파이썬으로 직접 구현하는 머신러닝 / 딥 러닝 실습

Dec 21, 2020

Myoungsung You(famous77@kaist.ac.kr)

Master's Student

Network and System Security Lab

Graduate School of Information Security

KAIST

Index

1. 머신러닝 이론 및 실습
    1) 머신러닝이란?
    2) 손실함수와 경사하강법
    3) 로지스틱 회귀 실습 (MNIST 데이터셋 이진 분류)
2. 딥러닝 이론 및 실습
    1) 퍼셉트론과 인공신경망
    2) Forward/Backward propagation
    3) Multi layer perceptron 실습 (MNIST 손글씨 데이터셋 분류)

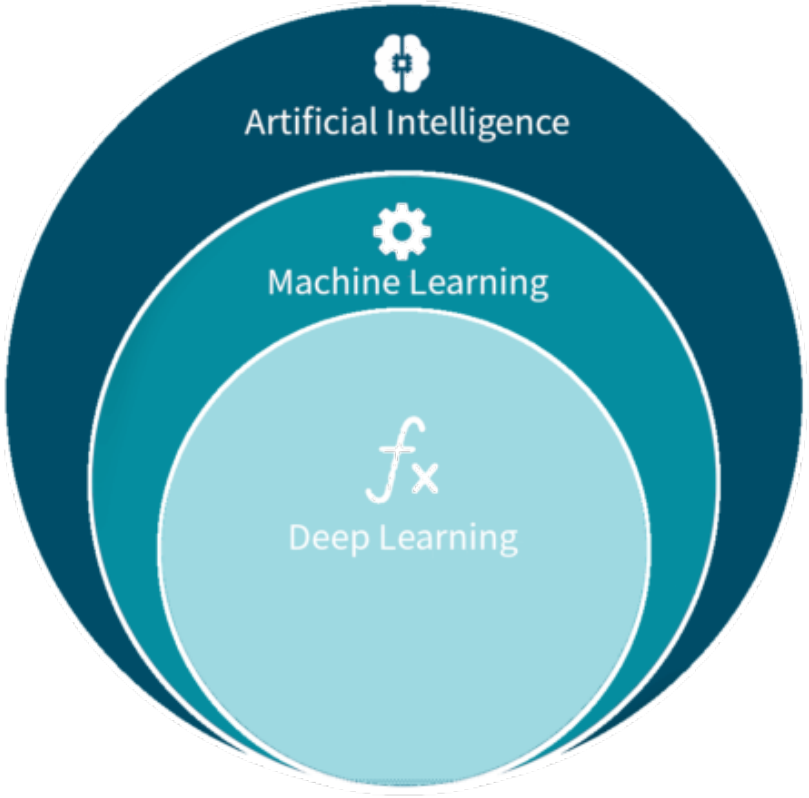
강사소개

강사 **유명성** famous77@kaist.ac.kr

- KAIST 정보보호대학원 석사과정
- 충북대학교 컴퓨터공학과 졸업
- 차세대 보안리더 양성 프로그램 6기 최고인재 선정
- 2019 대한민국 인재상 수상(부총리 겸 교육부장관상)
- 과기부장관상, 행안부장관상, 교육부장관상 등 수상경력 多
- EBS 과학 다큐 비욘드 “해킹 30 초 전” 출연

1. 머신러닝 이론 및 실습

- Artificial intelligence / Machine learning / Deep learning



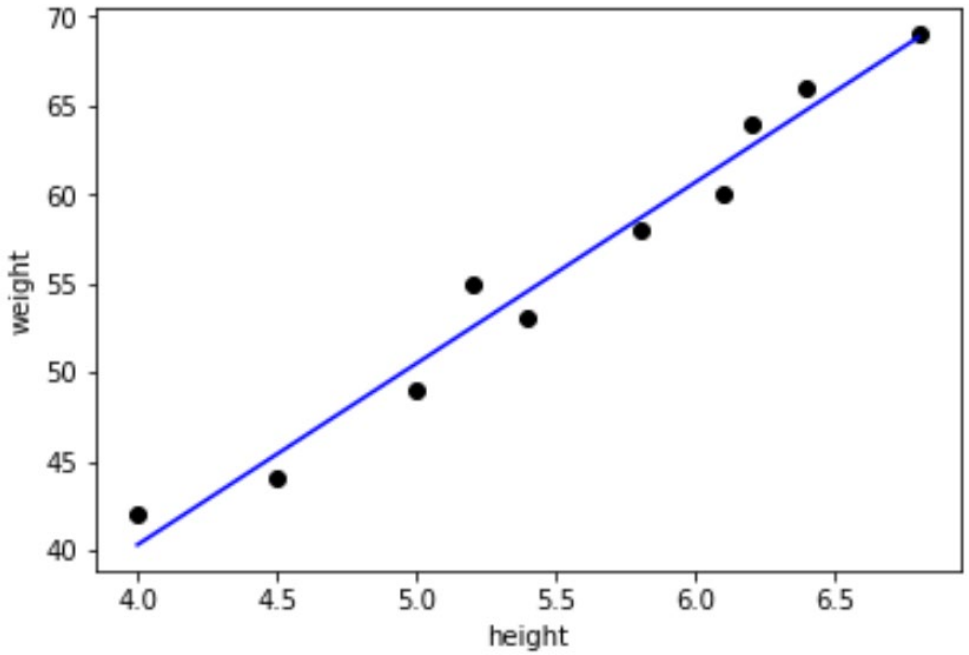
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Machine Learning

- 통계적인 기법을 통해 데이터로부터 특정 작업을 수행할 수 있는 알고리즘을 기계가 스스로 찾아내게 하는 기술



[height에 대한 weight를 예측하는 그래프 찾기]

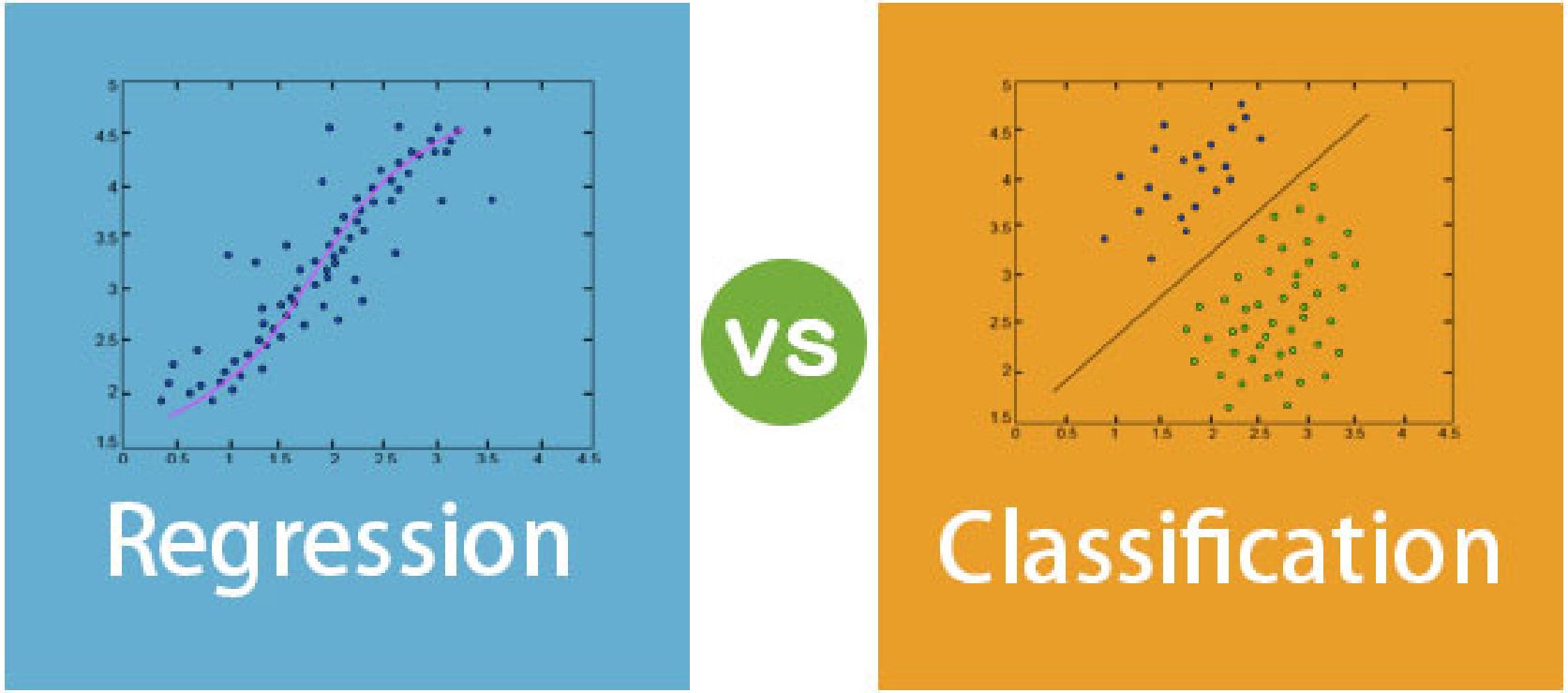
머신 러닝 모델 학습방법

- 지도학습 (Supervised learning)
  - 입력 데이터와 정답을 함께 제공
- 비지도학습 (Unsupervised learning)
  - 입력 데이터로만 학습, 주어진 데이터를 자동으로 분석해 묶어주는 클러스터링
  - Self supervised learning, Autoencoder 등
- 강화학습 (Reinforcement learning)
  - 측정값(관측값)을 입력 받고 행동 방침을 출력하는 학습 방법
  - 출력 단계에서는 정답 유무를 모르지만, 그 결과에 대한 보상으로 정답 여부 확인

지도학습의 회귀와 분류

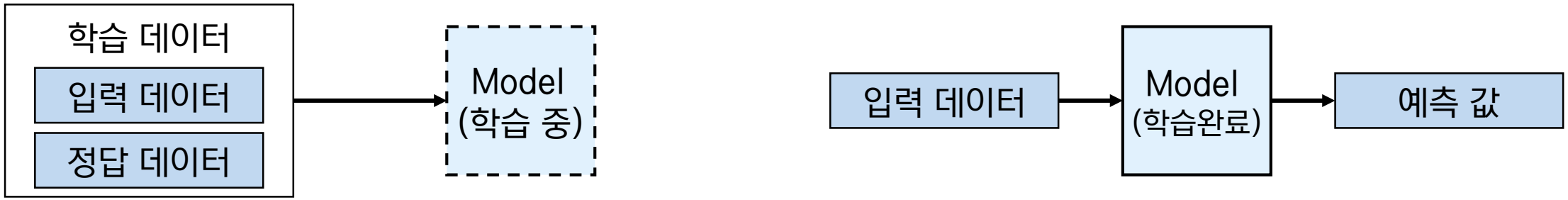
- 분류 (Classification)
  - 모델의 예측 값이 이산 값 (discrete value)으로 나오는 유형
  - 동물 사진 분류, 고혈압 유무 판단
- 회귀 (Regression)
  - 모델의 예측 값이 연속적인 값 (Continuous value)으로 나오는 유형
  - 하루 예상 매출액, 주가, 부동산 가격

- 지도학습의 회귀와 분류



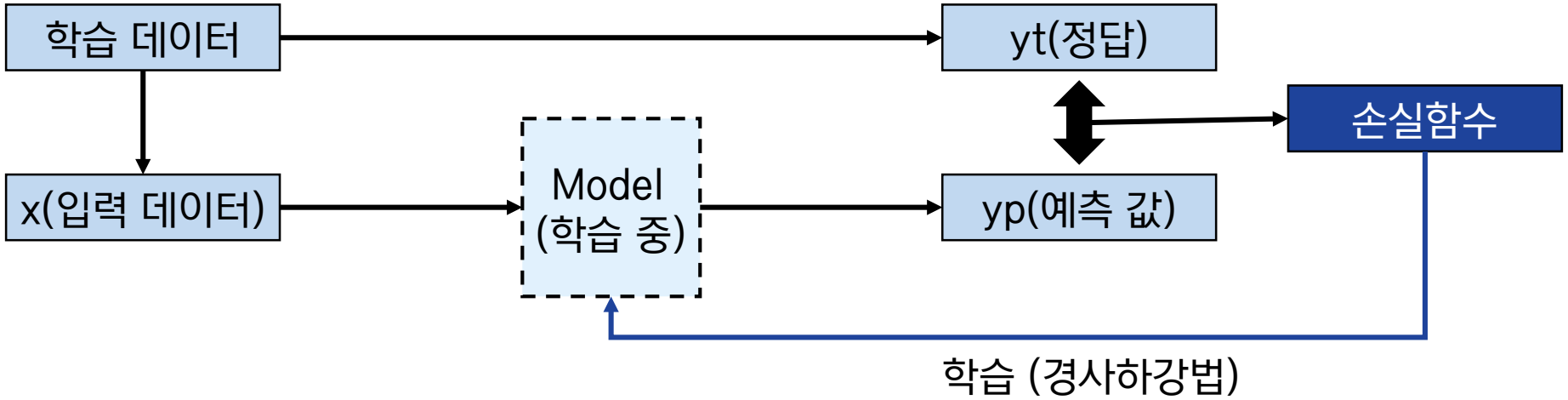
학습 단계와 예측 단계

- 머신러닝 학습은 학습 단계와 예측 단계로 구성
- 학습 단계 : (입력, 정답) 데이터셋을 이용해 모델의 예측 값이 정답과 가까워지도록 모델을 정교하게 만드는 단계
  - 모델의 파라메터 수정
- 예측 단계 : 입력 데이터만 주고 모델이 정답을 예측하는 단계



손실함수와 경사하강법

- 머신러닝 모델은 대부분 손실함수와 경사하강법을 이용하여 학습
- 손실함수 : 모델의 예측 값이 정답과 얼마나 가까운지 계산하는 함수, 오차함수
- 경사하강법 : 손실함수의 값이 최소가 되도록 모델의 매개변수를 조정하는 알고리즘
  - **머신러닝에서 학습은 오차를 최소화하도록 모델의 매개변수를 수정하는 것**

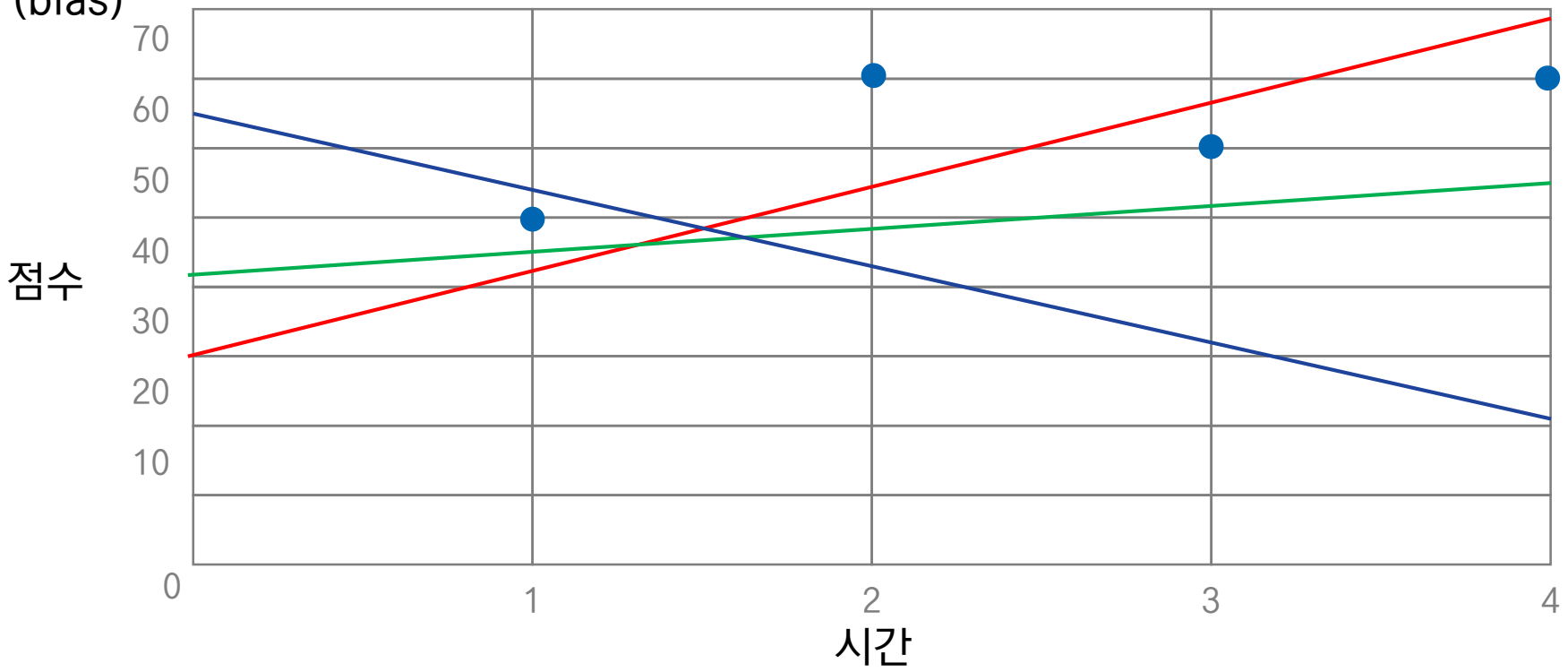


선형회귀 (Linear Regression)

- 종속 변수 y와 한 개 이상의 독립 변수 x와의 선형 상관 관계를 모델링하는 기법
- 단순 선형회귀 분석 : $y = Wx + b$, 하나의 직선을 통해 모델링
- 다중 선형회귀 분석 : $y = W_1x_1 + \cdots W_nx_n + b$, 여러 직선 사용

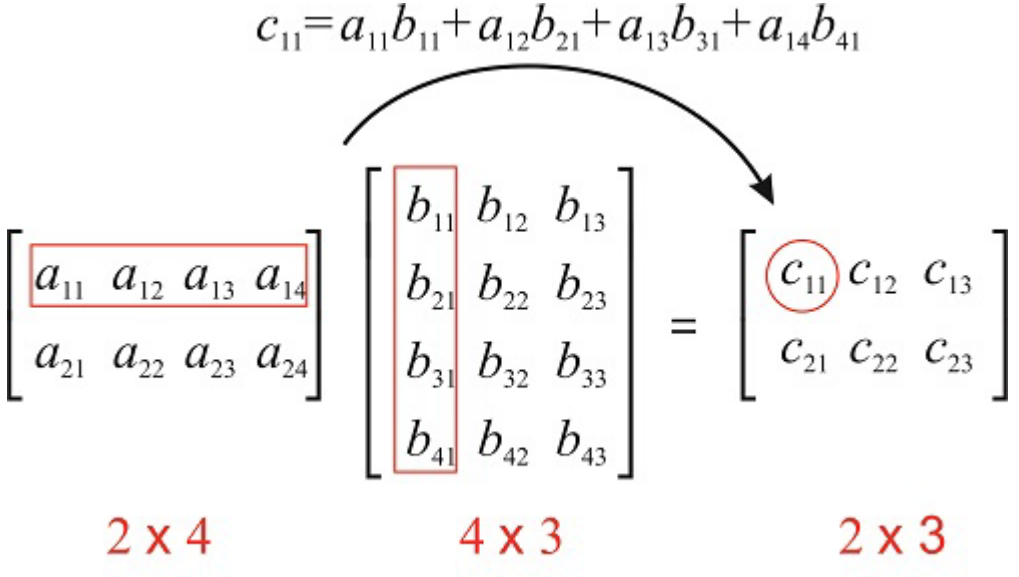
선형회귀 가설(Hypothesis) 세우기

- 주어진 데이터로부터 x, y 관계에 대한 가설 수립 $\mathrm{H}(x) = Wx + b$
- W : 가중치 (weight), b : 편향 (bias)



벡터 연산

- 독립 변수와 가중치들을 벡터를 통해 간단하게 표현할 수 있음
- 독립 변수 벡터(입력 데이터)와 가중치 벡터들의 곱의 합은 벡터의 내적으로 계산
- $\mathrm{H}(x) = W_1 x_1 + W_2 x_2 \ \ldots + W_n x_n + b$
- $\mathrm{H}(X) = XW + B$



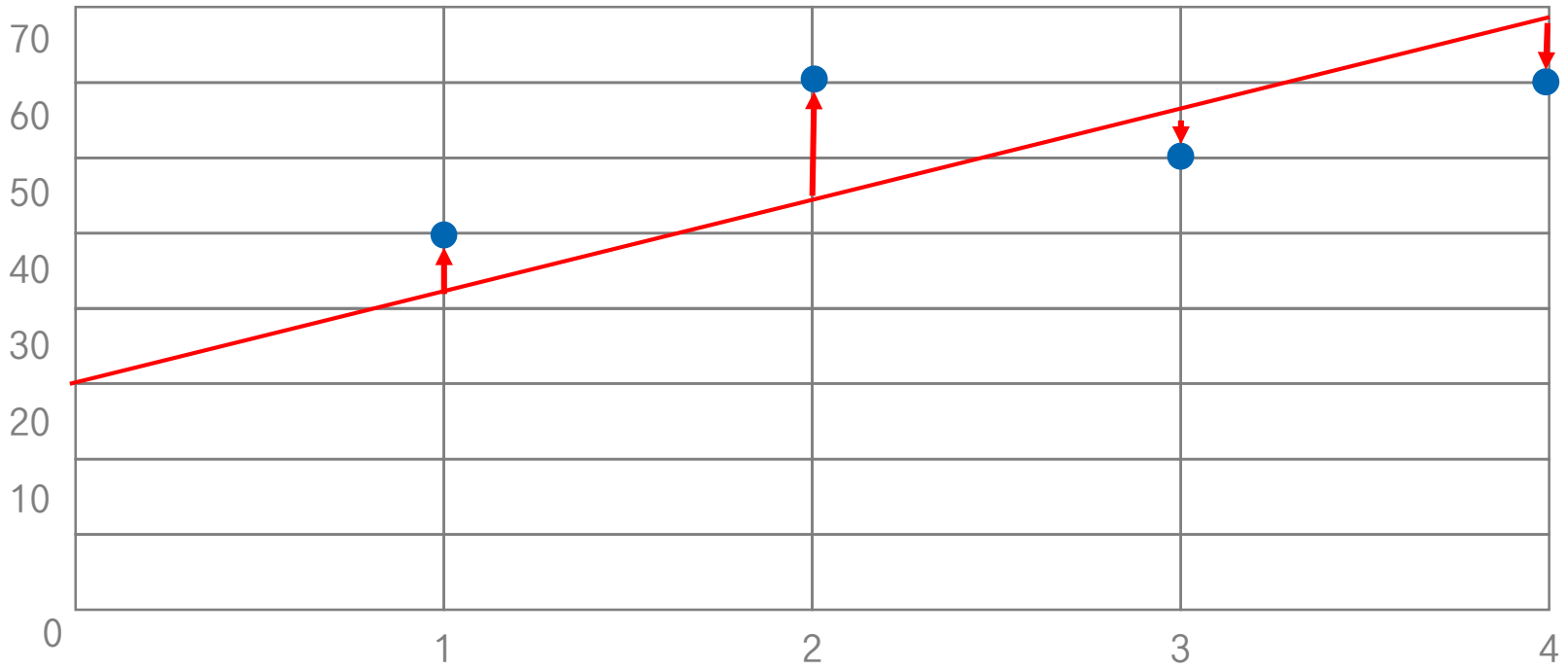
가설에 대한 오차 구하기

- 비용함수를 이용해 측정값과 실제값의 오차 계산 (평균제곱오차, Mean square error)

$$\mathrm{H}(x) = Wx + b$$

$$\mathrm{Loss}(W,b) = \frac{1}{n}\sum_{i=1}^{n}\left(y^i - H(x^i)\right)^2$$

$$W, b \rightarrow minimize\ Loss(W,b)$$



오차를 바탕으로 가설의 파라메터 갱신

- 경사하강법 (Gradient descent)을 사용해 반복적으로 계산하며 W 값을 갱신
  - Loss를 최소화하는 가중치를 찾는 것이 목표
  - 해당 가중치가 Loss에 미친 영향도를 계산 ($\frac{\partial Loss}{\partial W}$, gradient)
  - Gradient를 바탕으로 가중치를 새로운 값으로 갱신
    - $W_{new} = W_{old} - \mu \frac{\partial Loss}{\partial W}$
  - 정해진 시간만큼 또는 모델의 출력에 대한 Loss가 일정 값 이하로 작아질 때 까지 반복

오차함수를 가중치로 미분해 gradient 계산

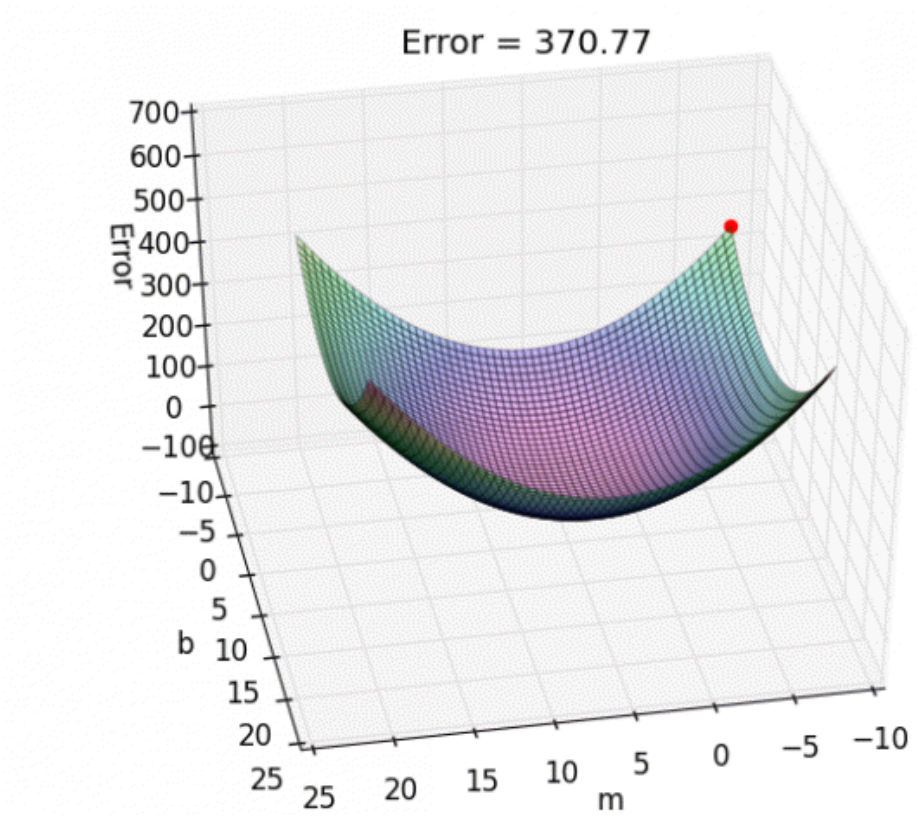
- 학습의 목표는 오차함수의 값을 최소화하는 파라메터 (가중치, 편향)을 찾는 것
- $L(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left((Wx_i + b) - t_i\right)^2$
- $J(W,b) = argmin\left(Loss(W,b)\right)$
- $\frac{\partial J(W,b)}{\partial W} = \frac{\partial}{\partial W}\left(\frac{1}{m}\sum_{i=1}^{m}\left((Wx_i + b) - t_i\right)^2\right) = \frac{2}{m}\sum_{i=1}^{m}\left((Wx_i + b) - t_i\right)x_i$
- $\frac{\partial J(W,b)}{\partial b} = \frac{\partial}{\partial b}\left(\frac{1}{m}\sum_{i=1}^{m}\left((Wx_i + b) - t_i\right)^2\right) = \frac{2}{m}\sum_{i=1}^{m}\left((Wx_i + b) - t_i\right)$
- $W_{new} = W_{old} - \mu\frac{\partial J(W,b)}{\partial W}$
- $b_{new} = b_{old} - \mu\frac{\partial J(W,b)}{\partial b}$

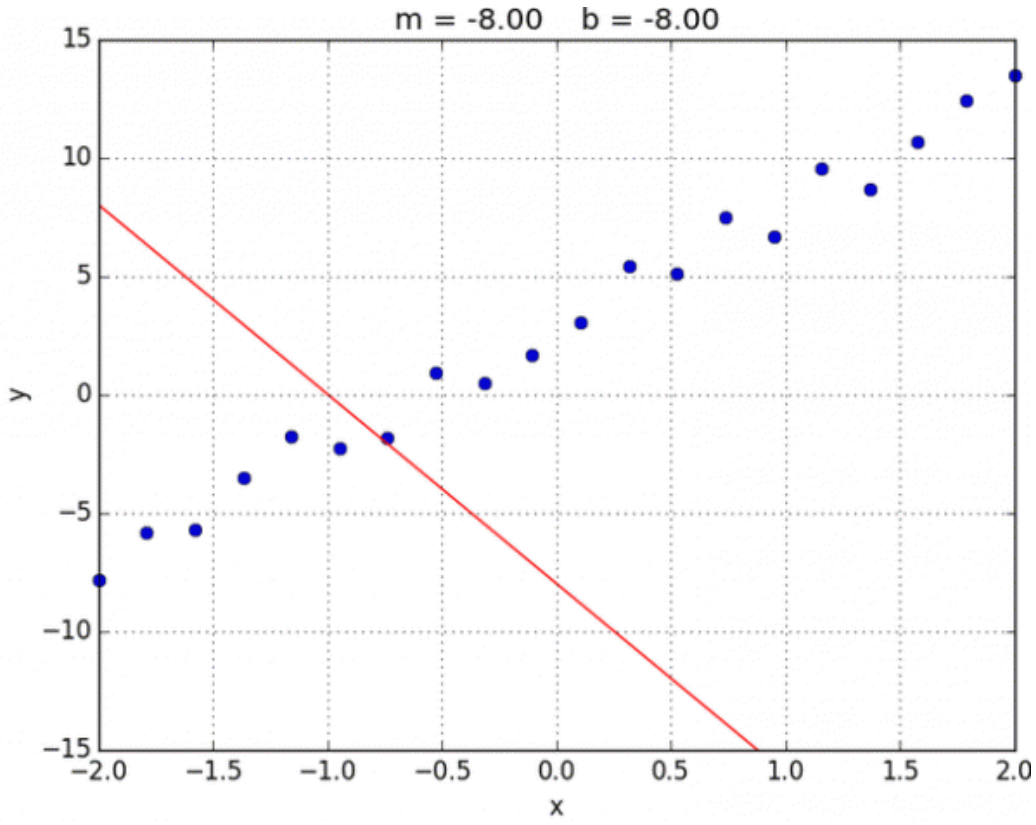
경사하강법을 통한 가중치 수정 예제

- $x_1 = 2,\ x_2 = 3,\ \ t_1 = 46,\ \ t_2 = 72,\ \mu = 0.2,\ w = 15, b = 2$
- $y_1 = wx_1 + b = 15 * 2 + 2 = 32$
- $Loss(y_1, t_1) = \frac{1}{2}(wx_1 + b - t_1)^2 = 196$
- $\frac{\partial J(w,b)}{\partial w} = \big((wx_1 + b) - t_1\big)x_1 = (32 - 46) * 2 = -28$
- $\frac{\partial J(w,b)}{\partial b} = (wx_1 + b) - t_1 = (32 - 46) = -14$
- $w_{new} = w_{old} - \mu \frac{\partial J(w,b)}{\partial w} = 15 - 0.2 * -28 = 20.6$
- $b_{new} = b_{old} - \mu \frac{\partial J(w\ b)}{\partial b} = 2 - 0.2 * -14 = 4.8$
- $y_2 = wx_2 + b = 20.6 * 3 + 4.8 = 66.6$

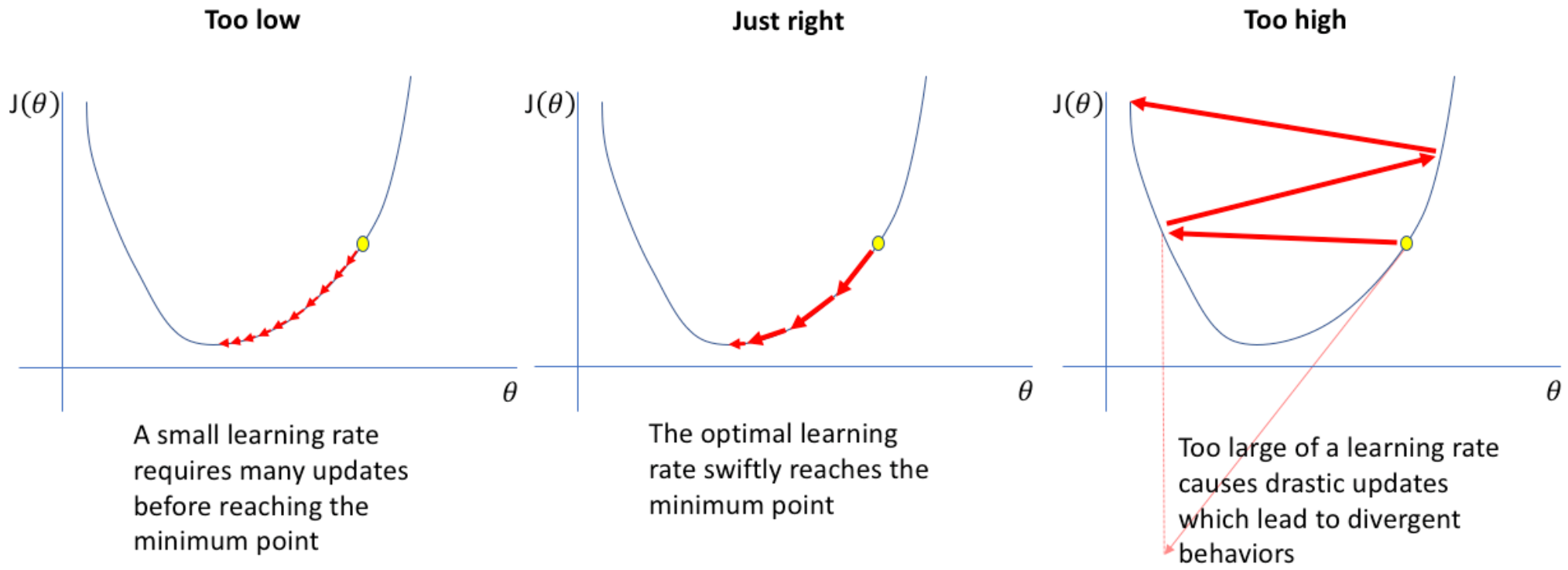
w, b는 학습을 통해 구할 수 있는 parameter
μ는 학습을 통해 구할 수 없는 parameter

경사하강법에 의한 가설의 변화



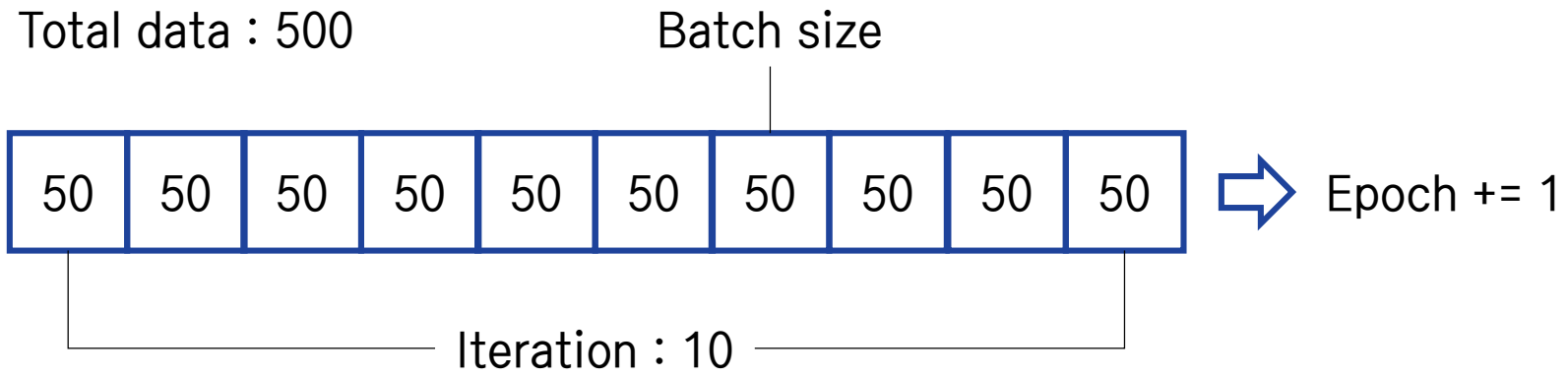


학습률(learning rate)에 따른 변화



Batch, Epoch, Iteration

- Epoch : 신경망에서 전체 데이터에 대해 순/역전파가 끝난 상태
- Batch size : 매개변수를 업데이트하는 데이터의 단위
- Iteration : 한 번의 Epoch를 끝내기 위해 필요한 Batch의 수



Batch GD (plain or vanilla GD)

1. 가중치를 랜덤한 값으로 초기화
2. Loss가 수렴할 때까지 반복
    1. 출력에 대한 Loss 계산
    2. 각 가중치에 대한 gradient 계산,
    3. 각 가중치를 gradient를 기반으로 업데이트

$$W_{new} = W_{old} - \mu \frac{1}{B} \sum_{i=1}^{B} \left((W x_i + b) - t_i \right) x_i$$

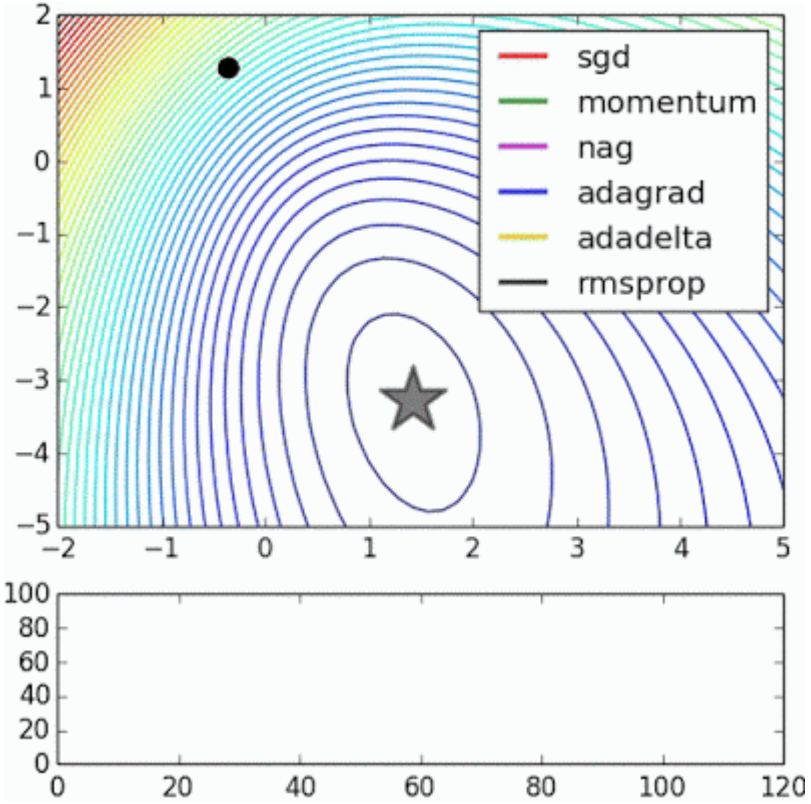
Mini-Batch GD

1. 가중치를 랜덤한 값으로 초기화
2. Loss가 수렴할 때까지 반복
    1. 전체 데이터 셋에서 배치 사이즈 (B_m) 만큼 랜덤한 샘플 추출
    2. 선택한 배치에 대한 출력과 Loss 계산
    3. 각 가중치에 대한 gradient 계산,
    4. 각 가중치를 gradient를 기반으로 업데이트

$$W_{new} = W_{old} - \mu \frac{1}{B_m} \sum_{i=1}^{B_m} \left((W x_i + b) - t_i \right) x_i$$

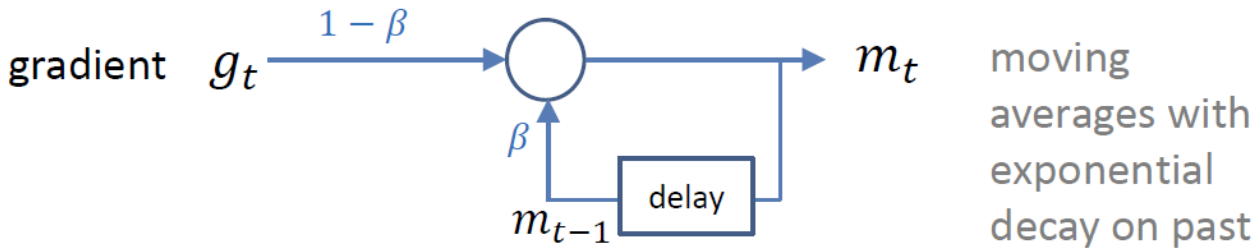
경사하강법의 종류

- Batch Gradient Descent (Vanilla GD)
- Stochastic Gradient Descent (SGD)
  - 전체 데이터를 랜덤하게 샘플링해 작은 단위로 학습
  - Mini-Batch Gradient Descent (Mini-batch SGD)
  - SGD with Momentum
    - Gradient의 이동평균을 이용
  - Adam
    - Gradient 및 Gradient^2의 이동평균 이용
    - 학습율을 자동으로 조절



SGD with momentum

- Gradient를 직접 사용해 가중치를 업데이트하지 않고 gradient의 이동평균을 사용
- 진행하던 방향으로 관성을 가지고 움직으로 안정적인 업데이트 가능



$$m_t = \beta m_{t-1} + (1-\beta) g_t$$

$$w^{\text{new}} = w^{\text{prev}} - \mu m_t$$

Adam optimizer

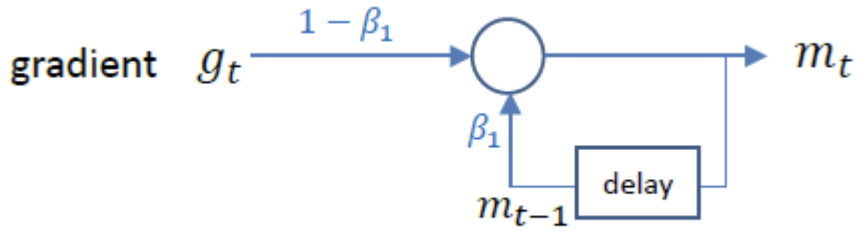
- Adaptive moment estimation, 학습율을 적절히 변경하면서 가중치를 업데이트
- 실제 ML 응용프로그램에서 매우 잘 동작

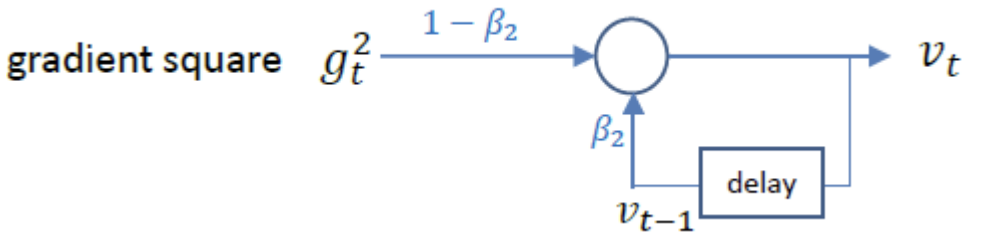
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

gradient MA

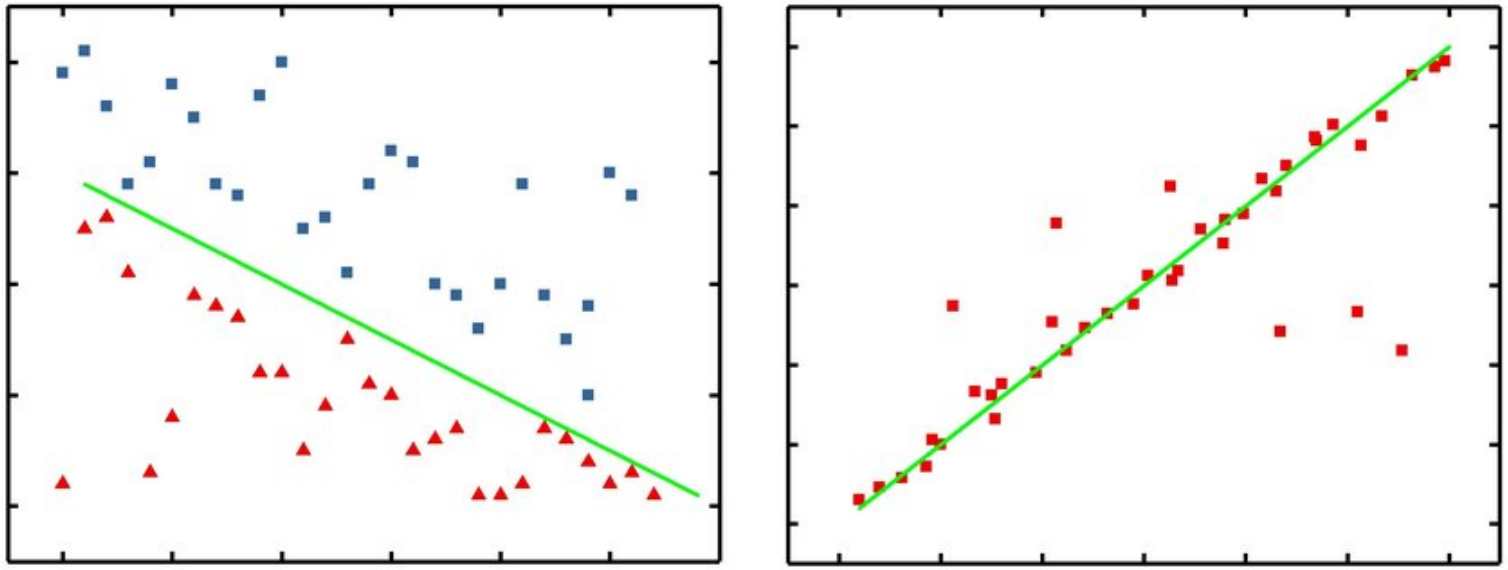
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$





로지스틱 회귀 (Logistic Regression)

- 독립 변수와 종속 변수의 관계를 이용해 사건의 발생 가능성 (확률)을 예측
  - Classification에 주로 사용, Binary classification 등

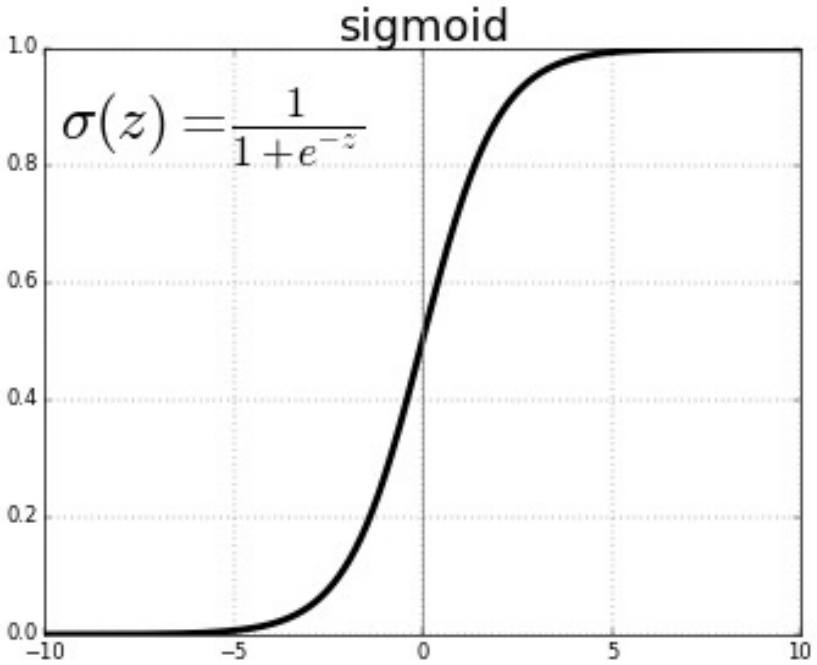


(a) Logistic Regression

(b) Linear Regression

시그모이드 함수 (Sigmoid Function)

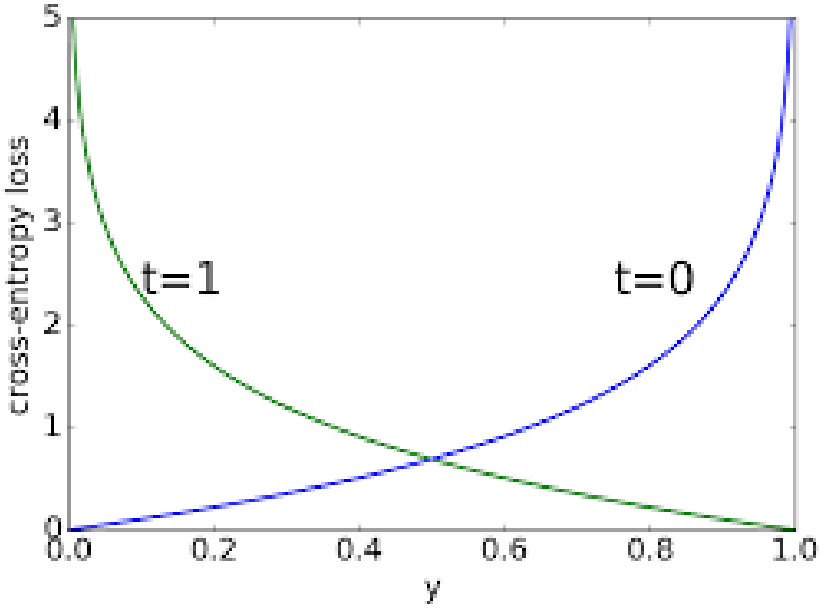
- $\mathrm{H}(x) = \frac{1}{1+e^{-(Wx+b)}} = sigmoid(Wx+b) = \sigma(Wx+b)$
- 0 ~ 1 사이의 출력을 갖는 S자 형태의 함수
- 출력값을 이진분류 문제 예측에 사용 -> 확률 값처럼 이용
  - 0.5 이상일 경우 Class 1
  - 0.5 미만일 경우 Class 0
- $\mathrm{H}(x)$의 W, b를 구하는 것은 linear regression과 동일



로지스틱 회귀의 Loss Function

- Cross entropy loss function
- 정답 (t)가 1일 경우 sigmoid의 출력 (H(WTx))이 0에 가까울 수록 cost function의 값이 커짐
- 정답 (t)가 0일 경우 sigmoid의 출력 (H(WTx))이 1에 가까울 수록 cost function의 값이 커짐

- $J(W) = \frac{1}{n}\sum_{i=1}^{n} Loss(H(W^T x^i), y^i)$
- $if\ t = 1\ then\ Loss(H(x), t) = -\log(H(W^T x))$
- $else\ Loss(H(W^T x), t) = -\log(1 - H(W^T x))$
- $Loss(H(x), y) = -(t log(H(W^T x)) + (1 - t)\log(1 - H(W^T x)))$
- $J(W) = -\frac{1}{n}\sum_{i=1}^{n} \left(t^i \log\left(H(W^T x^i)\right) + \left(1 - t^i\right) \log\left(1 - H(W^T x^i)\right)\right)$



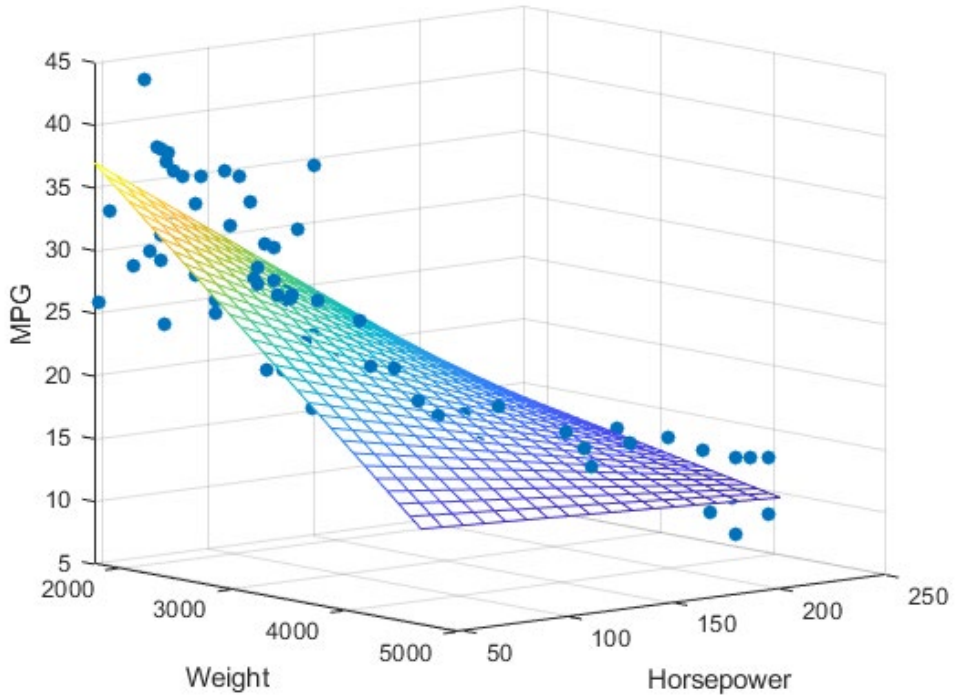
Binary cross entropy gradient

- $Z = w^T X,\ Y = \sigma(Z)$
- $\sigma(Z) = \frac{1}{1+e^{-Z}}, \acute{\sigma}(Z) = \sigma(Z)\left(1-\sigma(Z)\right)$
- $J(W,X) = -\frac{1}{n}\sum_{i=1}^{n}\left(t^i \log\left(\sigma\left(w^T x^i\right)\right) + (1-t)\log\left(1-\sigma\left(w^T x^i\right)\right)\right)$
- $\frac{\partial J}{\partial W} = \frac{\partial Z}{\partial W}\frac{\partial Y}{\partial Z}\frac{\partial J}{\partial Y}$ *(chain rule)*
- $\frac{\partial J}{\partial y_i} = -\frac{t_i}{y_i} + \frac{1-t_i}{1-y_i}$
- $\frac{\partial J}{\partial W} = \frac{1}{n}\sum_{i=1}^{n} x_i y_i (1-y_i)\left(-\frac{t_i}{y_i} + \frac{1-t_i}{1-y_i}\right) = \frac{1}{n}(Y-T)X$

다중 입력 변수에 대한 회귀

- 종속변수 y에 대해 2개 이상의 독립변수 x가 있는 경우
- 입력 벡터의 차원과 가중치 벡터의 차원이 1이 아닌 n차원
- $\mathrm{H}(x) = W_1 x_1 + W_2 x_2 \ \ldots + W_n x_n + b$
- $\mathrm{H}(x) = \sigma(W_1 x_1 + W_2 x_2 \ \ldots + W_n x_n + b)$

중간고사	기말고사	프로젝트	성적
100	95	95	96
80	80	75	78
60	100	100	86
55	75	100	76

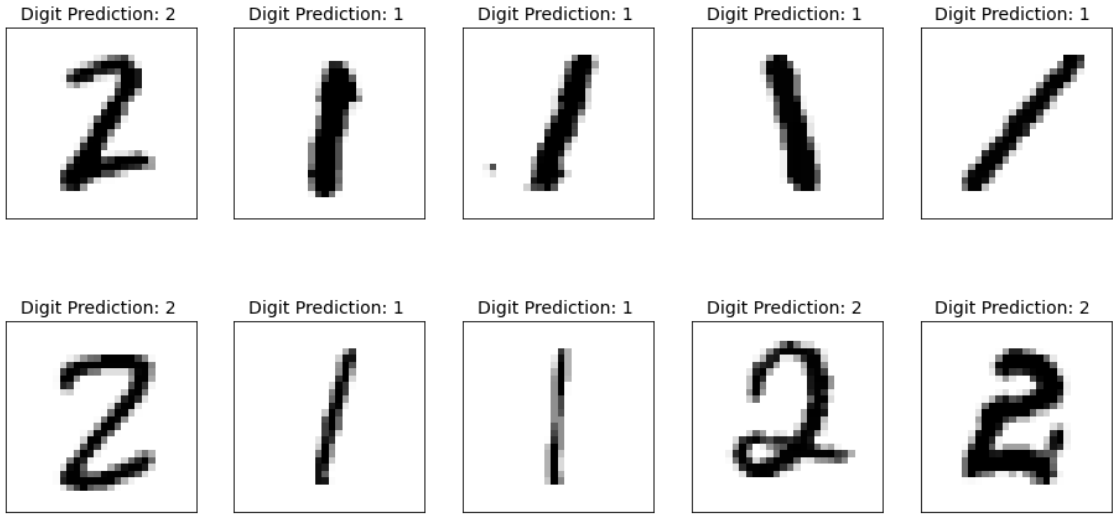


다양한 머신러닝 알고리즘

- 지도학습
  - K-최근접 알고리즘
  - Support vector machine (SVM)
  - 결정 트리 (Decision tree)
  - 로지스틱 회귀 (Logistic regression)
- 비지도학습
  - K-means 클러스터링
  - DBSCAN
  - agglomerative 클러스터링
  - Principal component analysis (PCA)

Logistic regression 직접 구현하기 (logistic.ipynb)

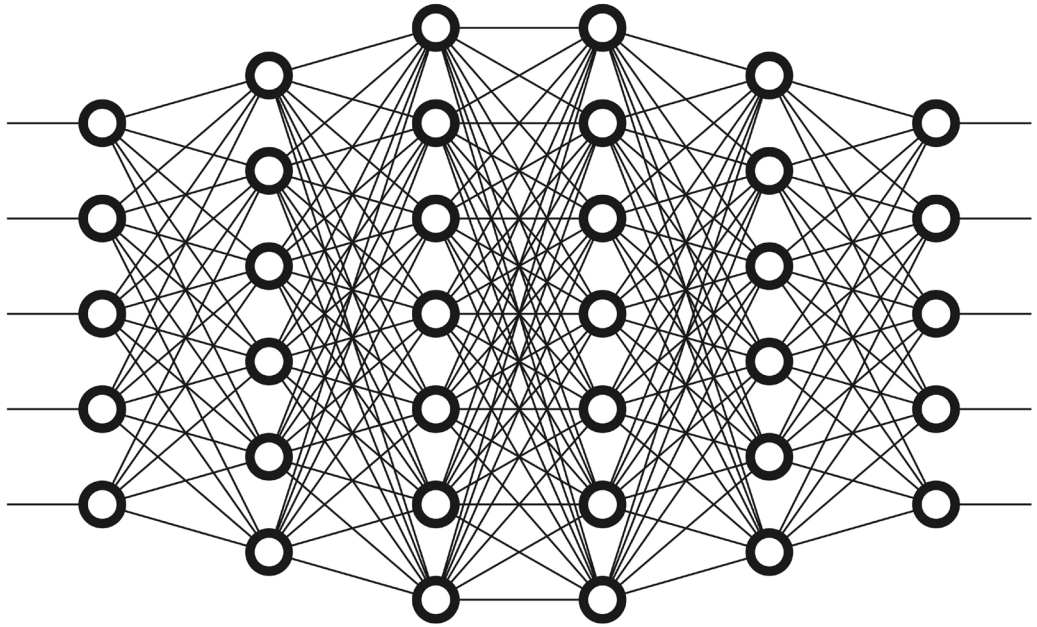
- Machine learning library를 사용하지 않고 numpy (수학연산) library만 사용해 logistic regression 구현
- 구현한 모델을 사용해 MNIST 손 글씨 데이터셋 분류
  - Logistic regression은 이진 분류만 가능하기 10개의 클래스 중 2가지만 사용



2. 딥 러닝 이론 및 실습

딥 러닝(Deep Learning)

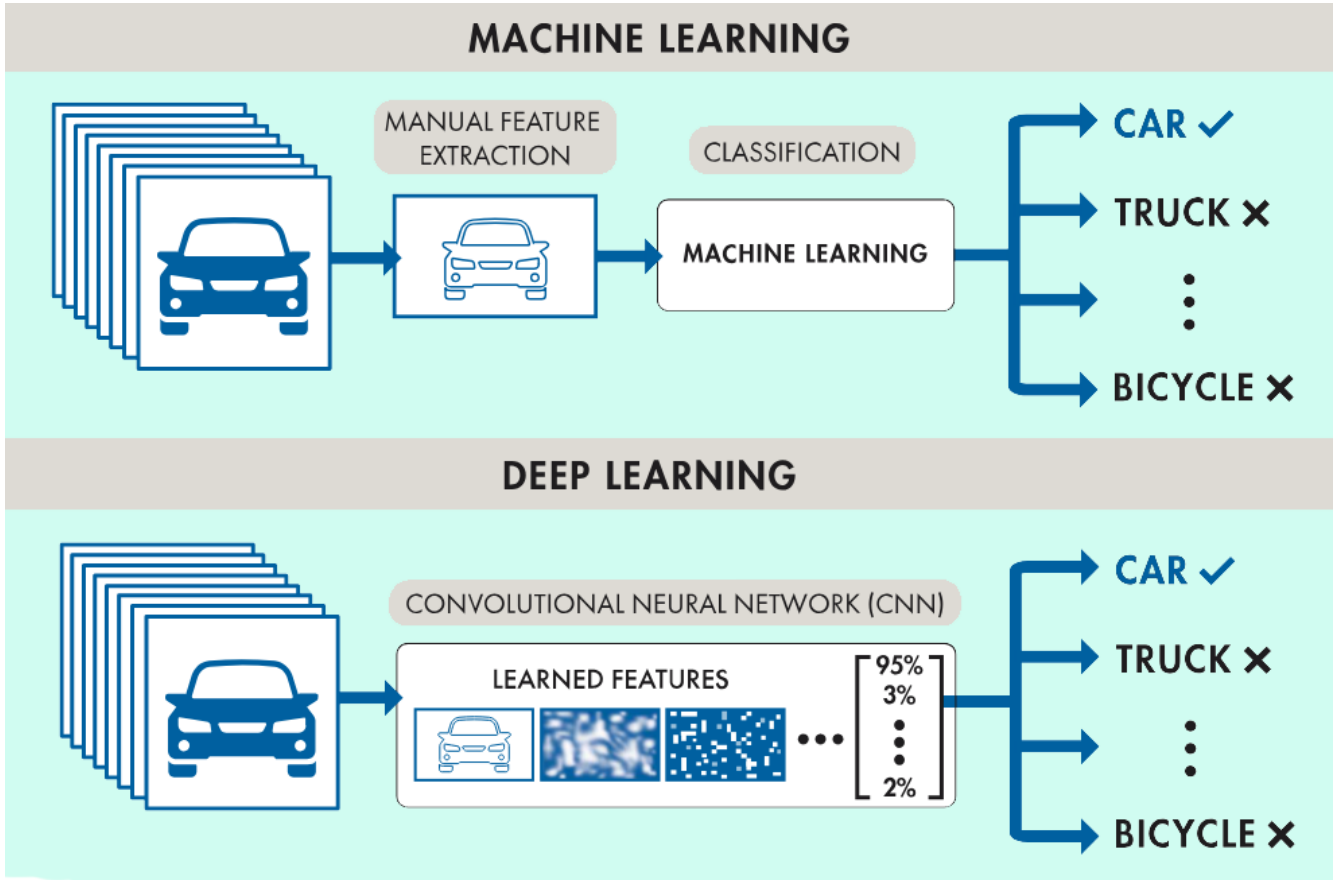
- 많은 양의 신경층 (Neural Network)에 입력된 데이터가 여러 층을 거치면서 특징이 추출되고, 자동으로 추상적인 지식을 추출해 모델을 학습시키는 방법
- 데이터의 특징 추출과 분류가 하나의 과정으로 통합됨
- 신경층의 깊이에 제약이 있었지만 GPU 등의 발전으로 극복
- 영상처리, 자연어처리 등 다양한 분야에 응용



2. 딥 러닝 이론 및 실습

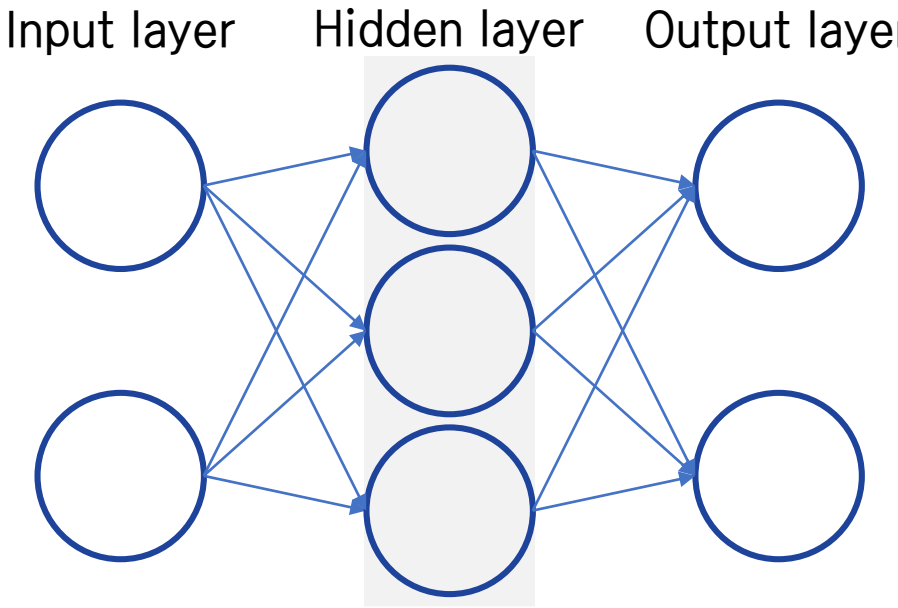
딥 러닝과 머신러닝의 차이

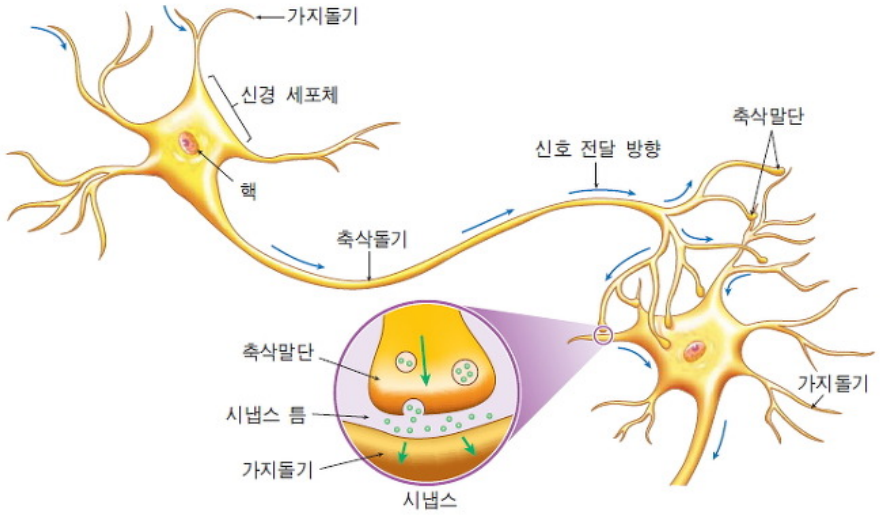
- 머신러닝의 경우 학습 데이터에서 학습에 사용할 특성 (feature)을 수동으로 찾아서 사용
- Domain expert가 데이터를 세세히 분석하여 feature를 추출 -> feature engineering
- 딥 러닝은 학습에 사용할 특성을 자동으로 찾아서 사용 (feature를 찾는 것까지 학습)
- 딥 러닝은 머신러닝에 비해 학습 시간이 매우 길고 모델 자체가 무거움



인공 신경망 (Neural Network)

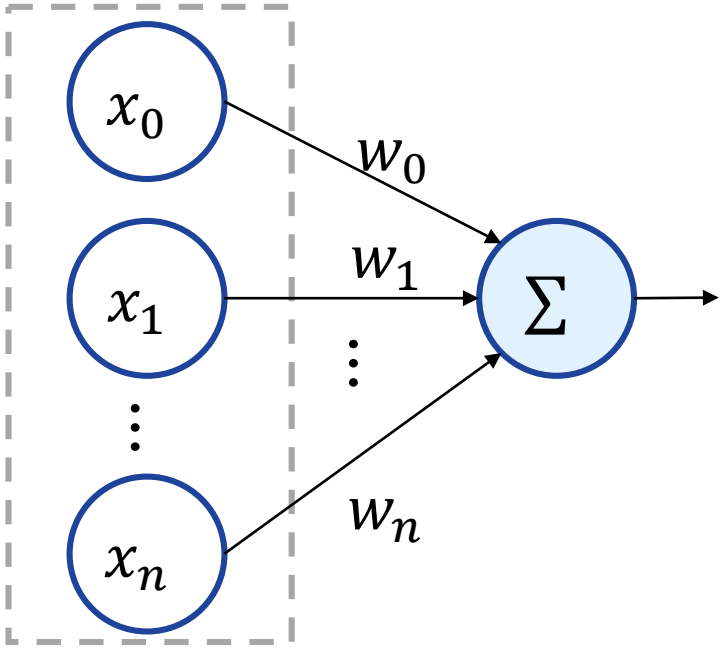
- 신경망은 여러 퍼셉트론을 서로 연결한 일종의 Network로, 데이터를 통해 스스로 학습하여 가중치를 찾아낸다.
- 신경망이 학습한다는 것은 특정 데이터에 맞게 가중치를 찾아 업데이트 하는 것을 의미한다.
- 은닉층의 동작은 입력/출력층과 달리 보이지 않기 때문에 은닉(hidden)층으로 불린다.





퍼셉트론 (Perceptron)

- 다수의 신호를 입력 받아 하나의 신호를 출력하는 알고리즘, 인간의 뉴런을 모델링
- 입력 값과 가중치의 곱의 합이 임계치 이상이면 출력을 다음 뉴런으로 전달



$$\sum_{0}^{n} w_i\, x_i > \theta \Rightarrow 1$$

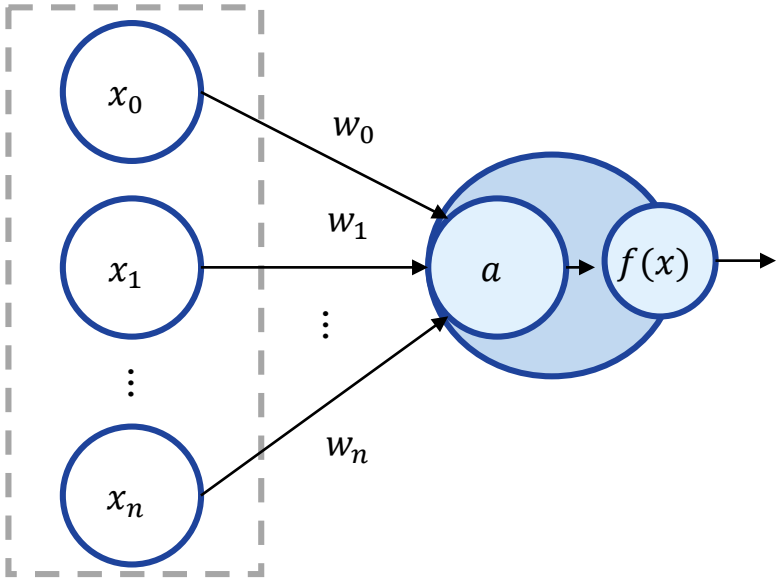
$$\sum_{0}^{n} w_i\, x_i + bias < 0 \Rightarrow 1$$

$$f(x) = \begin{Bmatrix} 0 \left(\sum_{0}^{n} w_i\, x_i \leq \theta \right) \\ 1 \left(\sum_{0}^{n} w_i\, x_i > \theta \right) \end{Bmatrix}$$

가중치(w_i) : 입력신호의 비중
편향($bias$) : 모델의 활성화 정도

Activation Function

- 활성화 함수는 입력 신호의 총합을 출력 신호로 변환하는 함수이다.
- 입력 받은 신호를 얼마나 출력 노드로 전달할지 결정한다.
- 신경망에서 활성화 함수는 반드시 “비선형” 함수여야 한다.



$$h(x) = cx$$

$$f(x) = h\Big(h\big(h(x)\big)\Big) = c * c * c * x = c^3x$$

* 활성화 함수가 선형 함수이면, 신경망 적층의 의미가 없다.

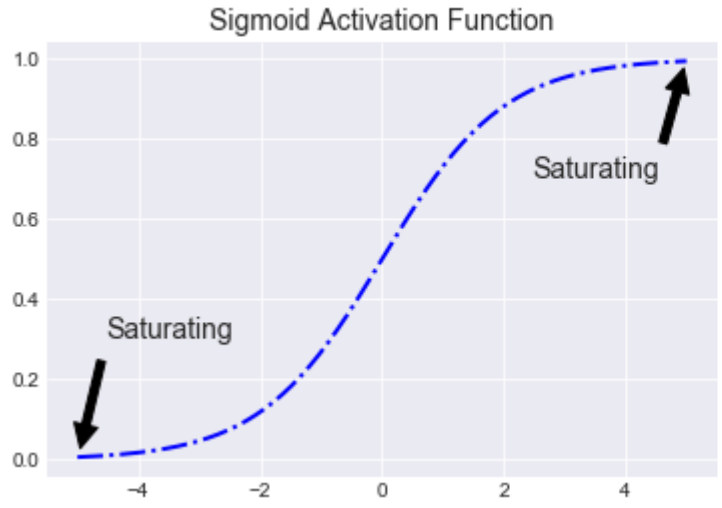
Sigmoid function

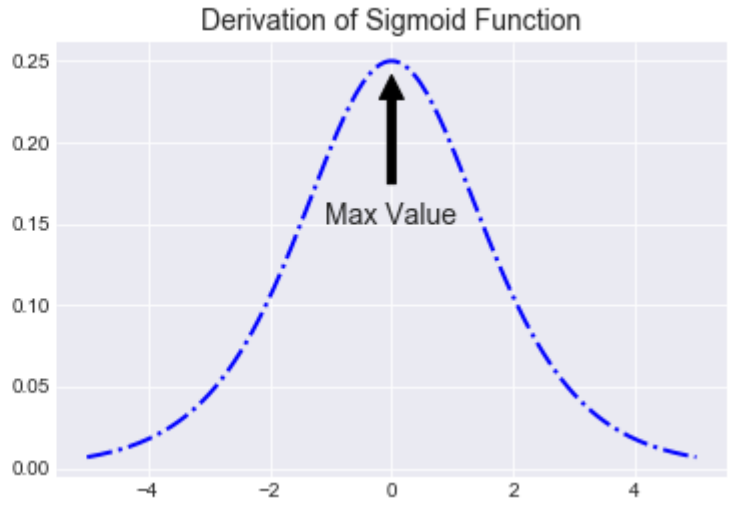
- 시그모이드 함수는 계단 함수에 비해 완만한 곡선 형태 출력을 갖는 비선형 함수이다.
- 함수값은 0 ~ 1 사이이며, 중간값은 $\frac{1}{2}$이다.
- 매우 큰 값을 가지면 함수값은 거의 1, 매우 작은 값을 가지면 거의 0이다.

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

$$f'(t) = f(t)(1 - f(t))$$

```
def sigmoid(z):
    return 1/ (1+np.exp(-z))

def sigmoid_derivation(z):
    return sigmoid(z)*(1-sigmoid(z))
```



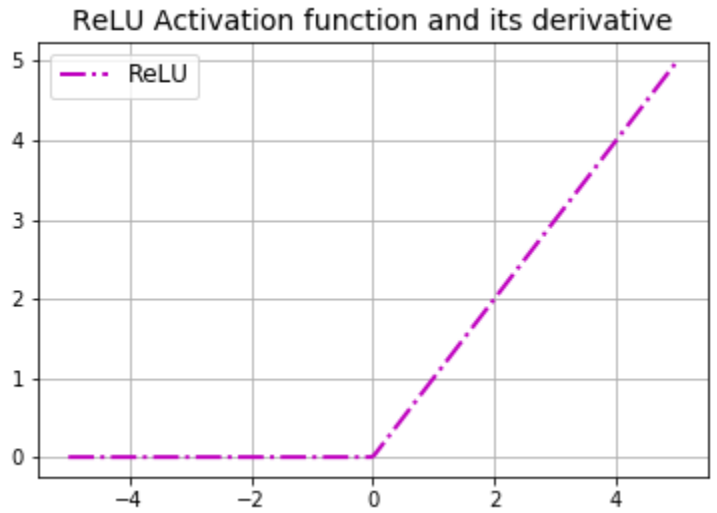


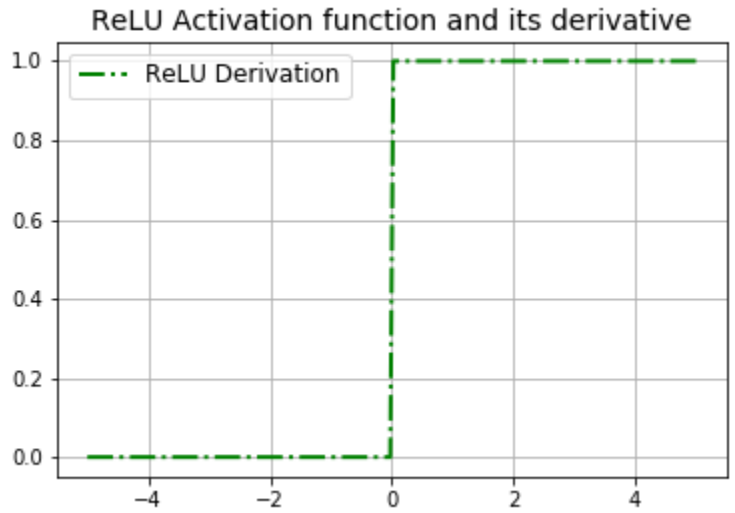
ReLU function

- ReLU (Rectified Linear Unit) 함수는 x > 0 이면 기울기가 1인 직선, x < 0 이면 함수값이 0이다.
- 타 활성화 함수에 비해 간단하기 때문에 학습 속도가 빠르다.
- x < 0 인 값들에서 기울기가 0이기 때문에 뉴런이 죽는 문제가 있을 수 있다.

$$f(x) = \max(0, x)$$
$$f'(x) = \text{step}(x)$$

```
def relu(z):
    return np.maximum(0, z)

def derivative_relu(z):
    return (z >= 0).astype(z.dtype)
```





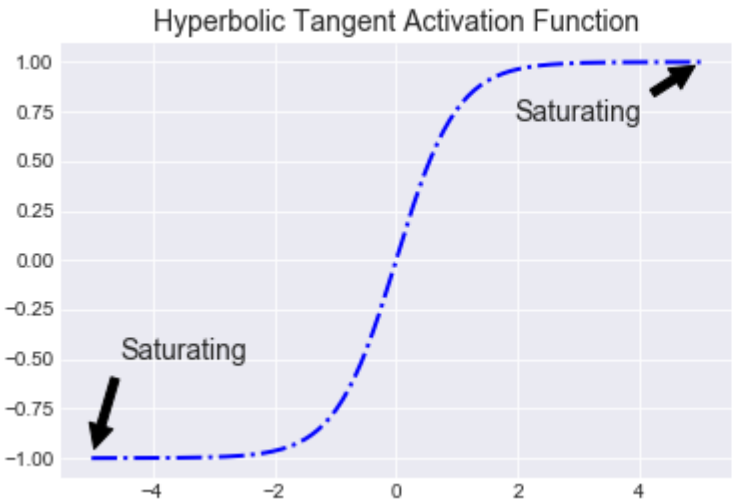
Hyperbolic tangent function

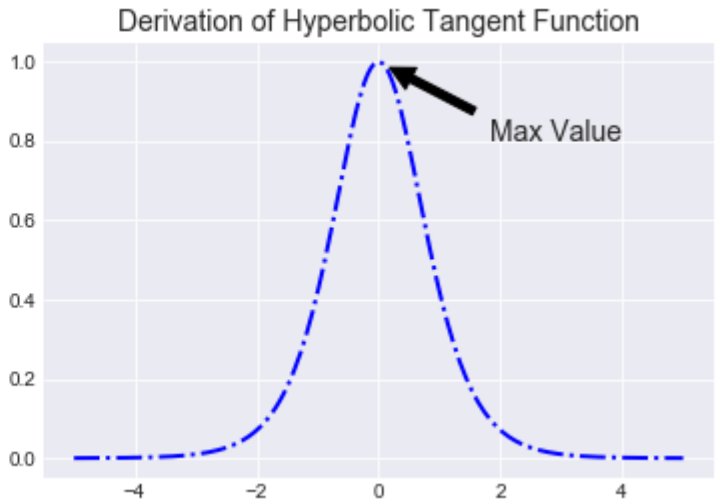
- Sigmoid 함수의 rescaled 버전
- 미분한 값의 범위가 0 ~ 1이기 때문에 Sigmoid 함수보다 gradient vanishing이 덜함
- RNN 등에서 많이 사용

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

$$f'(x) = 1 - \tanh(x)^2$$

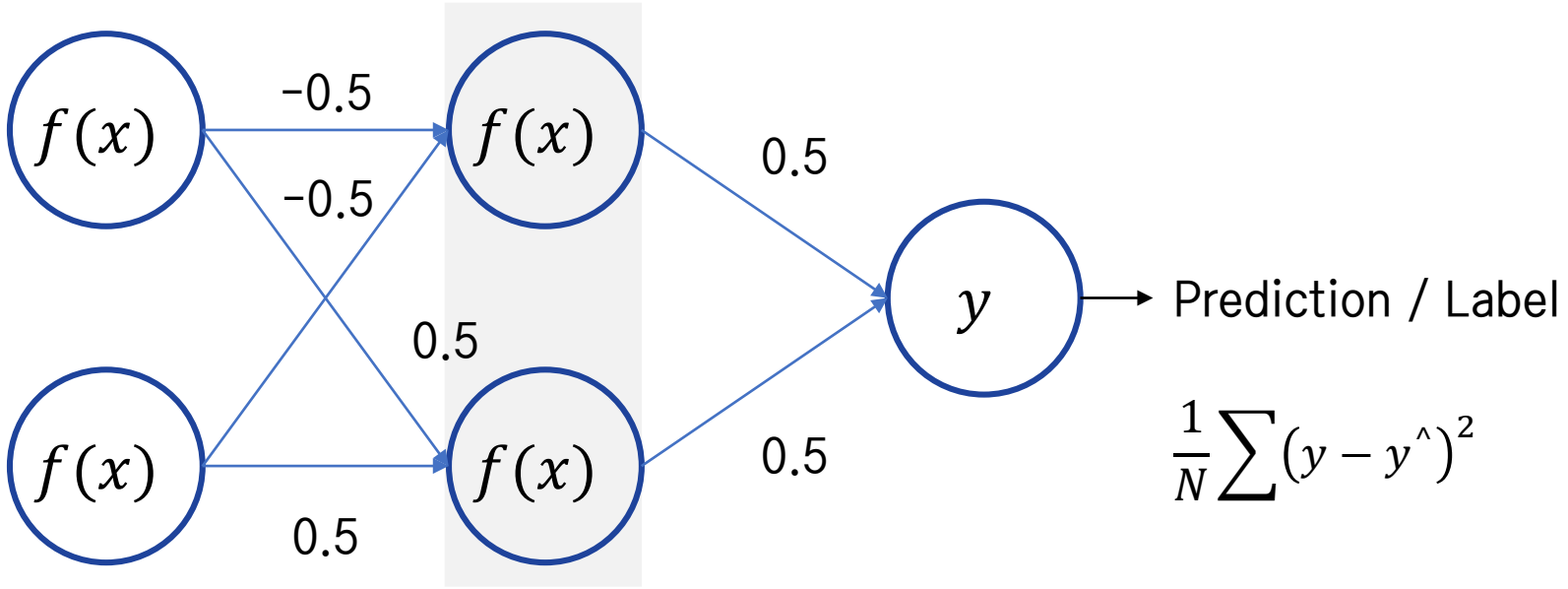
```
def tanh(z):
    return (np.exp(z)-np.exp(-z))/(np.exp(z)+np.exp(-z))

def tanh_derivation(z):
    return (1- tanh(z)**2)
```





Loss Function

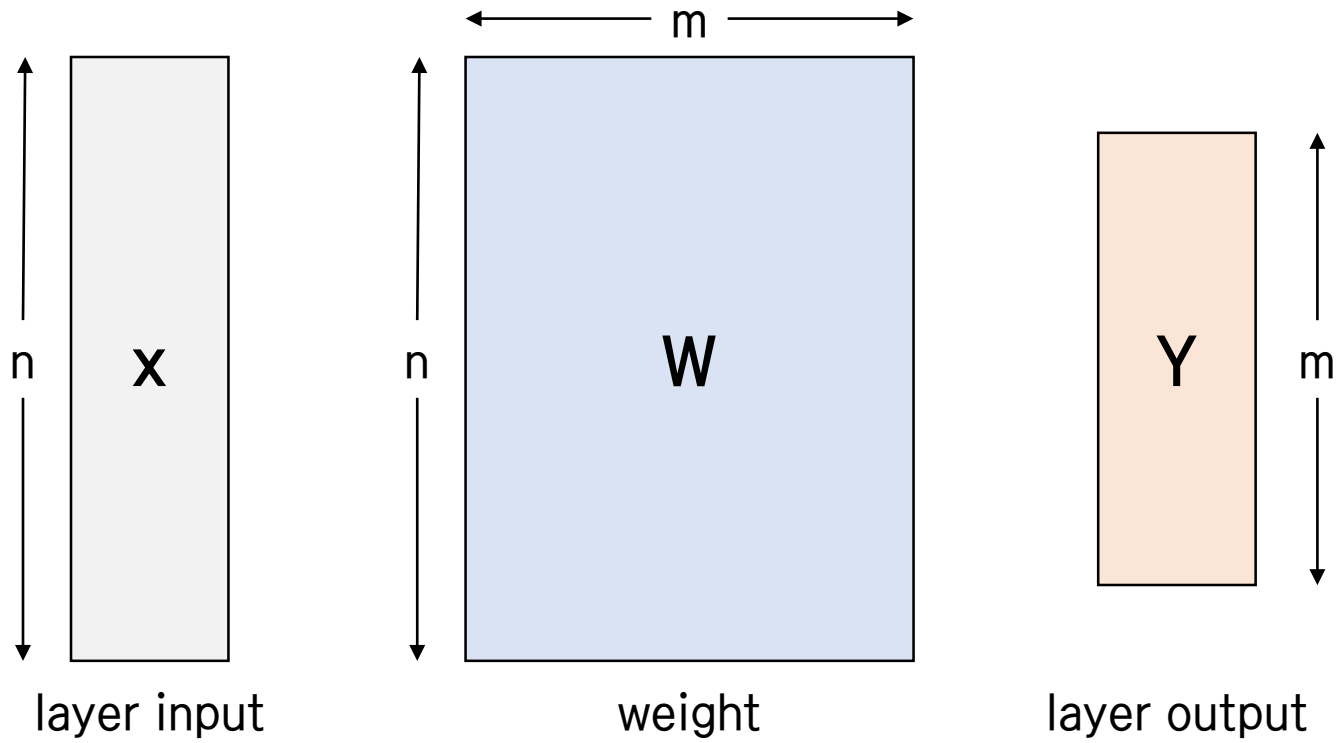
- 손실함수는 기대 값과 예측 값의 차이를 수치화해주는 함수로 오차가 클 수록 함수의 값은 커진다.
- 회귀문제에서는 MSE (Mean Squared Error), 분류문제에서는 Cross-Entropy를 주로 사용



Back Propagation

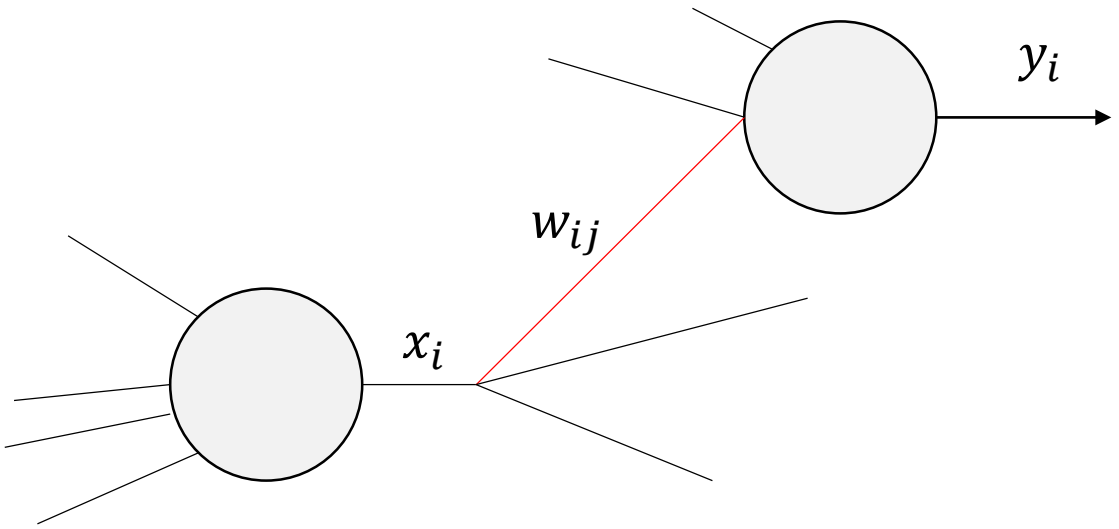
- 역전파는 출력층에서 입력층 방향으로 loss에 대한 gradient를 전파하며 계산
- N층의 노드는 N+1 층으로부터 Loss에 대한 gradient $(\frac{\partial J}{\partial y})$를 전달 받아, 자신의 가중치에 대한 gradient $(\frac{\partial J}{\partial W})$ 를 계산해 가중치를 업데이트
- N층의 노드는 가중치 업데이트 후 N-1층으로 전달할 gradient $(\frac{\partial J}{\partial x})$를 계산한 뒤 N-1층으로 전달

Back Propagation



- $x^T W = y^T$
- $y = W^T x$
- $\frac{\partial J}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial J}{\partial y} = W\frac{\partial J}{\partial y}$
- $\frac{\partial J}{\partial W} = \frac{\partial y}{\partial W}\frac{\partial J}{\partial y} = x\left[\frac{\partial J}{\partial y}\right]^T$

Back Propagation



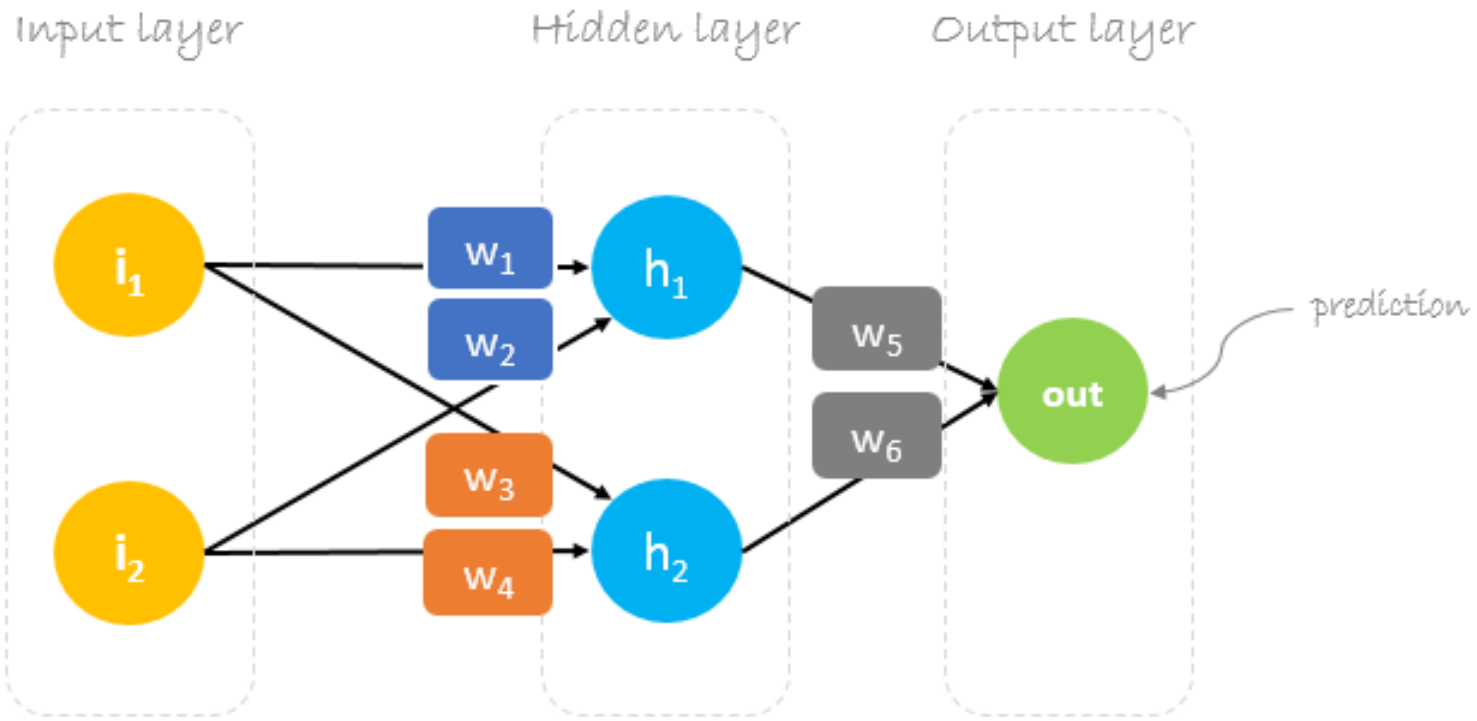
- 마지막 출력층
  - $\frac{\partial J}{\partial y} = y^{rightmost} - t$
  - $J = \frac{1}{2}\|y^{rightmost} - t\|^2$

- $\frac{\partial J}{\partial w_{ij}} = \frac{\partial y_i}{\partial w_{ij}} \frac{\partial J}{\partial y_i}$
- $y_i = x_1 w_{1j} + x_2 w_{2j} + \cdots x_n w_{nj}$
- $\frac{\partial y_i}{\partial w_{ij}} = x_i$
- $\frac{\partial J}{\partial w_{ij}} = x_i \frac{\partial J}{\partial y_i}$
- $\frac{\partial J}{\partial W} = x \left[\frac{\partial J}{\partial y}\right]^T$ 앞쪽 레이어에서 전달 받은 gradient
- $\frac{\partial J}{\partial x_i} = \frac{\partial y_i}{\partial x_i} \frac{\partial J}{\partial y_i}$
- $y_i = x_1 w_{1j} + x_2 w_{2j} + \cdots x_n w_{nj}$
- $\frac{\partial y_i}{\partial x_i} = w_{ij}$
- $\frac{\partial J}{\partial x_i} = \sum_{j=1}^{m} w_{ij} \frac{\partial J}{\partial y_i}$
- $\frac{\partial J}{\partial X} = W \frac{\partial J}{\partial y}$ 앞쪽 레이어에서 전달 받은 gradient

Summary: Back Propagation

- Nonlinear activation 함수가 없을 경우
  - 가중치 업데이트를 위한 gradient : $\frac{\partial J}{\partial W} = x\left[\frac{\partial J}{\partial y}\right]^T$
  - 이전 레이어로 전달할 gradient : $\frac{\partial J}{\partial X} = W\frac{\partial J}{\partial y}$
- Nonlinear activation 함수 (g(z))가 있을 경우
  - $f(y) = g'(z)$ z는 nonlinear activation 함수의 y에 대한 출력값
  - 가중치 업데이트를 위한 gradient : $\frac{\partial J}{\partial W} = x\left[f(y)\cdot\frac{\partial J}{\partial y}\right]^T$
  - 이전 레이어로 전달할 gradient : $\frac{\partial J}{\partial X} = \left[W\left[f(y)\cdot\frac{\partial J}{\partial y}\right]\right]$

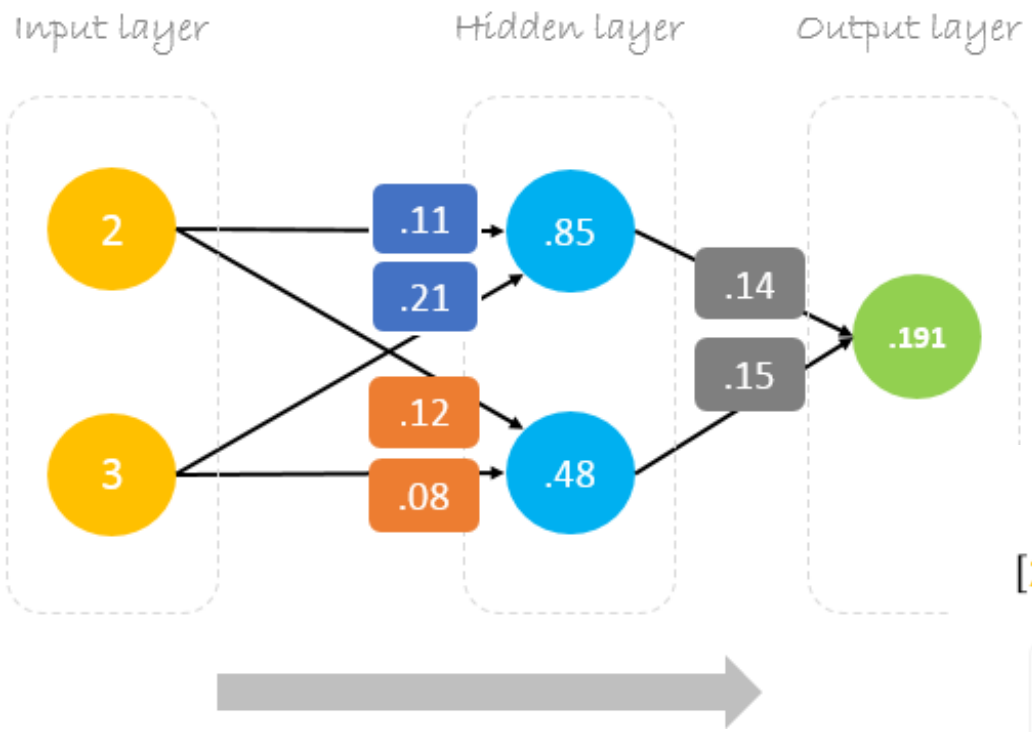
Back propagation example



- $w_1 = 0.11$
- $w_2 = 0.21$
- $w_3 = 0.12$
- $w_4 = 0.08$
- $w_5 = 0.14$
- $w_6 = 0.15$

- $x_1 = 2$
- $x_2 = 3$
- $t_1 = 1$

Forward pass



- nonlinear activation 함수가 없음으로 $y = W^T x$

Forward Pass

$$[2 \quad 3] \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = [0.85 \quad 0.48] \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = [0.191]$$

Matrix multiplication

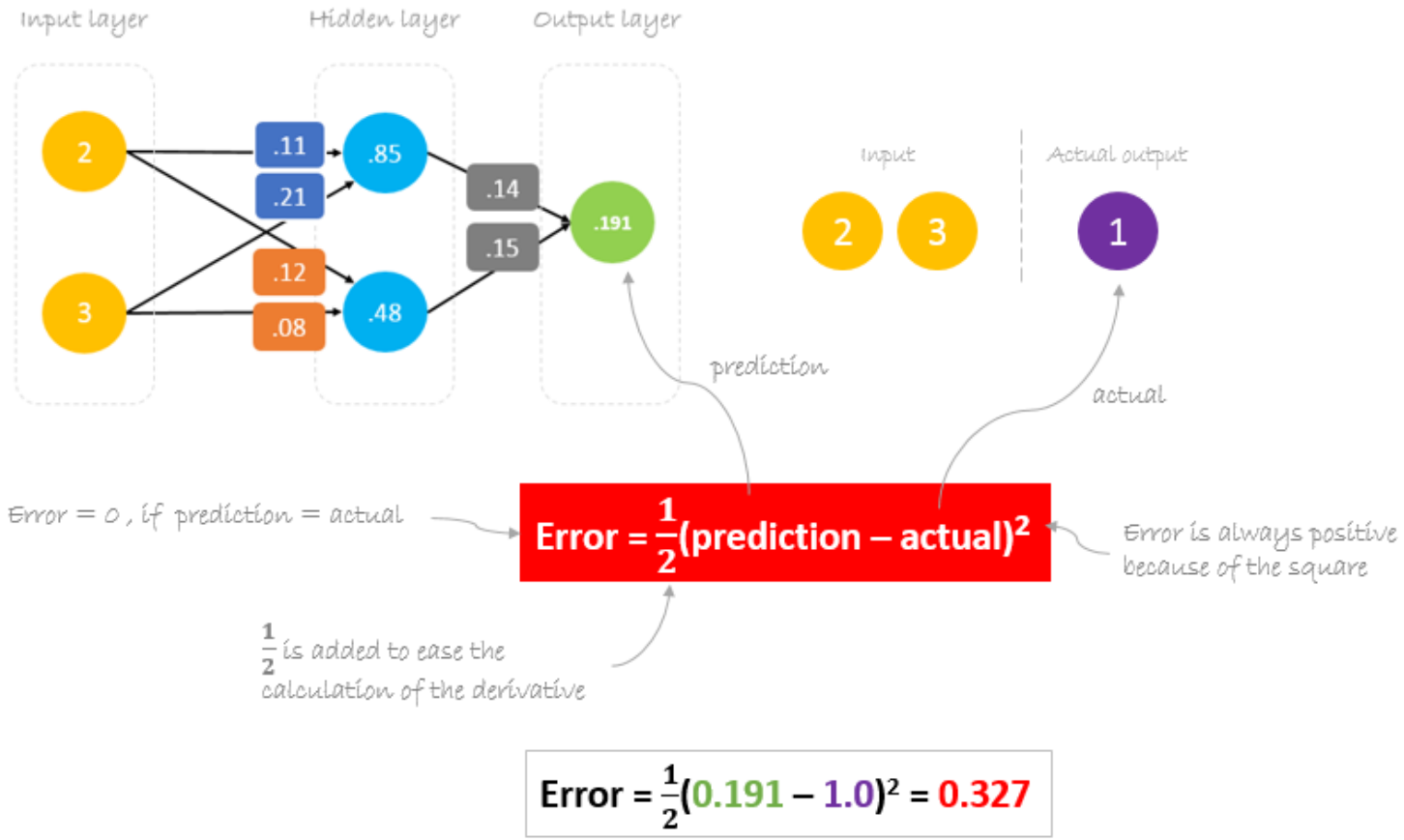
Details

2 x .11 + 3 x .21 = .85

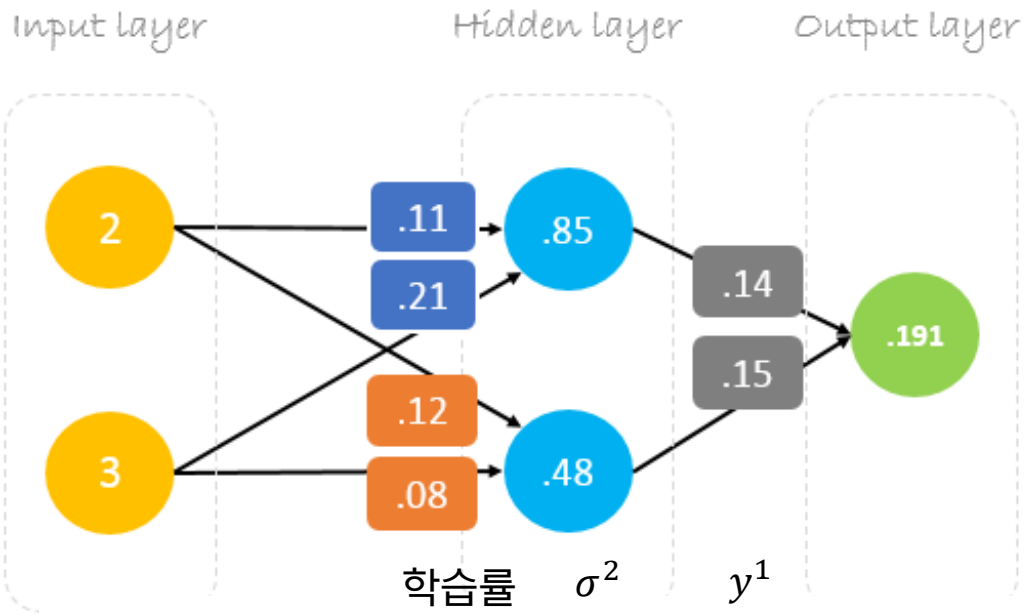
2 x .12 + 3 x .08 = .48

.85 x .14 + .48 x .15 = .191

Calculating error



Back propagation



$$\delta^2 = y^2 - t = 0.191 - 1 = -0.809$$
$$\delta^1 = W^2\delta^2 = [0.14 \quad 0.15]^T(-0.809)$$

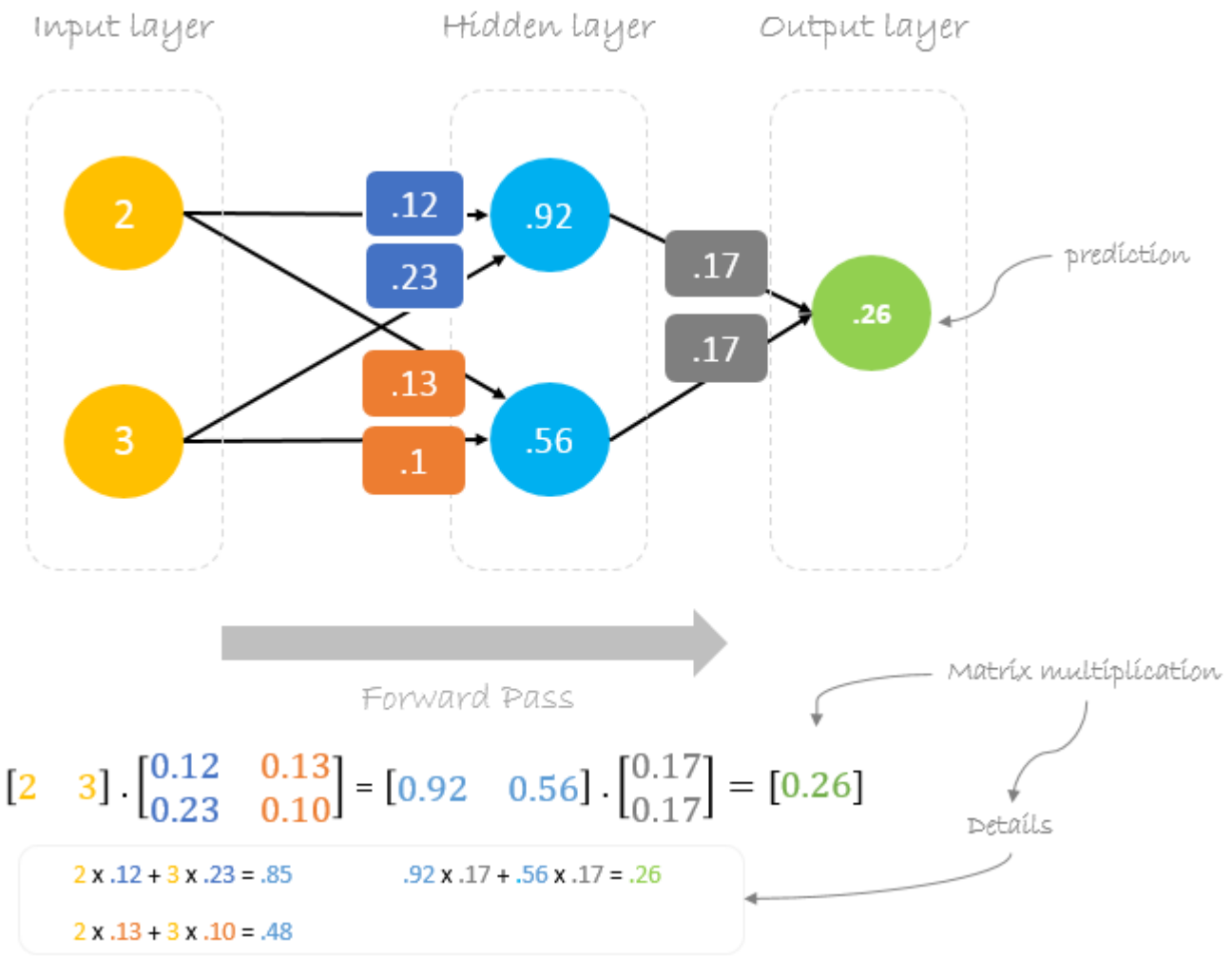
$$W^2 \leftarrow W^2 - \mu y^1 {\delta^2}^T$$
$$W^1 \leftarrow W^1 - \mu y^0 {\delta^1}^T$$

학습률 σ^2 y^1

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix}.[0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

- Forward propagation

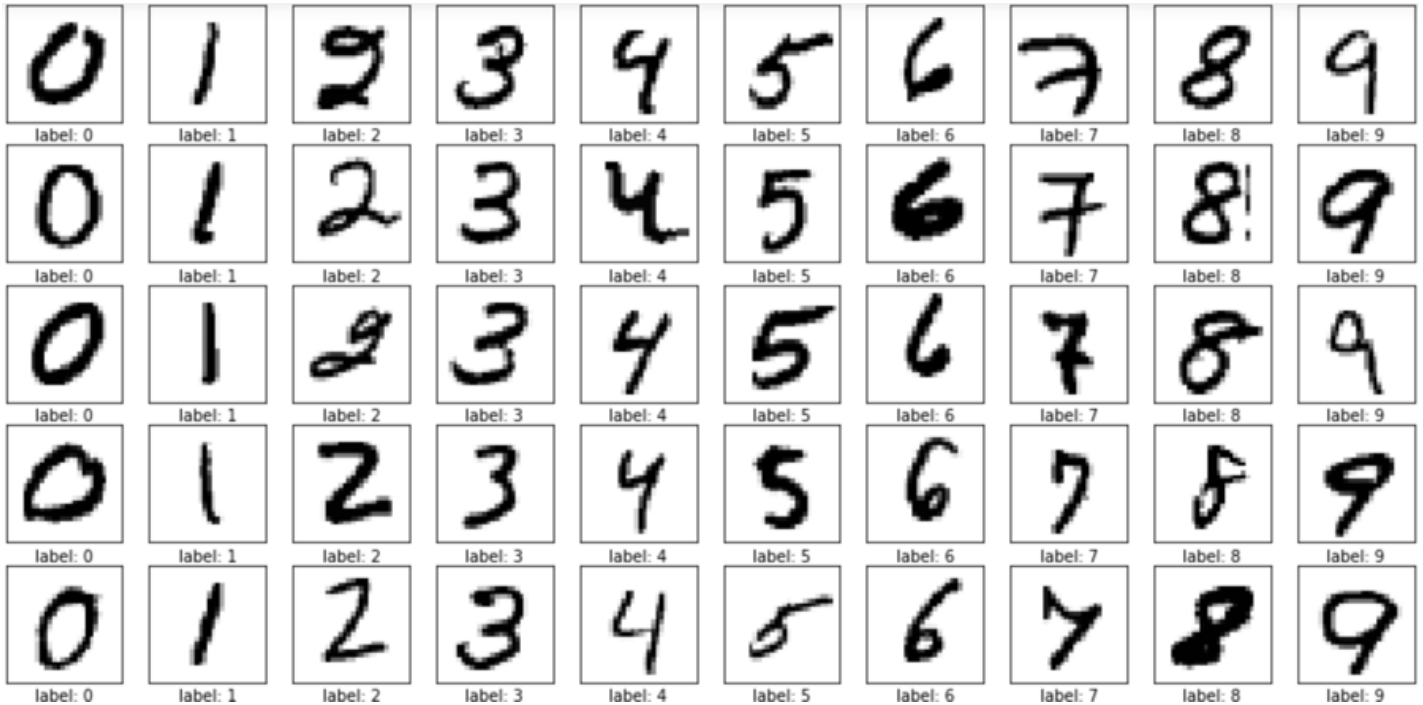


Neural Network 동작 과정 요약

1. 신경망을 구성하는 퍼셉트론들의 가중치를 초기화
2. 순전파 (Forward propagation)
   - 입력 Layer부터 출력 Layer까지 모든 조합의 경우의 수에 가중치를 대입하고 계산
   - 각 퍼셉트론의 $\sum_0^n w_i\, x_i$ 값은 활성화 함수(Activation function)에 입력된다.
   - 각 Layer를 구성하는 퍼셉트론 마다 활성화 함수가 다를 수 있다.
3. 역전파 (Back propagation)
   - 최종 출력값과 기대값을 손실 함수(Loss function)에 입력해 오차를 계산
   - 구해진 오차를 앞쪽 Layer로 전파하며 각 퍼셉트론의 가중치를 업데이트 (경사하강법)
   - 훈련 데이터에 대해 오차가 일정 수준으로 수렴할 때까지 순전파/역전파 과정 반복

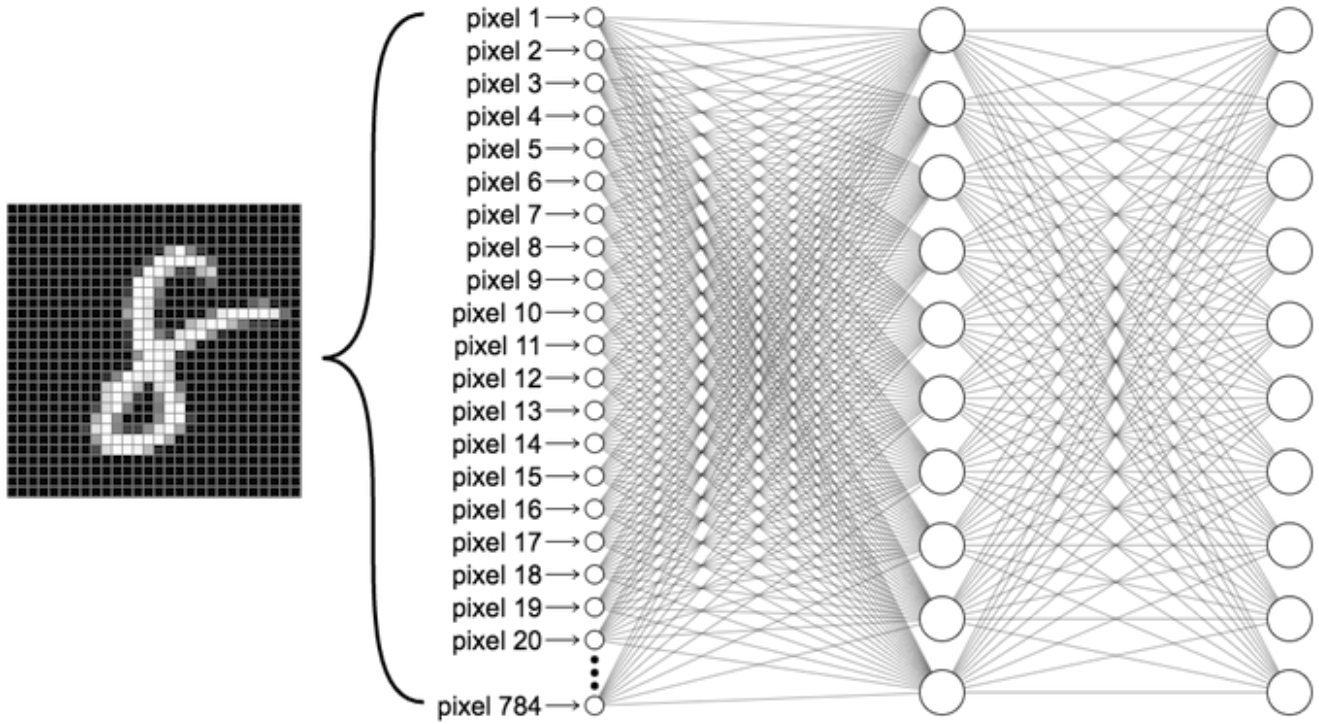
MNIST dataset 분류

- Multi layer perceptron (MLP)를 사용해 MNIST 손 글씨 데이터 분류하기 (mnist.ipynb)



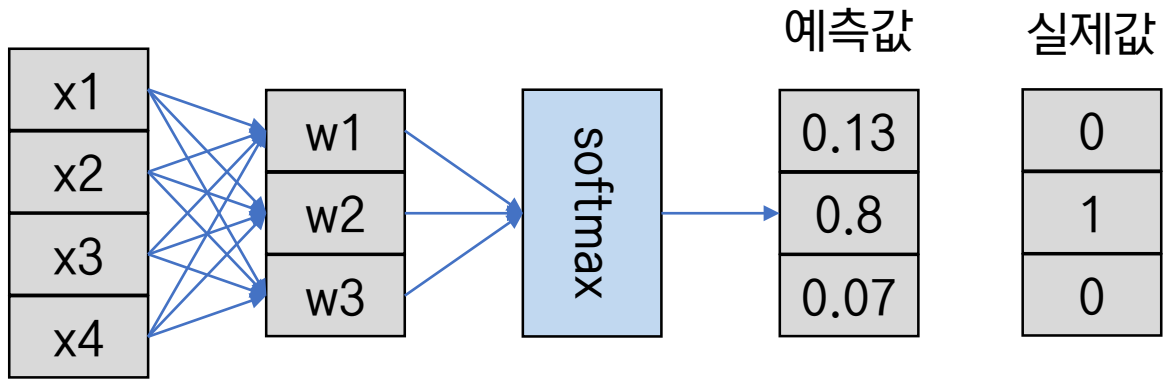
MNIST dataset 분류

- Multi layer perceptron (MLP)를 사용해 MNIST 손 글씨 데이터 분류하기 (mnist.ipynb)



Multi-class Classification

- Model의 마지막 layer의 activation function으로 softmax를 사용
- softmax layer의 노드의 수는 분류하고자 하는 class의 수와 동일
- softmax의 출력은 해당 입력이 각 클래스에 속할 확률로 0 ~ 1사이의 값을 가지며 합은 1
- 실제값은 one-hot encoding된 값



Softmax 함수

- 분류해야 하는 Class가 여러 개일 경우 Softmax 함수를 이용해 regression을 진행

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for j = 1,...,K}$$

- Softmax 함수는 K(Class의 수) 차원의 벡터를 입력 받아, 각 클래스에 대한 확률을 추정

$$softmax(z) = \left[\frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} \quad \frac{e^{z_2}}{\sum_{j=1}^{3} e^{z_j}} \quad \frac{e^{z_3}}{\sum_{j=1}^{3} e^{z_j}}\right] = [p_1, p_2, p_3] = \hat{y}$$

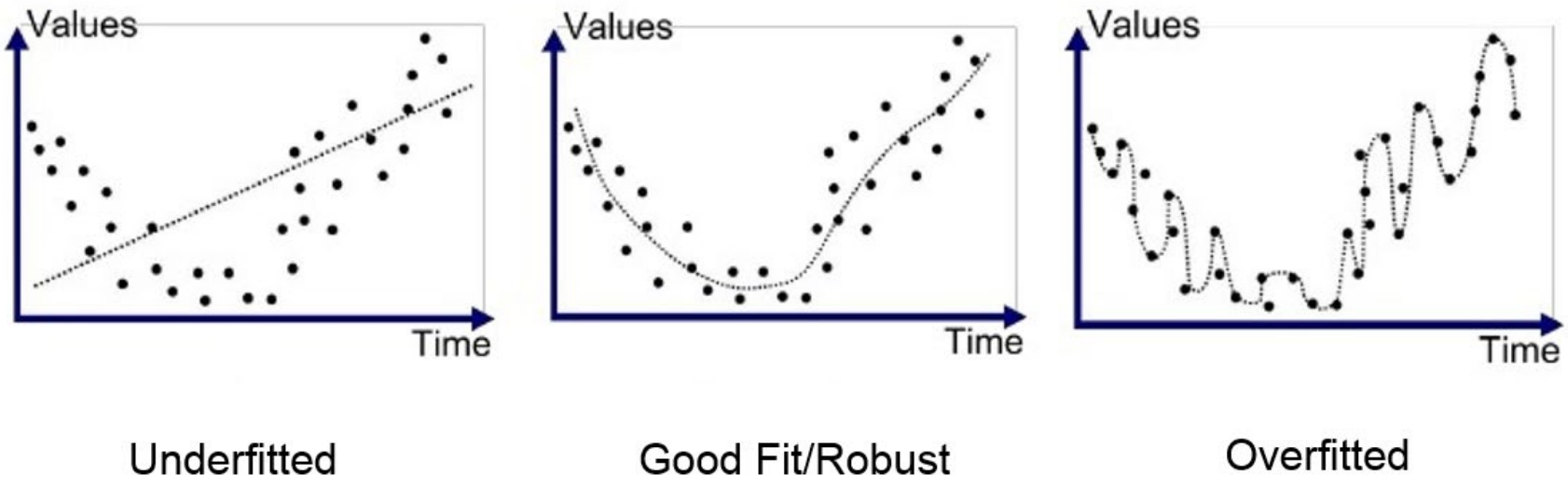
1번 클래스일 확률

Softmax with cross entropy loss

- 클래스의 수 n, softmax 함수의 i번째 입력값 (x_i), i번째 출력값 (y_i)
- $y_i = \frac{\exp(x_i)}{\sum_k \exp(x_k)}$
- 정답 벡터의 j번째 요소 $(t_i, t = [1,0,0], t_1 = 1, t_2 = 0, t_3 = 0)$, softmax 함수의 j번째 출력값 (y_j)
- $Loss = -\sum_j t_j log y_j$
- $\frac{\partial y_i}{\partial x_i} = \begin{cases} y_i(1-y_i)\,, & i \neq j \\ -y_i y_j\,, & i \neq j \end{cases}$
- $\frac{\partial L}{\partial x_i} = \sum_j \frac{\partial L}{\partial y_j}\frac{\partial y_i}{\partial x_i} = -\sum_j t_j \frac{1}{y_j}\frac{\partial y_j}{\partial x_i} = y_i - t_i$

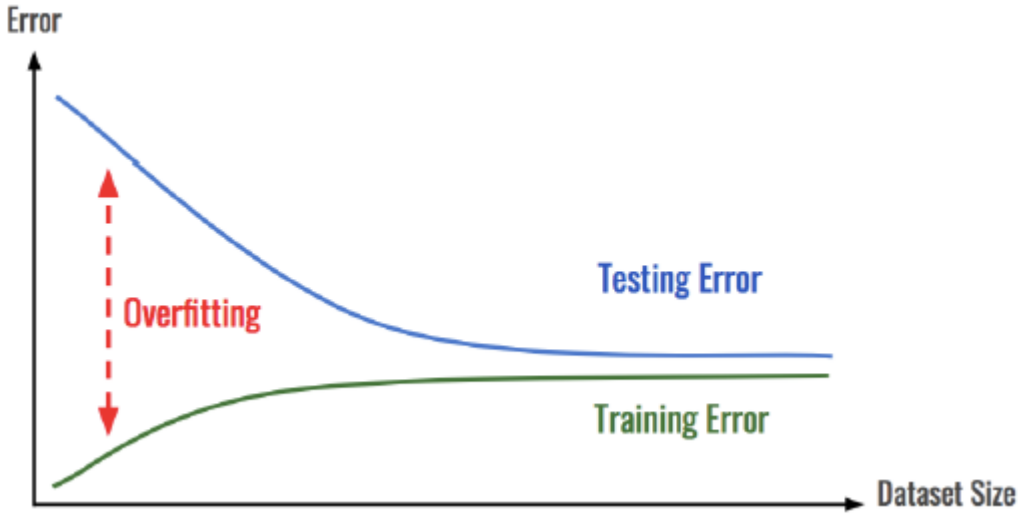
Overfitting

- 모델이 학습 데이터를 과도하게 학습하여(암기) 다른 데이터에서 제대로 동작하지 못 하는 상태
- 모델이 일반화 되어 있지 않고 훈련 데이터에서만 제대로 동작
- Training error는 낮아지는 반면, Test error는 높아진다.



Overfitting을 막기 위한 방법

- 더 많은 훈련 데이터 사용



- Data augmentation & Noise

Overfitting을 막기 위한 방법

- 모델의 복잡도 줄이기

- Early Stopping

Overfitting을 막기 위한 방법

- Regularization
  - Loss 함수에 가중치에 대한 term 추가

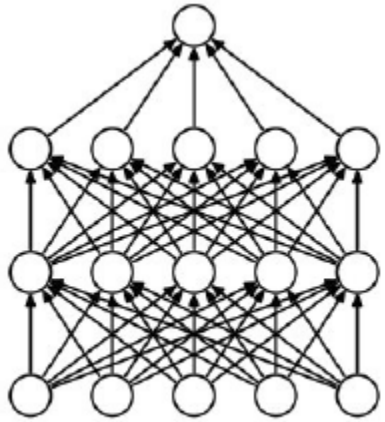
L1 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} |w_i|$$

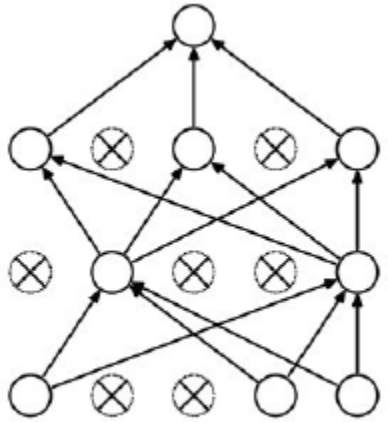
L2 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

- Dropout & Dropconnection
  - 학습 과정에서 매번 랜덤하게 node나 connection을 비 활성화



(a) Standard Neural Net



(b) After applying dropout.

famous77@kaist.ac.kr