Gradient Descent (and NN...)

경사 하강법

20200623

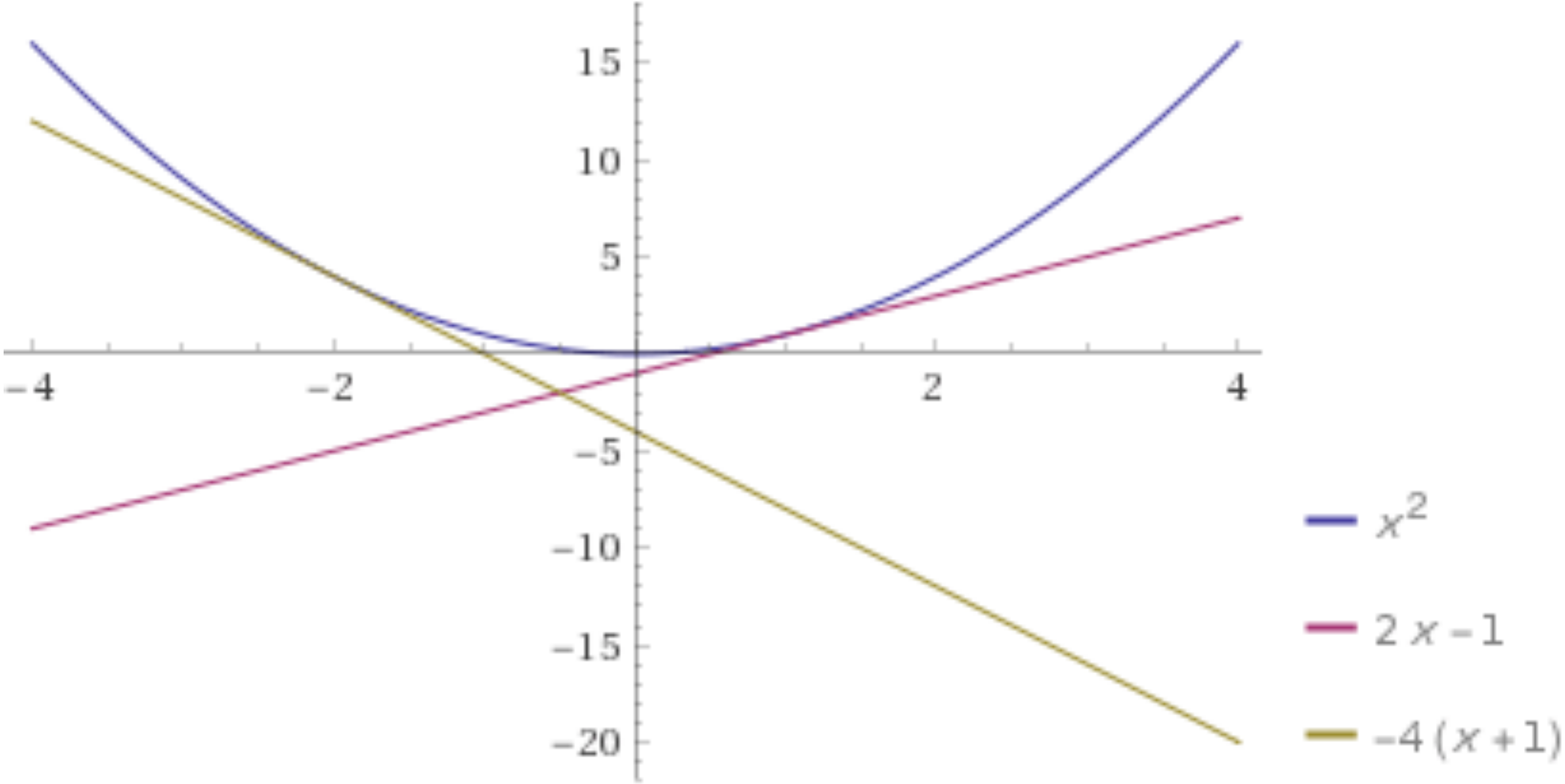
최적화 문제

- 목적함수(objective) / 손실함수(loss) / 비용함수(cost)를 최소화
- 회귀일 때: MSE, MAE 등
- 분류함수일 때?
  - crossentropy $-\sum_{x} p(x) log q(x)$

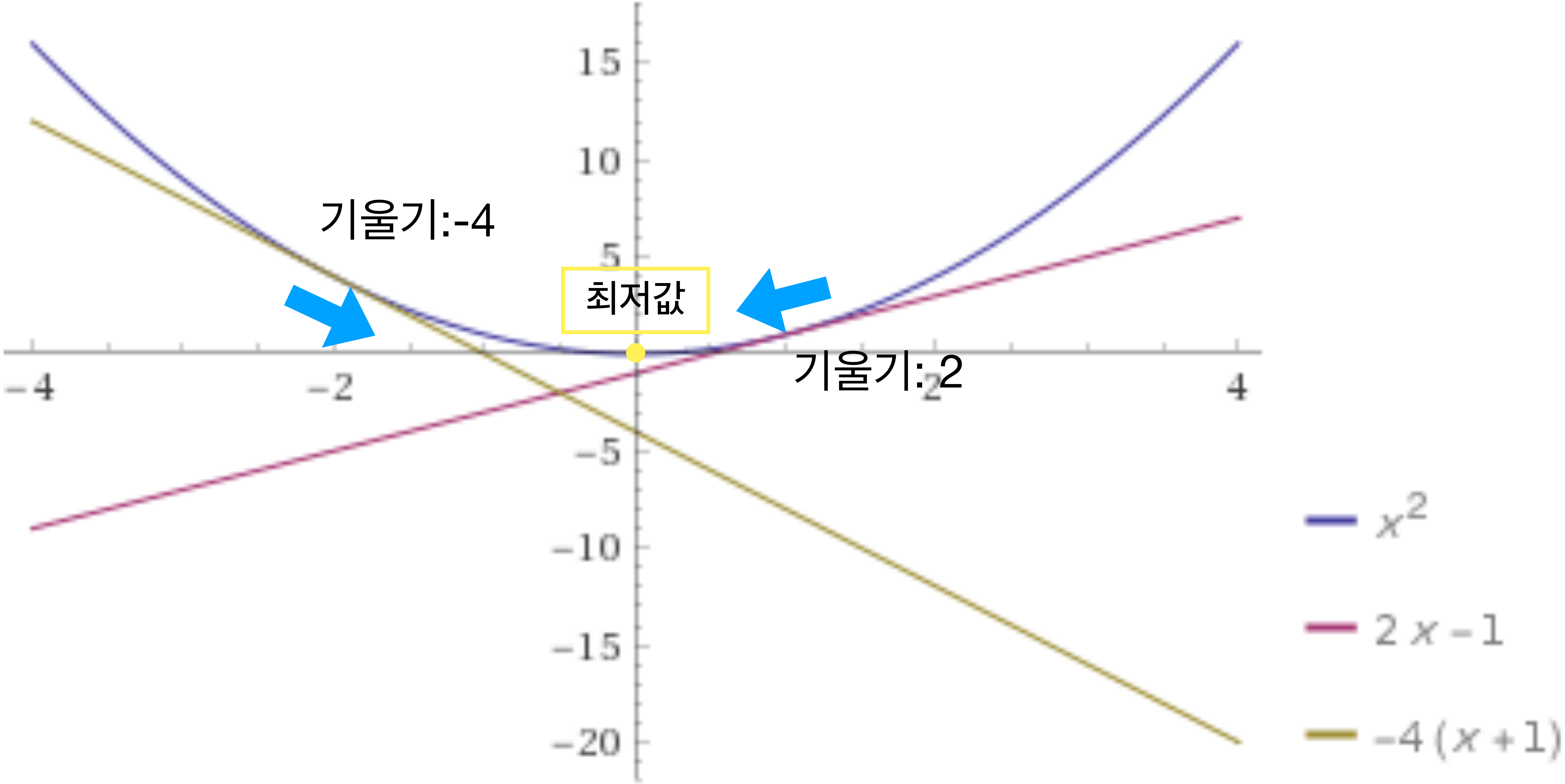
$$J(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right],$$

- 정규화
  - ridge, lasso

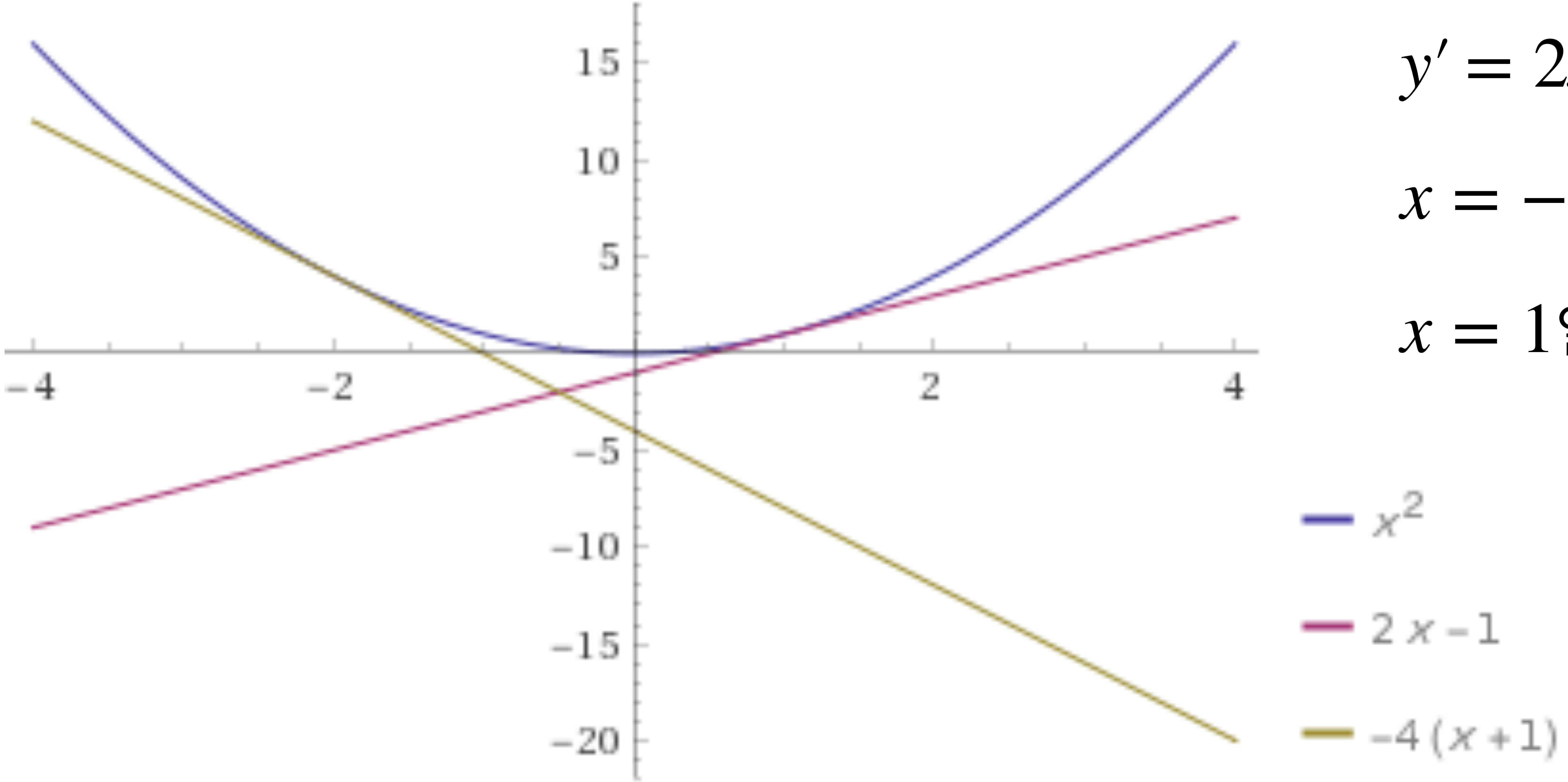
경사 하강법



경사 하강법



경사 하강법



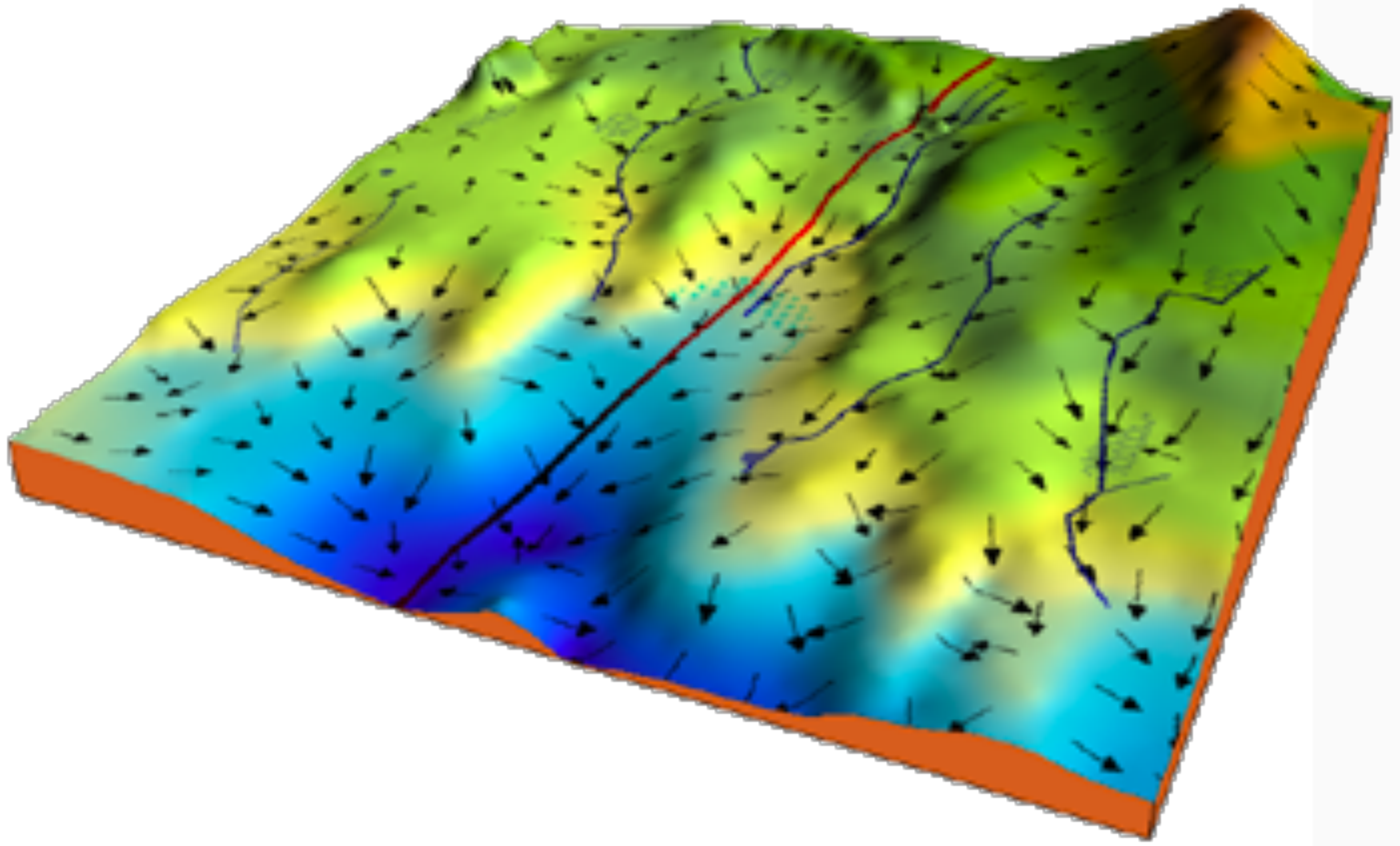
$y = x^2$

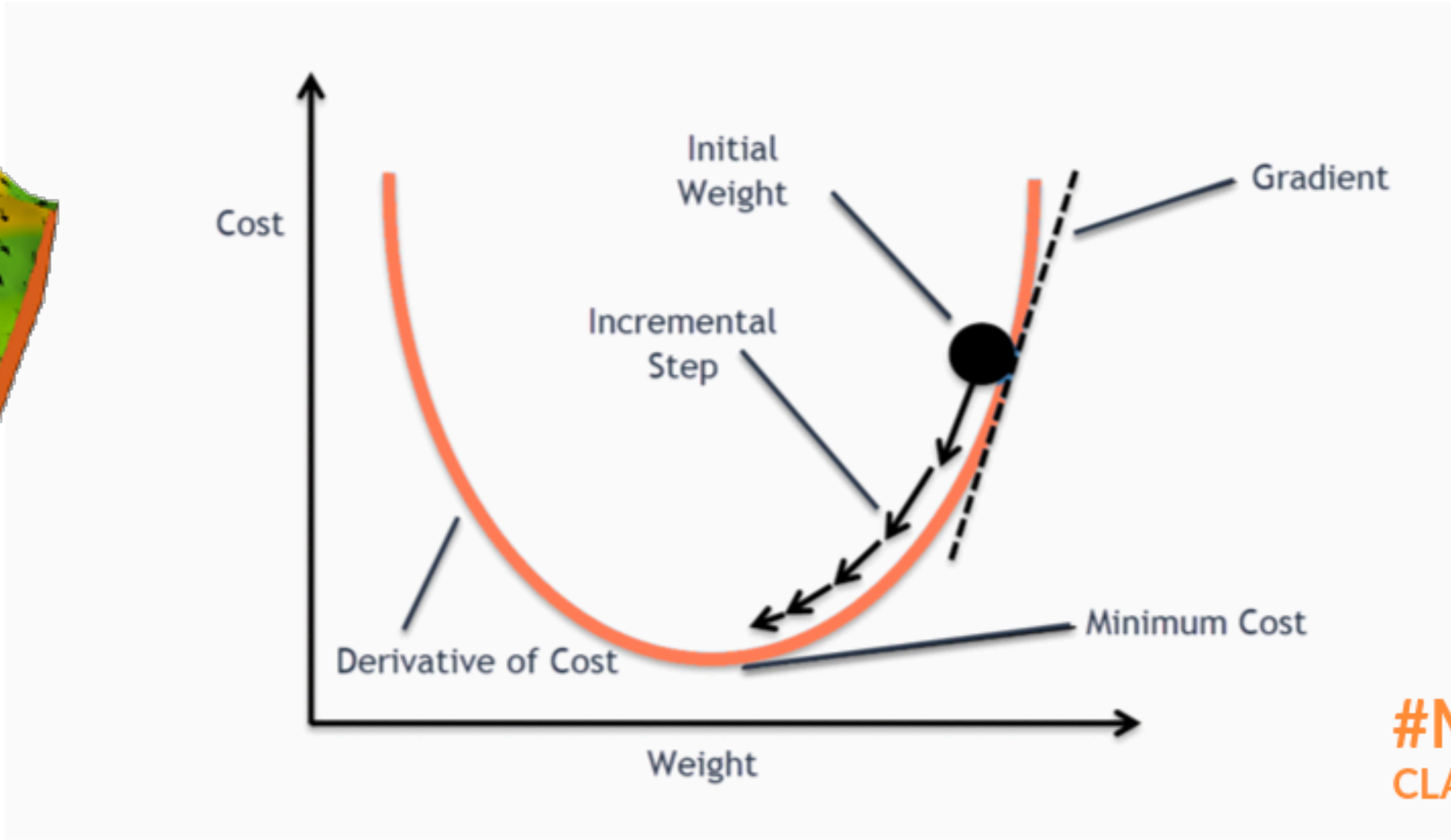
$y' = 2x$

$x = -2$일 때 $y' = -4$

$x = 1$일 때 $y' = 2$

- https://www.wolframalpha.com/input/?i=y+%3D+x%5E2%2C+y%3D2x-1%2C+y%3D-4x-4%2C+where+-4%3Cx%3C4




Cost
Initial Weight
Gradient
Incremental Step
Derivative of Cost
Minimum Cost
Weight

경사 하강법

- 랜덤한 위치에서 시작하고,
- 현재 위치에서 미분값의 반대 방향으로 이동
- 반복하면 최저값 - 또는 부분 최저값(local minimum)에 도달
- $x_{i+1} = x_i - \lambda f'(x_i)$
- 우리는 모델 패러미터 W에 대해 목적함수 J를 최적화하려고 하므로
- $W_{i+1} = W_i - \lambda J'(W_i)$ 또는 $W_{i+1} = W_i - \lambda \dfrac{\partial J(W_i)}{\partial W}$
- 보통 실제적으로는 수치적으로 경사도를 구함 $\dfrac{f(x+h) - f(x)}{h}$

Linear Regression

Cost Function

$$J\left(\Theta_0, \Theta_1\right) = \frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y_i]^2$$

Predicted Value ($h_\Theta(x_i)$), True Value (y_i)

Gradient Descent

$$\boxed{\Theta_j = \Theta_j - \alpha\frac{\partial}{\partial\Theta_j}J\left(\Theta_0, \Theta_1\right)}$$

Learning Rate (α)

Now,

$$\frac{\partial}{\partial\Theta}J_\Theta = \frac{\partial}{\partial\Theta}\frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y]^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x_i) - y)\frac{\partial}{\partial\Theta_j}(\Theta x_i - y)$$

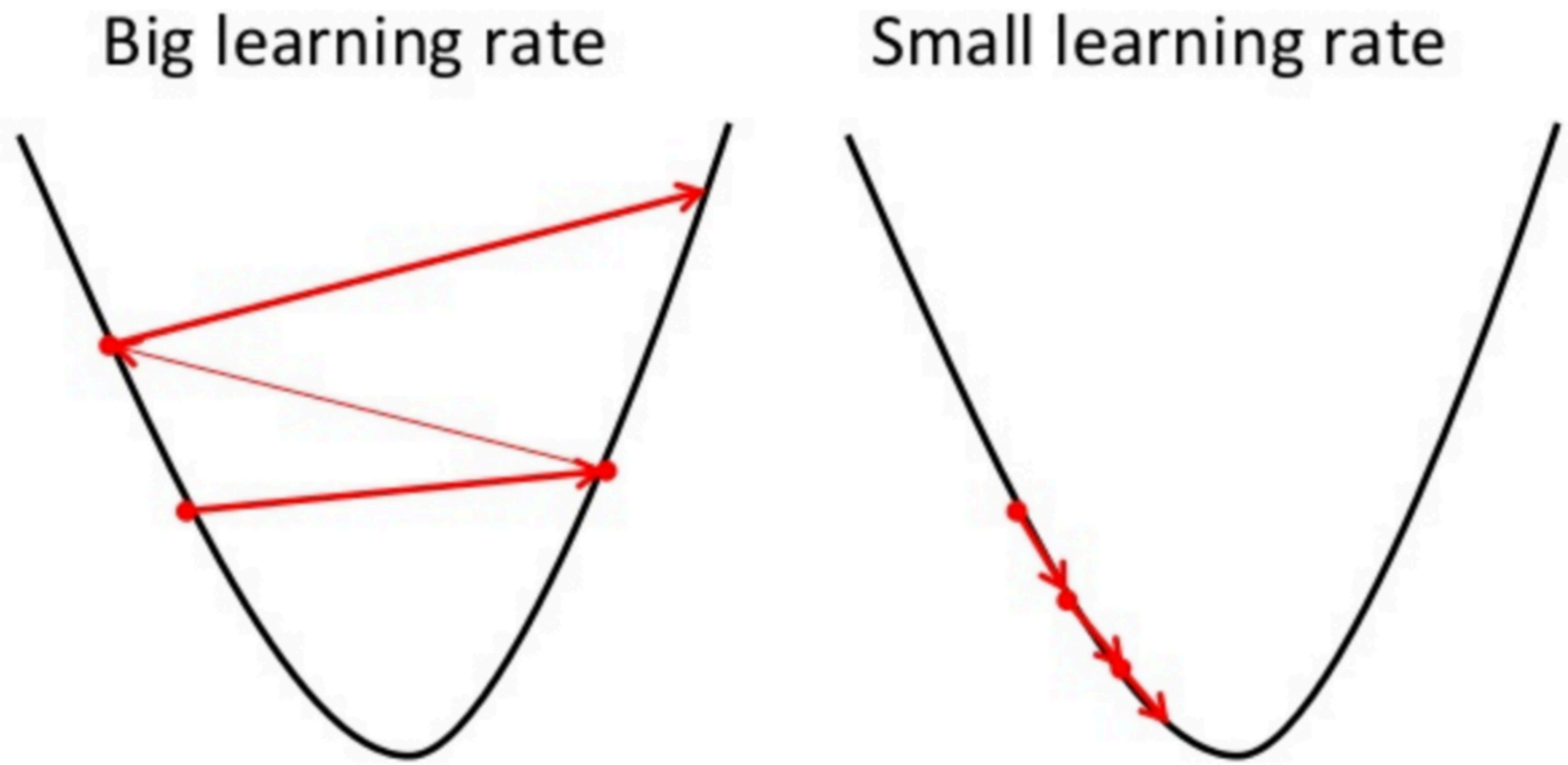
$$= \frac{1}{m}(h_\Theta(x_i) - y)x_i$$

Therefore,

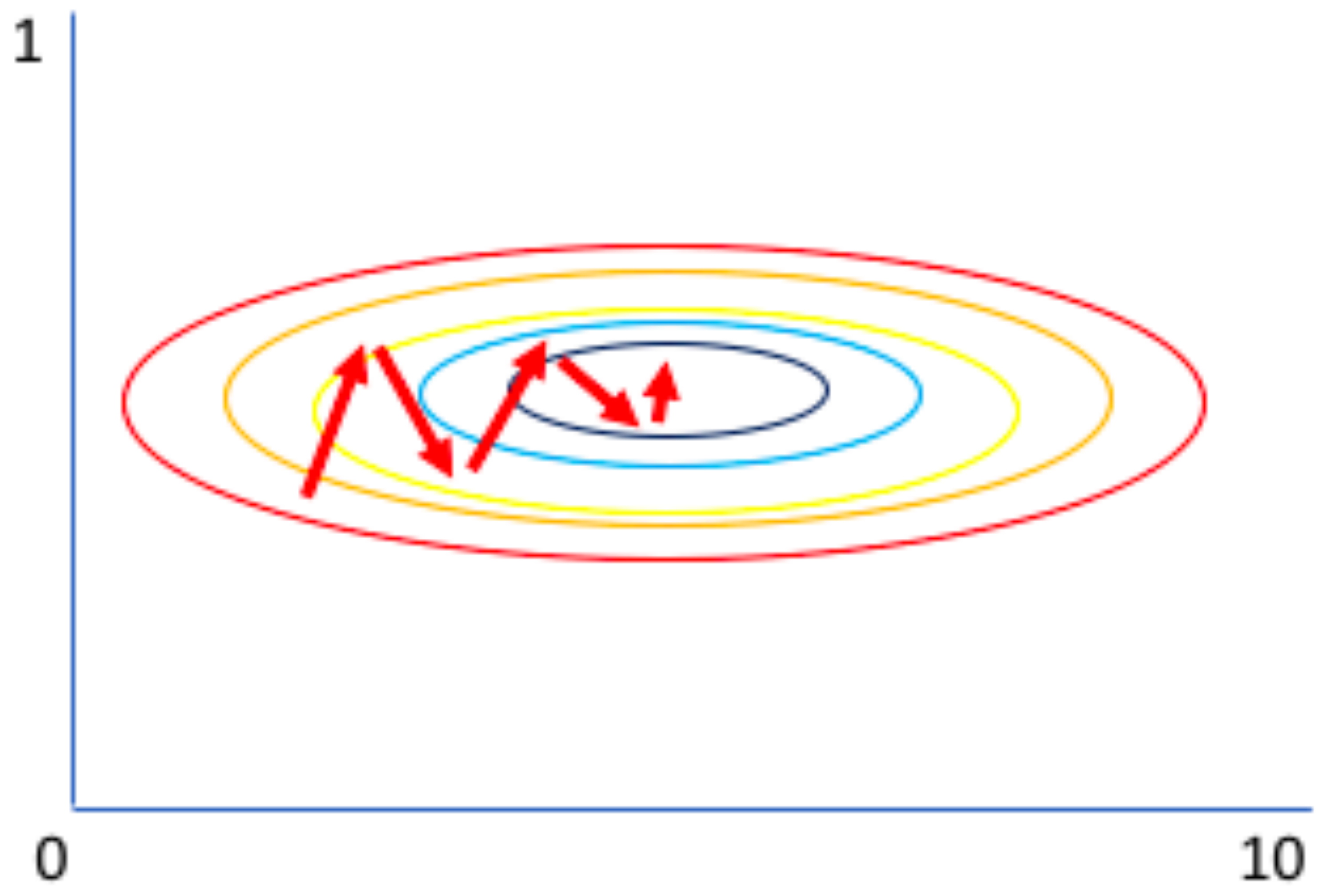
$$\boxed{\Theta_j := \Theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}[(h_\Theta(x_i) - y)x_i]}$$

Learning Rate λ

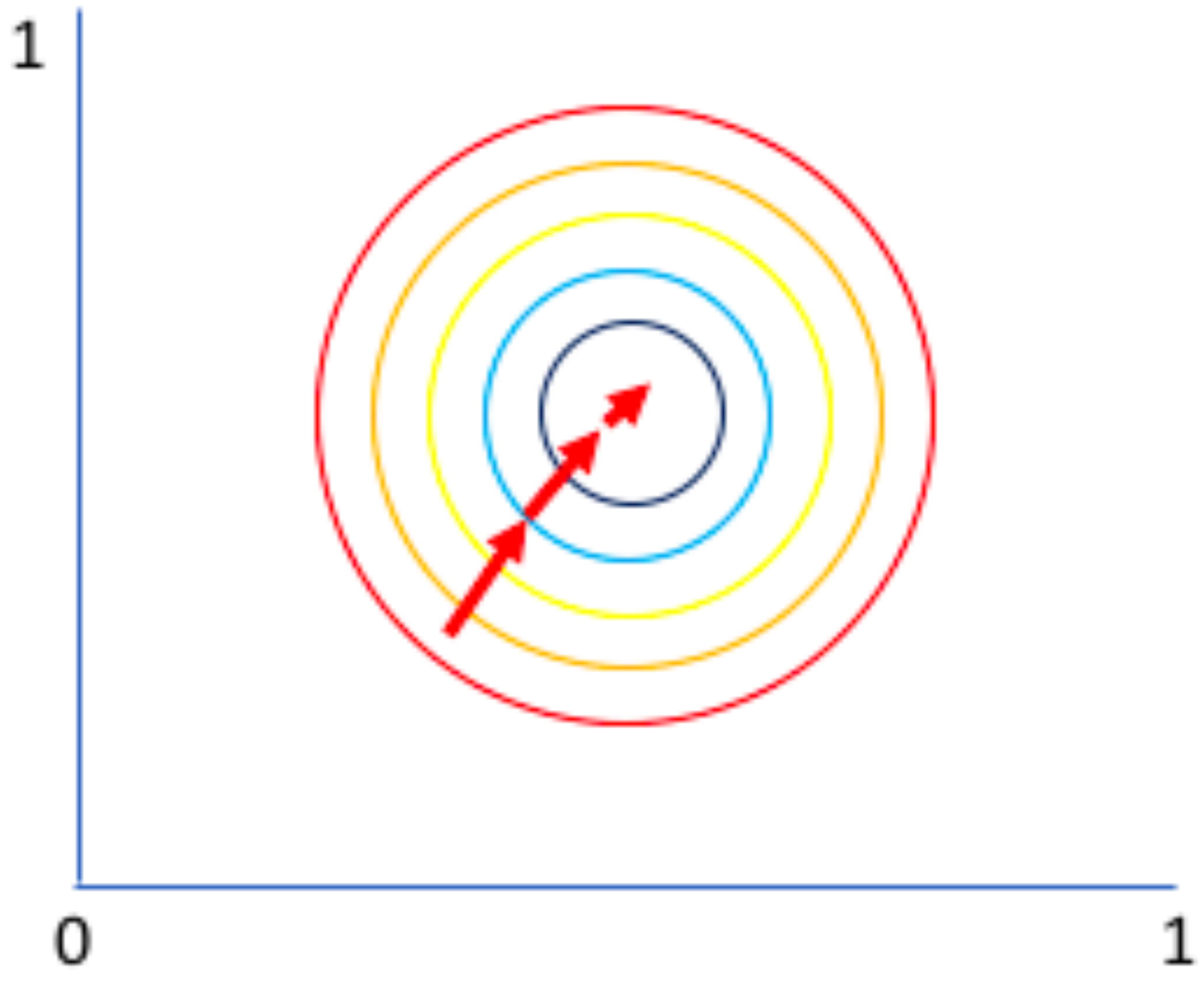
- Learning rate가 너무 크면 최소값으로 진행이 안 될 수도 있고,
- 너무 작으면 너무 느리게 학습될 수 있음
- 학습 진행에 따라 learning rate을 조정



Why normalize?



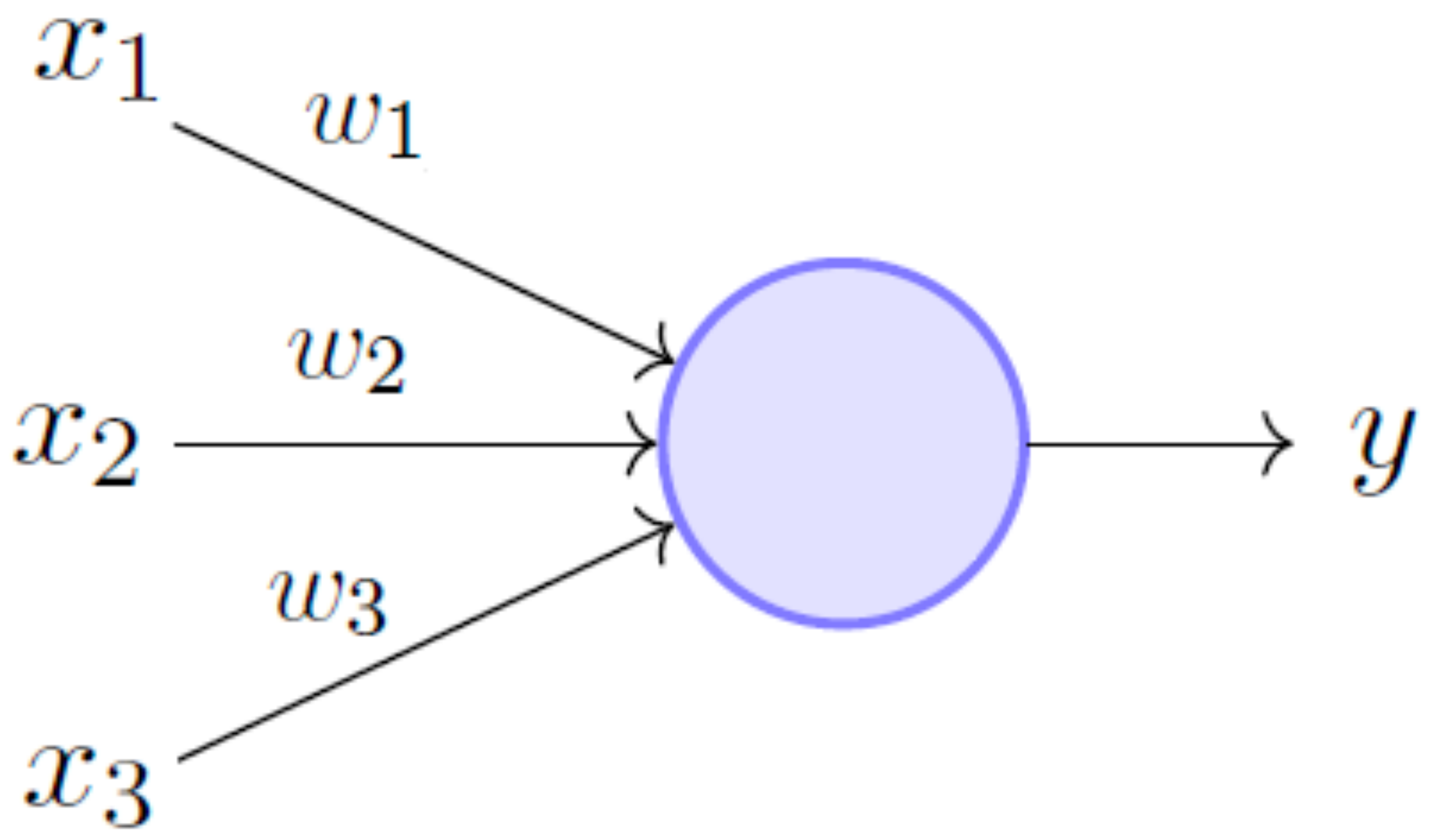
Gradient of larger parameter
dominates the update



Both parameters can be
updated in equal proportions

Perceptron!

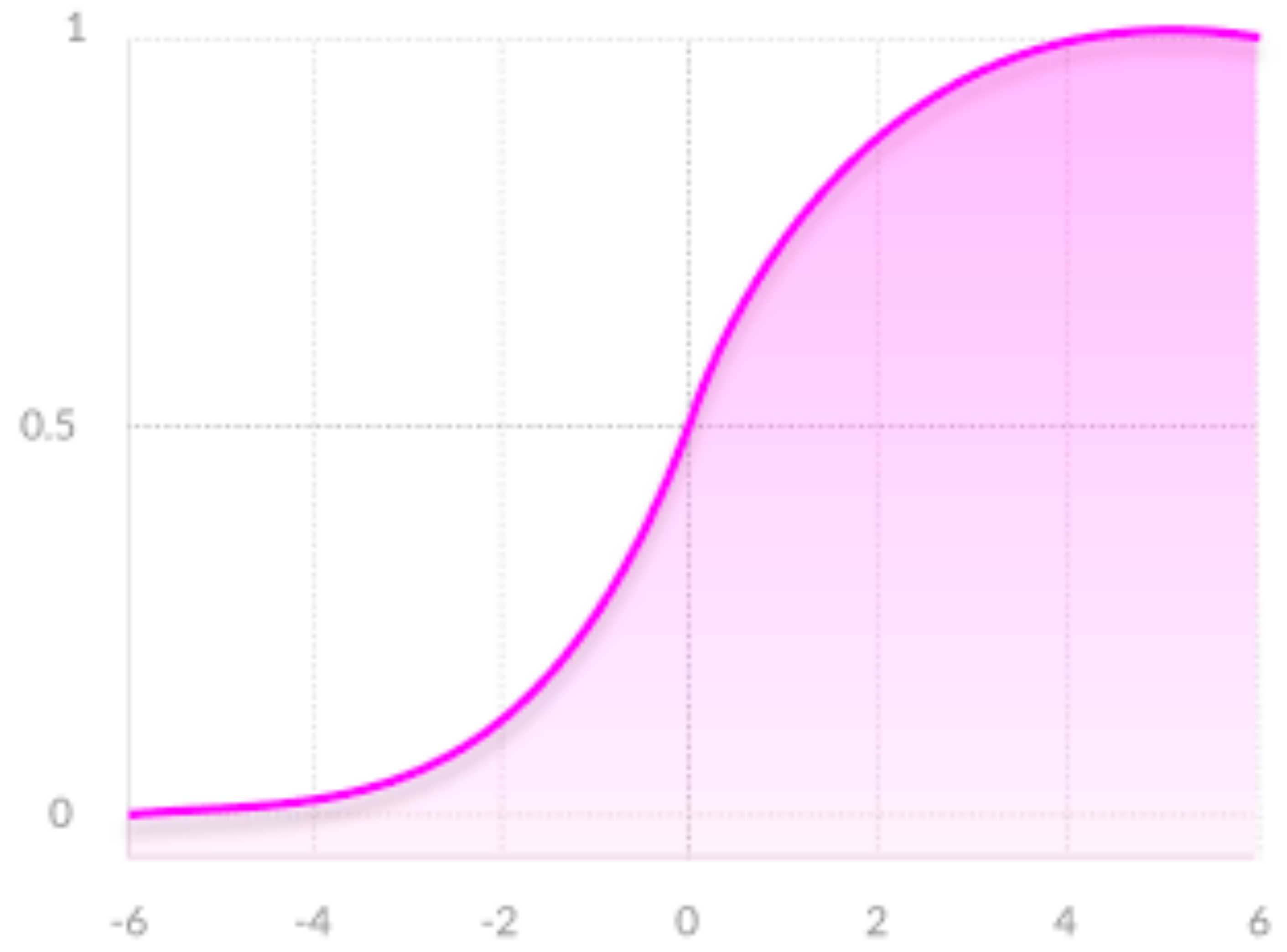
- 선형 회귀 모델

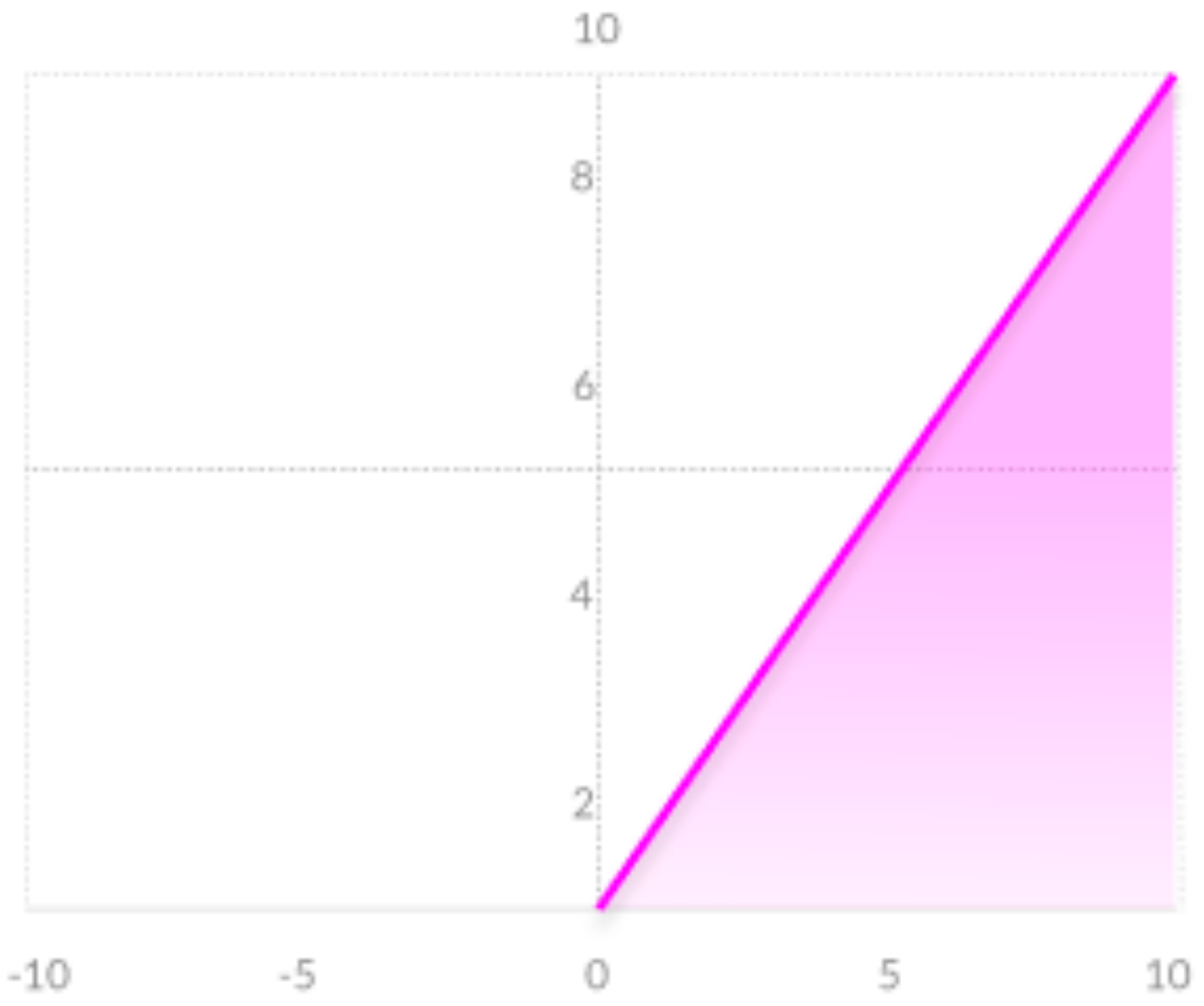


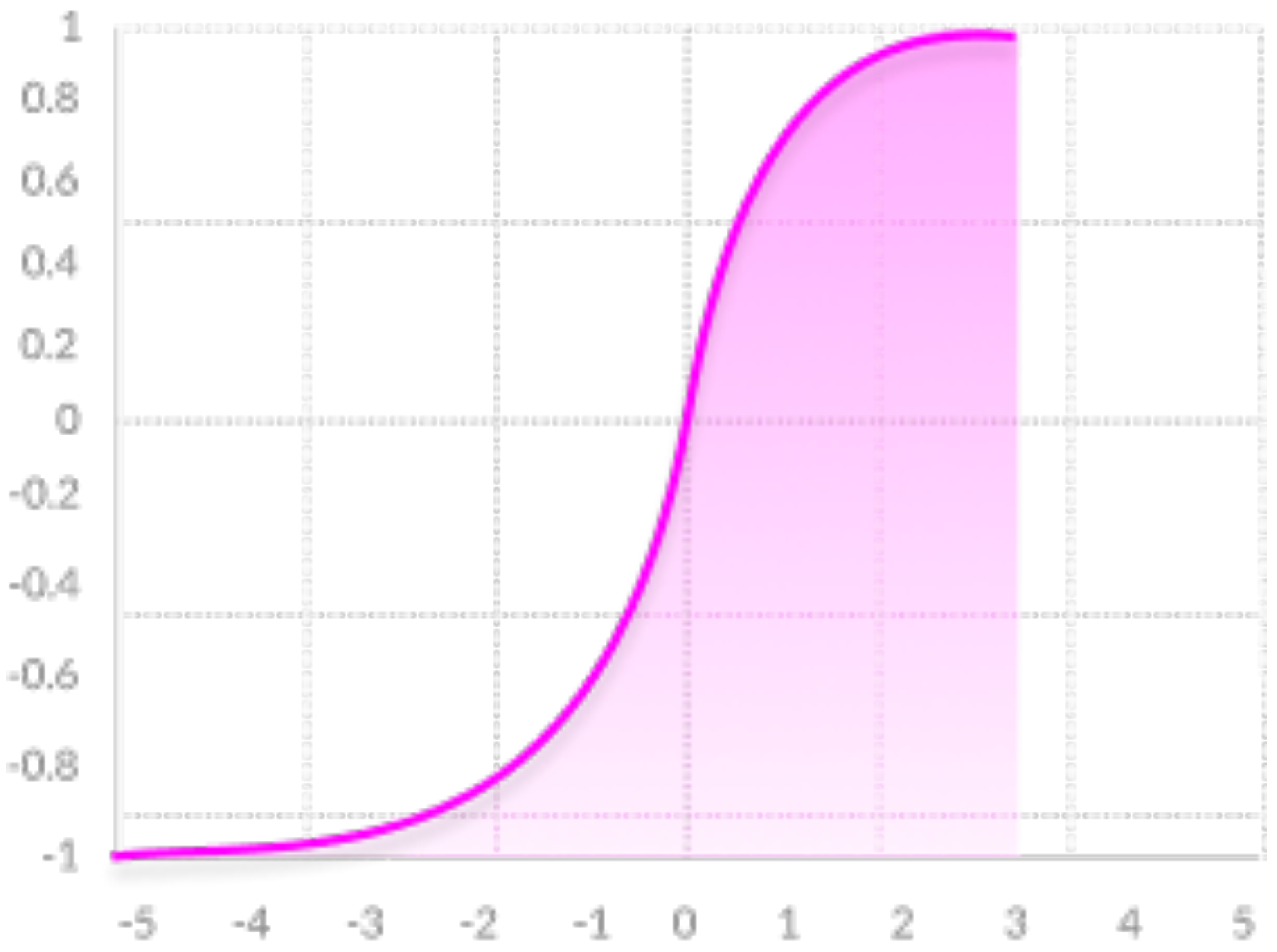
Perceptron Model (Minsky-Papert in 1969)

... + Activation function

- 퍼셉트론에 비선형성을 추가해주는 함수
- Logistic (Sigmoid)
- TanH: Logistic의 -1 ~ 1 버전
- ReLU (rectified linear unit)
- Softmax (classification)





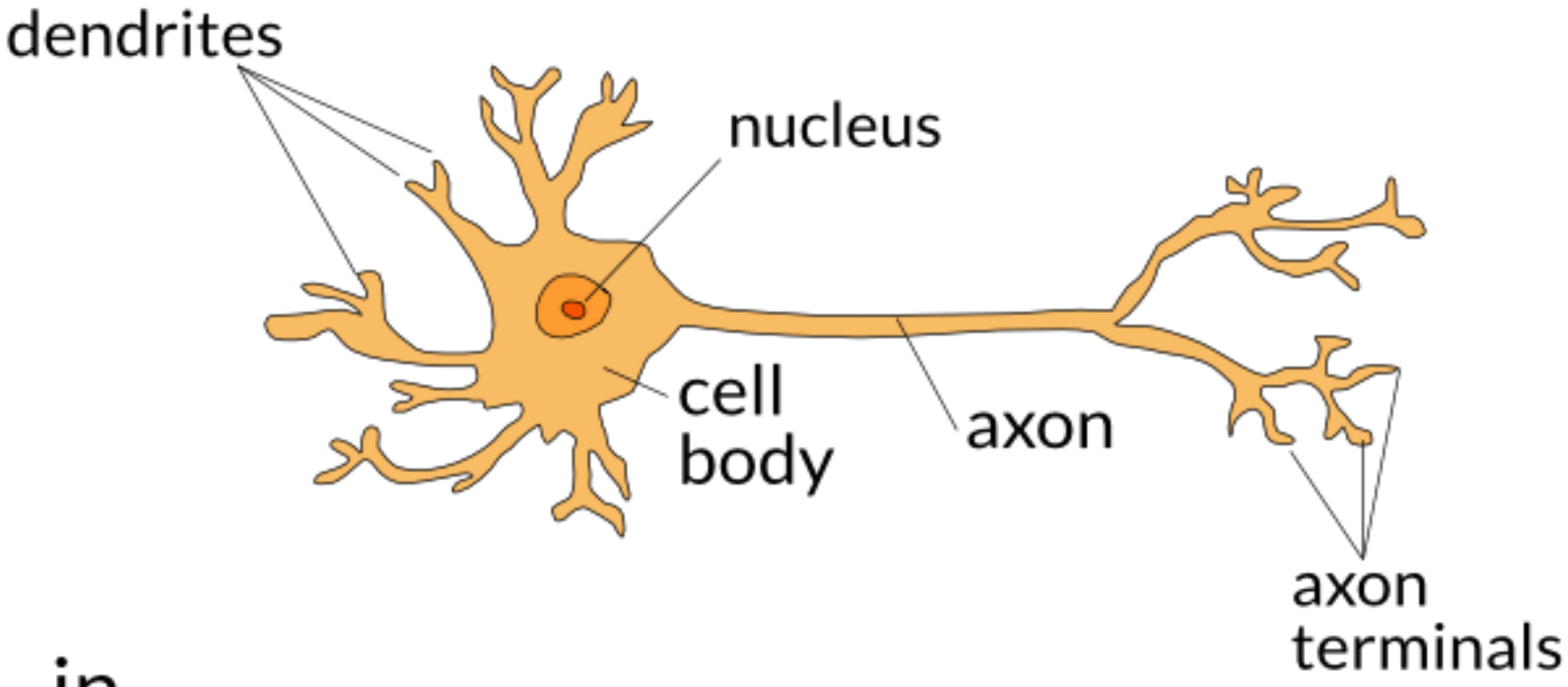


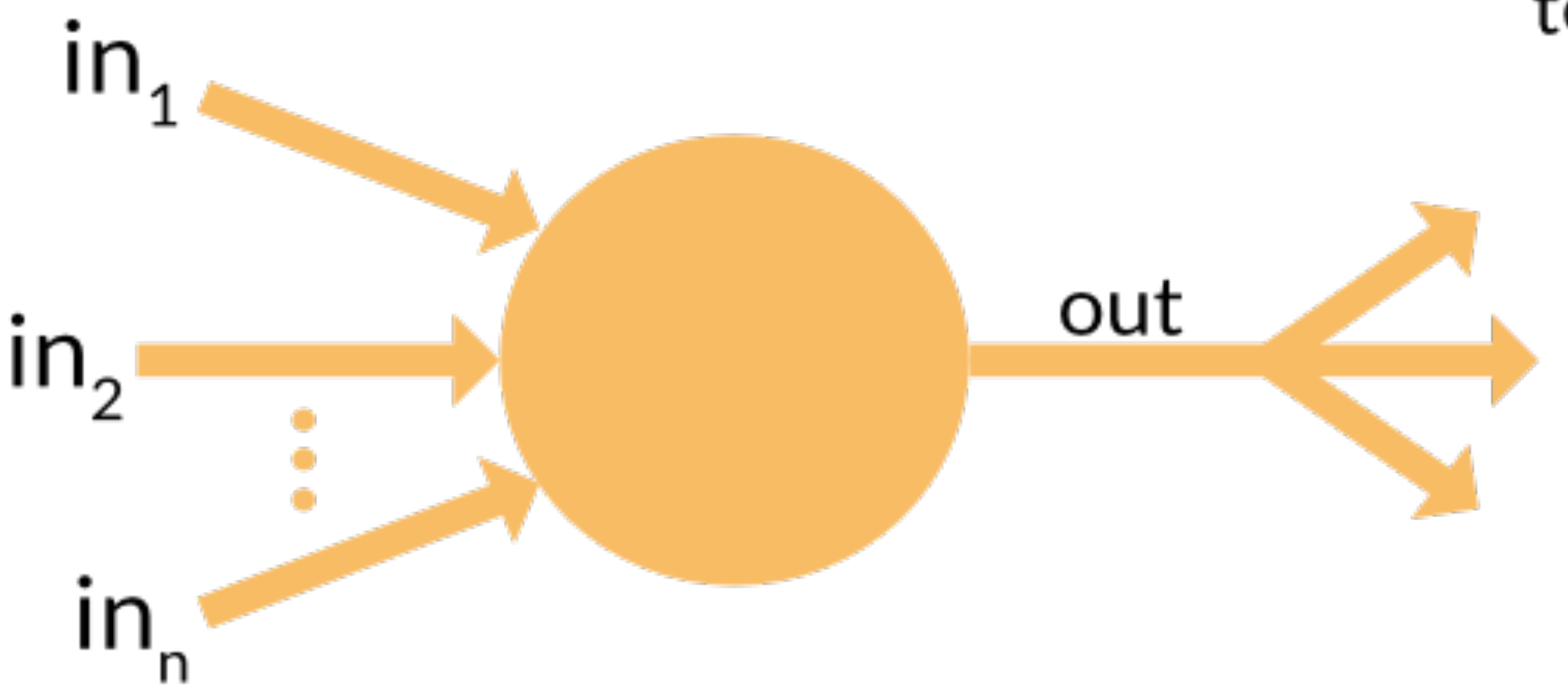
Softmax function

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

where $\theta = w_0x_0 + w_1x_1 + \ldots + w_kx_k = \sum_{i=0}^{k} w_ix_i = w^Tx$

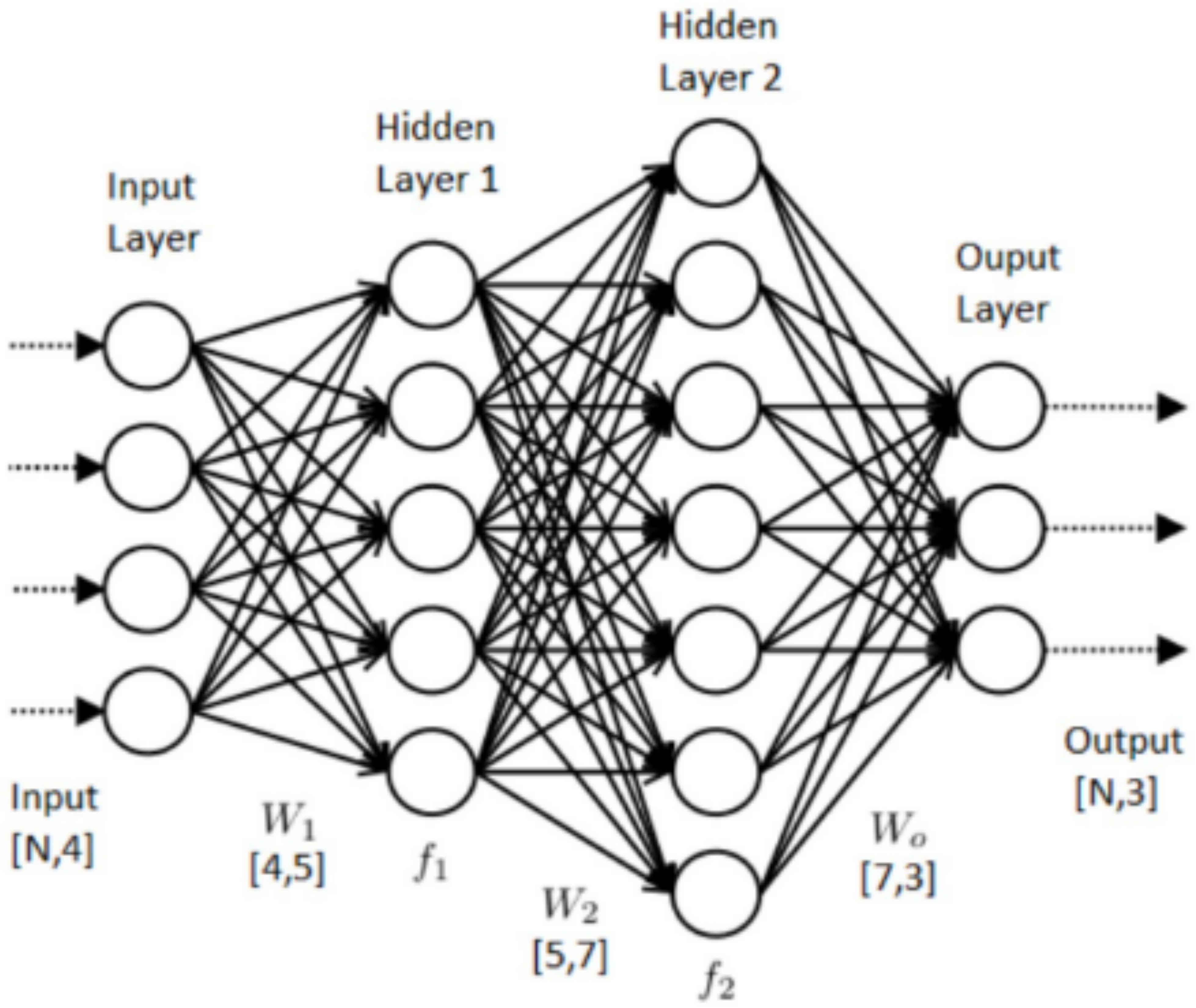
Neuron과 Perceptron



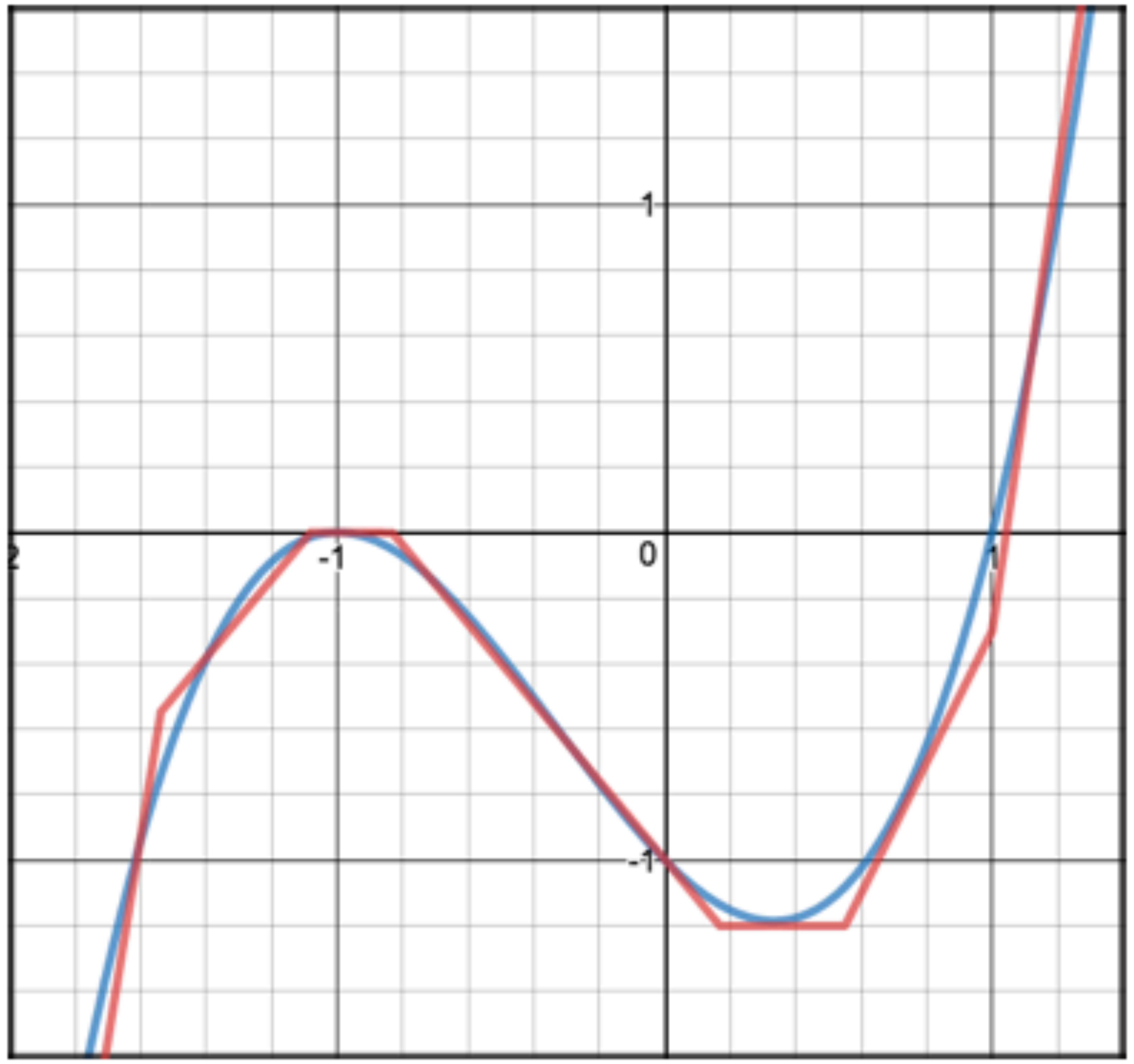


Neural Network

- Multilayer Perceptron (MLP)
  - Perceptron(+activation)을 여러 층 쌓아서 함수의 복잡성을 높임
  - 인접한 layer간의 모든 node가 연결됨
- Artificial Neural Network (ANN, NN)
  - MLP를 포함해 neuron 다수가 연결된 구조
- Deep Neural Network (DNN)
  - Layer가 많은 구조의 NN
  - 보통 3층 이상



Universal Function Approximation Theorem

- 1개의 히든 레이어를 가진 NN은 모든 함수를 근사할 수 있다!
  - 하지만 뉴런 개수가 무한히 필요할 수도 있음
- 

$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$