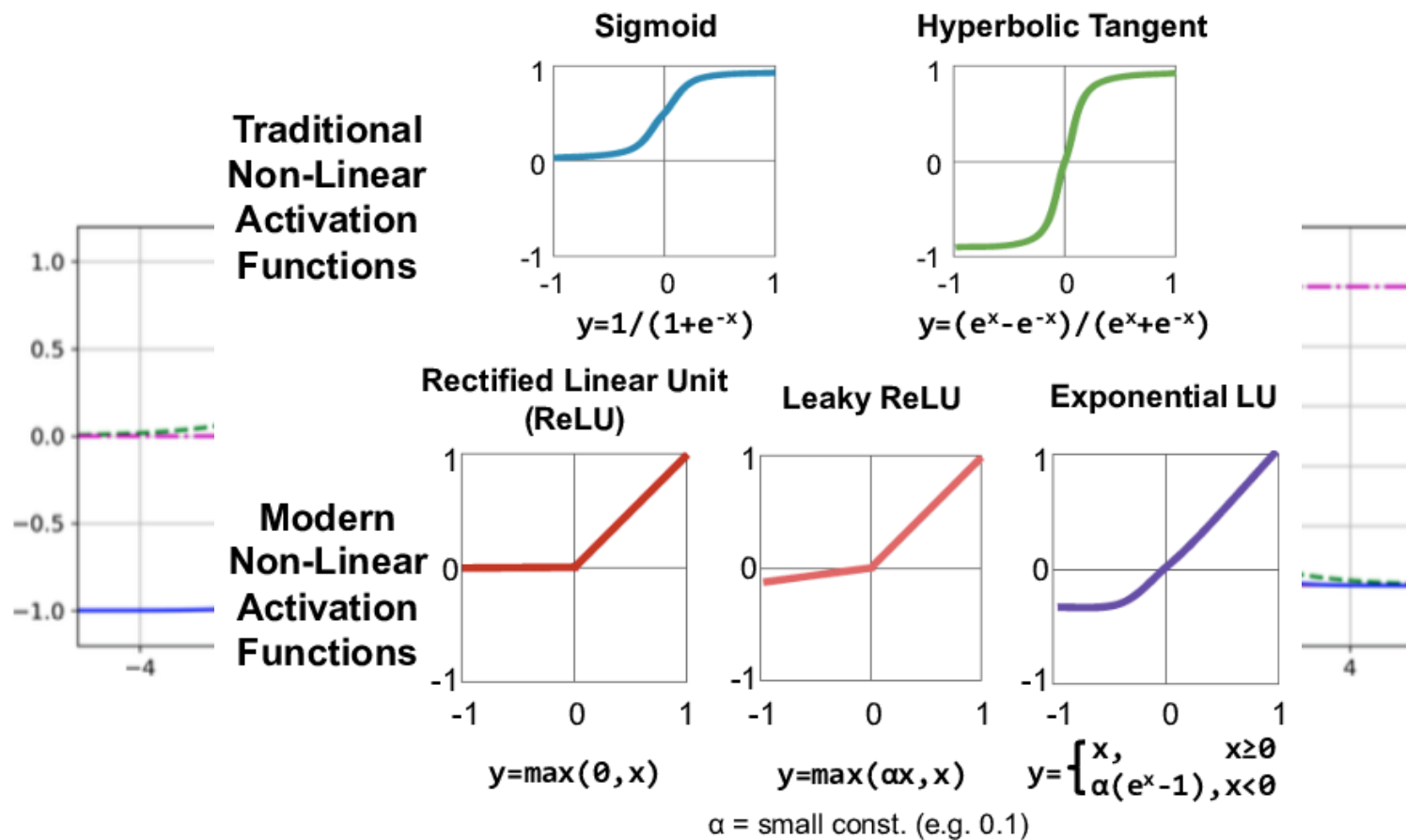


Inputs
Weights
Sum
Non-Linearity
Output

$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$



Inputs
Weights
Sum
Non-Linearity
Output



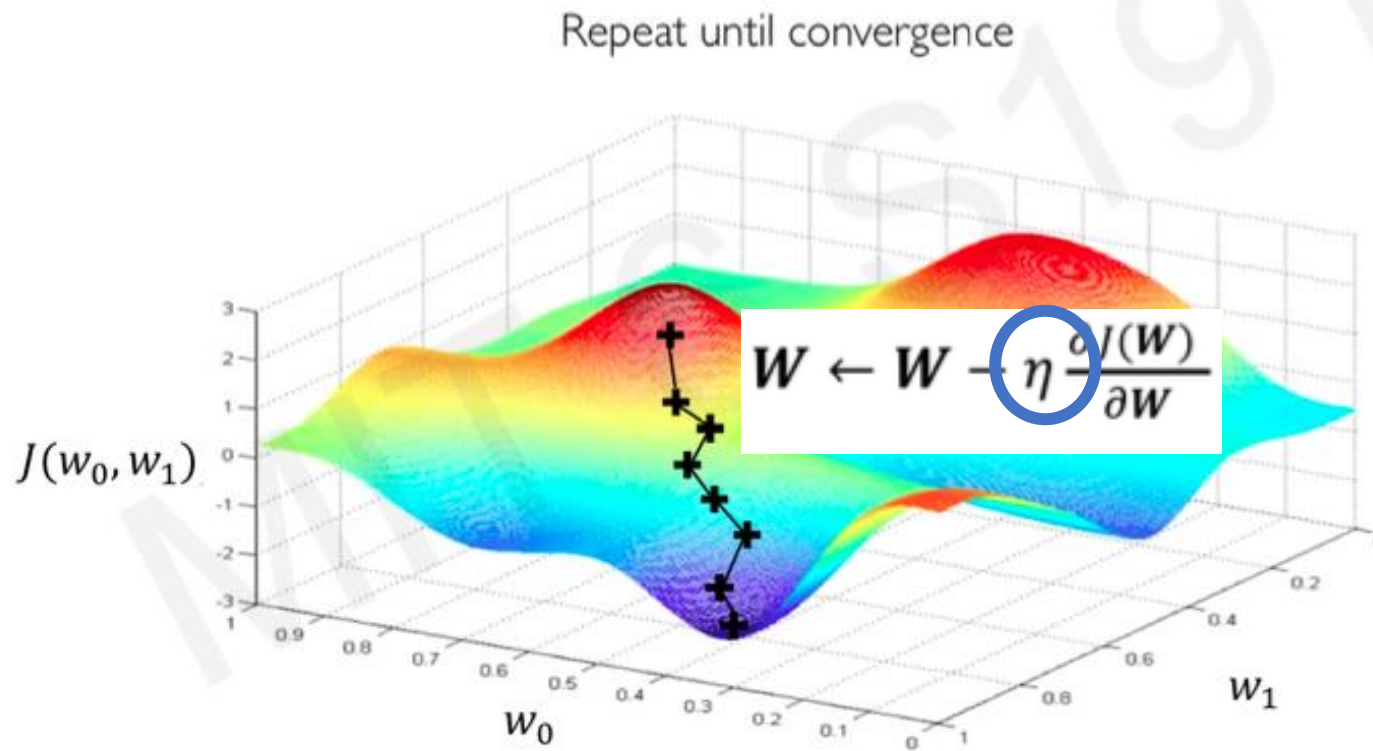
Inputs
 Weights
 Sum
 Non-Linearity
 Output

$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \left(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f(x^{(i)}; \mathbf{W})}_{\text{Predicted}} \right)^2$$

$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \underbrace{y^{(i)}}_{\text{Actual}} \log \left(\underbrace{f(x^{(i)}; \mathbf{W})}_{\text{Predicted}} \right) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \left(1 - \underbrace{f(x^{(i)}; \mathbf{W})}_{\text{Predicted}} \right)$$

Inputs
Weights
Sum
Non-Linearity
Output

그래디언트 디센트



Inputs
Weights
Sum
Non-Linearity
Output


Table 10-1. Typical Regression MLP Architecture

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., $28 \times 28 = 784$ for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber

$$P(y=j \mid \theta^{(i)}) = \frac{e^{\theta^{(i)}}}{\sum_{j=0}^k e^{\theta_k^{(i)}}}$$

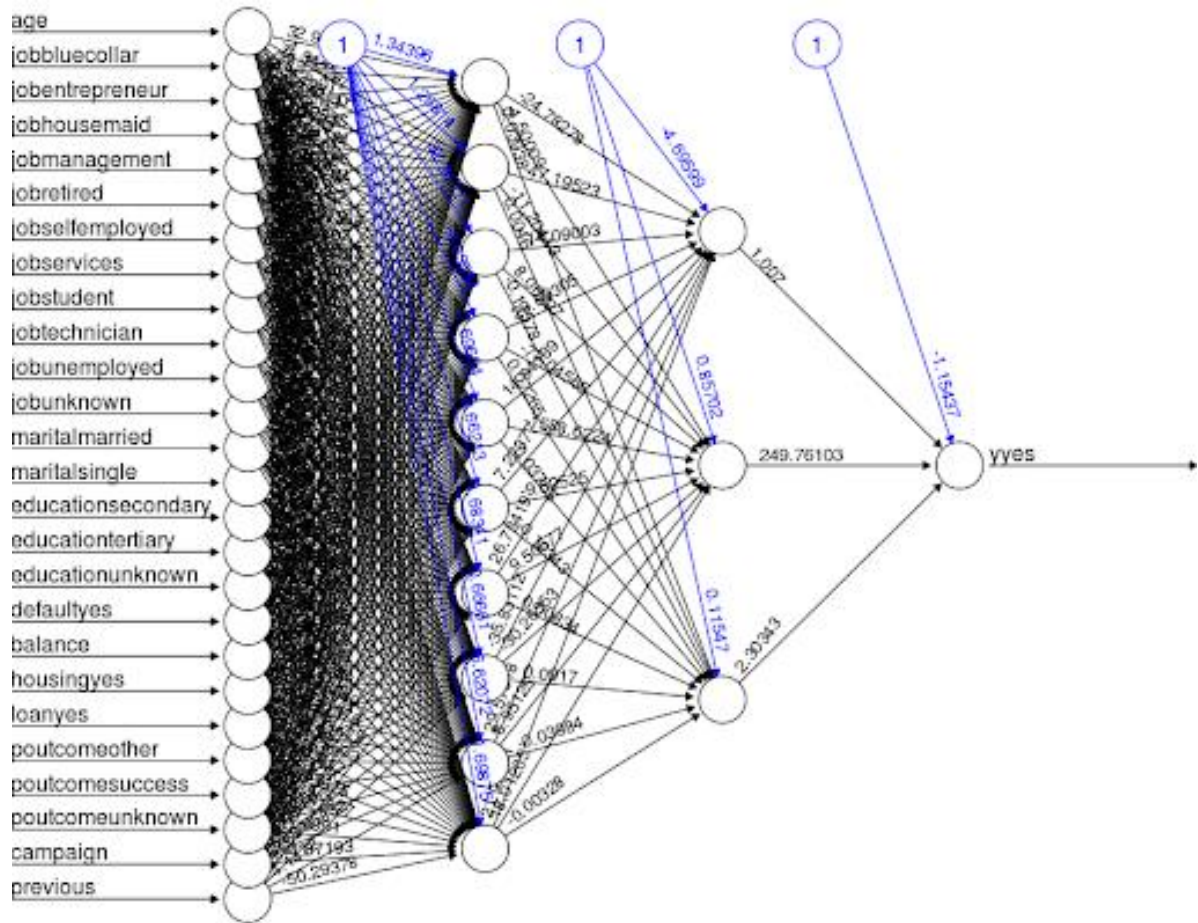
where $\theta = w_0x_0 + w_1x_1 + \dots + w_kx_k = \sum_{i=0}^k w_ix_i = w^T x$

Softmax function



A mathematical representation of the Softmax Regression function

딥러닝의 단계



MIT Lecture 1 Intro to Deep Learning

OREILLY Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow

YOUTUBE Speaking of AI

ABHRANIL BLOG Training Neural Networks with Genetic Algorithms

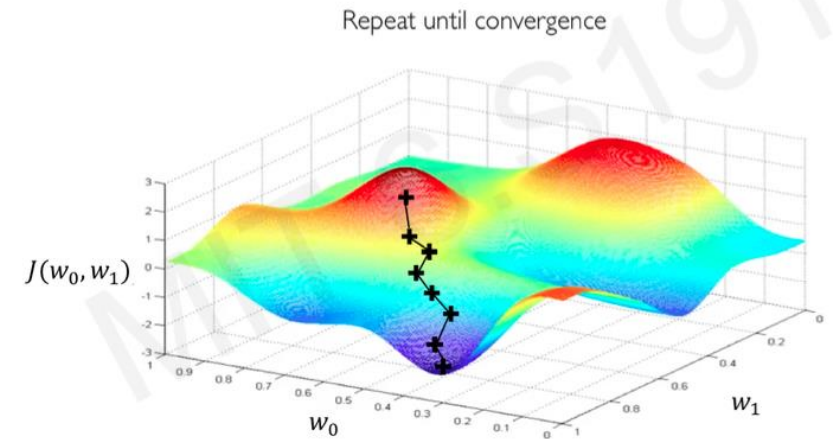
GOOGLE Images

역전파(Back propagation)

역전파

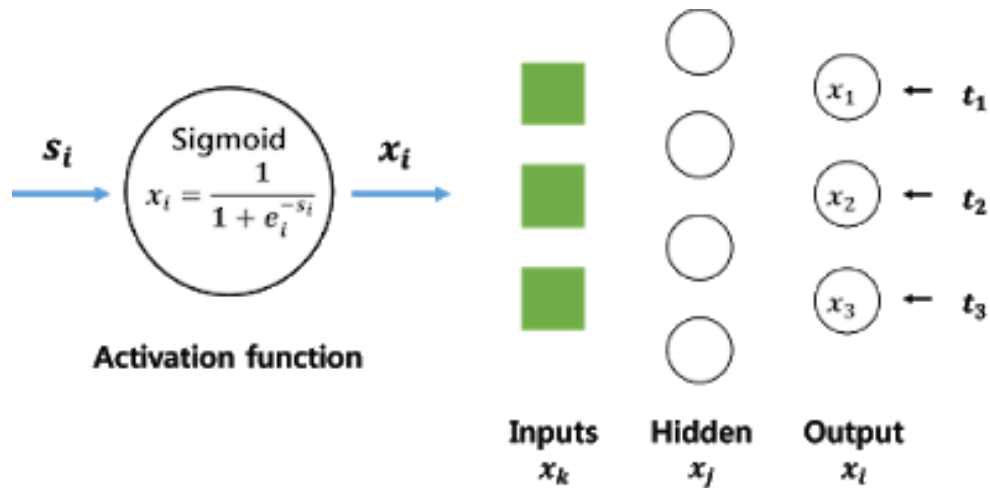
- 주어진 가중치를 통해 예측치를 구하는 과정: 순전파(Forward propagation)
- 순전파로 구한 예측치와 관측치의 오차가 감소하도록 가중치를 업데이트 하는 과정: 역전파(Back propagation)
- ‘오차 최소 = 손실함수 값이 최소가 되는 지점’
- 경사하강법을 통해 가중치를 업데이트하여 최소 지점을 찾음

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W} \quad J: \text{손실함수}, \eta: \text{학습률}$$



역전파(Backpropagation)

역전파 예시



가정

- 이진 분류
- 1개의 은닉층
- 활성화 함수: Sigmoid
- 손실함수는 Binary Cross Entropy

$$E = - \sum_{i=1}^{nout} (t_i \log(x_i) + (1 - t_i) \log(1 - x_i))$$

$$\text{where, } x_i = \frac{1}{1+e^{-s_i}}, s_i = \sum_{j=1} x_j w_{ji}.$$

(1) 출력층과 은닉층 사이의 가중치 업데이트

체인룰에 의해 다음이 유도됨

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}}$$

$$1. \frac{\partial E}{\partial x_i} = \frac{-t_i}{x_i} + \frac{1-t_i}{1-x_i} = \frac{x_i - t_i}{x_i(1-x_i)}$$

$$2. \frac{\partial x_i}{\partial s_i} = x_i(1 - x_i)$$

$$3. \frac{\partial s_i}{\partial w_{ji}} = x_j$$

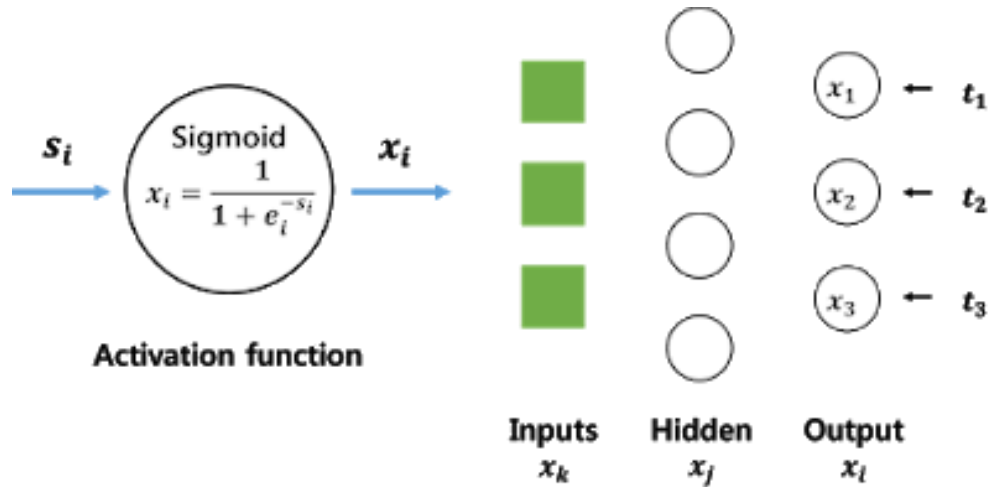
$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

$$\therefore \frac{\partial E}{\partial w_{ji}} = (x_i - t_i)x_j$$

역전파(Backpropagation)

역전파 예시



가정

- 이진 분류
- 1개의 은닉층
- 활성화 함수: Sigmoid
- 손실함수는 Binary Cross Entropy

$$E = - \sum_{i=1}^{nout} (t_i \log(x_i) + (1 - t_i) \log(1 - x_i))$$

$$\text{where, } x_i = \frac{1}{1+e^{-s_i}}, s_i = \sum_{j=1} x_j w_{ji}.$$

(2) 은닉층과 입력층 사이의 가중치 업데이트

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial w_{kj}}$$

$$1. \frac{\partial E}{\partial s_j} = \sum_{i=1}^{nout} \frac{\partial E}{\partial s_i} \frac{\partial s_i}{\partial x_j} \frac{\partial x_j}{\partial s_j} = \sum_{i=1}^{nout} (x_i - t_i)(w_{ji})(x_j(1 - x_j))$$

$$2. \frac{\partial s_j}{\partial w_{kj}} = x_k \left(\because s_j = \sum_{k=1} x_k w_{kj} \right)$$

$$\frac{\partial E}{\partial x_i} = \frac{-t_i}{x_i} + \frac{1-t_i}{1-x_i} = \frac{x_i - t_i}{x_i(1-x_i)}$$
$$\frac{\partial x_i}{\partial s_i} = x_i(1 - x_i)$$

$$\therefore \frac{\partial E}{\partial w_{kj}} = \sum_{i=1}^{nout} (x_i - t_i)(w_{ji})(x_j(1 - x_j))(x_k)$$

학습률(Learning Rate)

학습률

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

- 기울기(Gradient)에 사용자가 지정한 만큼 상수배한 것을 기존 가중치에 빼 최적화 진행
- 이 상수값이 학습률이며, 0과 1 사이의 값임
- 너무 크면 최소 지점을 찾지 못하는 발산 위험
- 너무 작으면 학습 속도가 너무 느리거나, 국소 최소 지점에 수렴하는 문제
- 경험적으로 조절하거나, Adaptive Learning Rate 방법 사용
- Adaptive Learning Rate 최적화 함수
AdaGrad, RMSProp, Adam

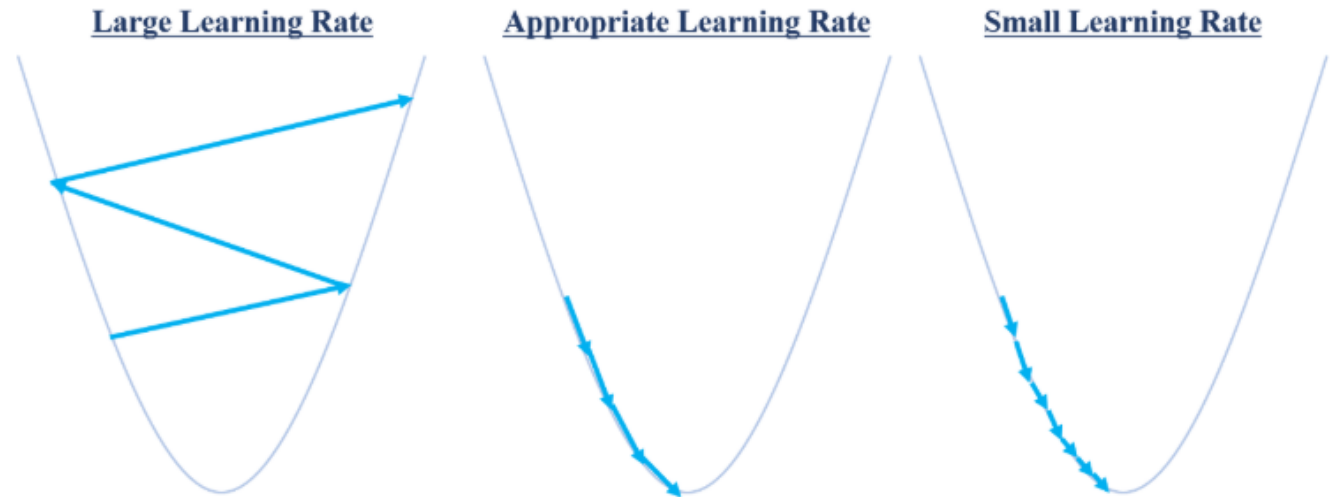


Figure 1: Learning Rate와 Gradient Descent

파라미터 최적화(Parameter Optimization)

경사하강법 기반 파라미터 최적화 방식

$$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$

(1) Batch Gradient Descent(BGD)

- 모든 데이터셋을 하나의 배치(batch)로 보고 전체 미분값을 평균하여 1 에폭(epoch)동안 1회의 업데이트
- 장점: 최적 지점을 확실히 찾을 수 있다
- 단점: 속도가 느리다

*에폭: 한 번의 학습이 진행되는 것

*배치: 한 번의 업데이트에 활용되는 데이터 덩어리

(2) Stochastic Gradient Descent(SGD)

- 임의로 하나의 데이터만을 뽑아 학습시키는 방법
- 장점: 속도가 빠르다
- 단점: 최적 지점을 찾는 과정에서 노이즈 발생 위험

(3) Mini-Batch Gradient Descent(MGD)

- 전체 데이터 중 N개의 데이터를 뽑아 미분값을 평균하여 업데이트
- BGD에 비해 속도가 빠르고, SGD에 비해 안정적
- 가장 많이 사용하는 방식

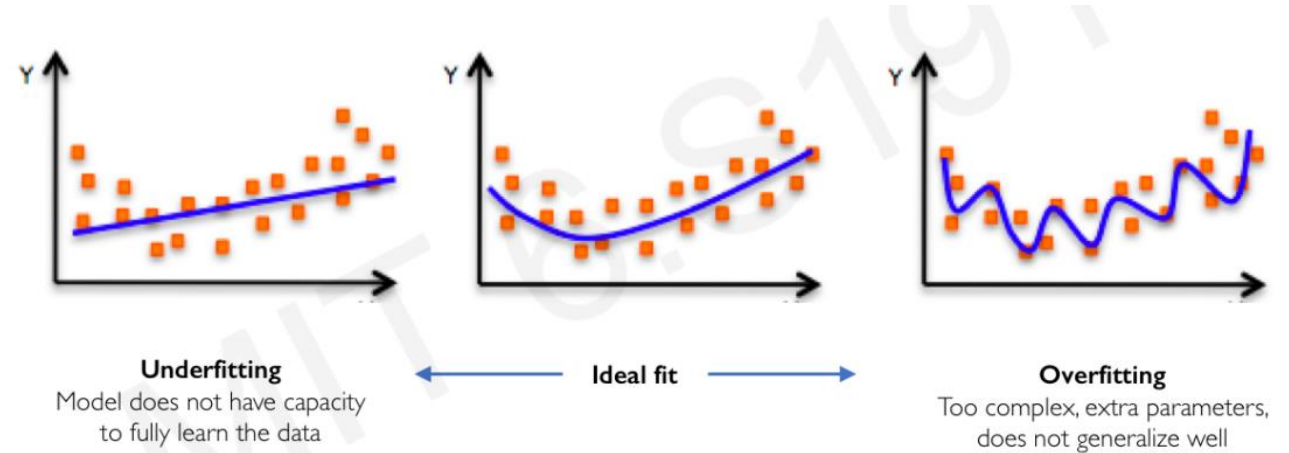
과적합 문제는 모델 성능과 직결됨
과적합을 조절하여 모델을 일반화하는 것이 관건

(1) 과대적합(Overfitting)

- 모델이 학습 데이터에 지나치게 피팅(fitting)되어 있는 상태
- 검증 데이터는 제대로 예측하지 못하는 문제가 발생

(2) 과소적합(Underfitting)

- 모델이 학습 데이터에 지나치게 피팅(fitting)되어 있지 않은 상태
- 이도 저도 아닌 일반화가 제대로 되지 않은 상태



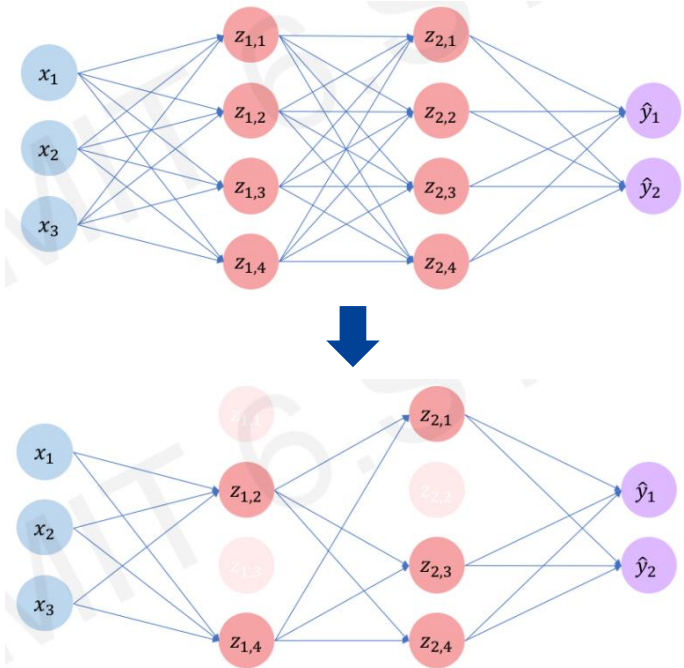
규제(Regularization)

규제

- 모델의 과적합을 방지하기 위한 방법

(1) 드롭아웃(Dropout) 방법

- 은닉층의 일부 노드를 임의로 학습 단계에서 제외



(2) 학습 조기 종료(Early Stopping)

- 학습 과정에서 손실값에 대한 커브가 올라가기 전에 학습을 미리 종료

