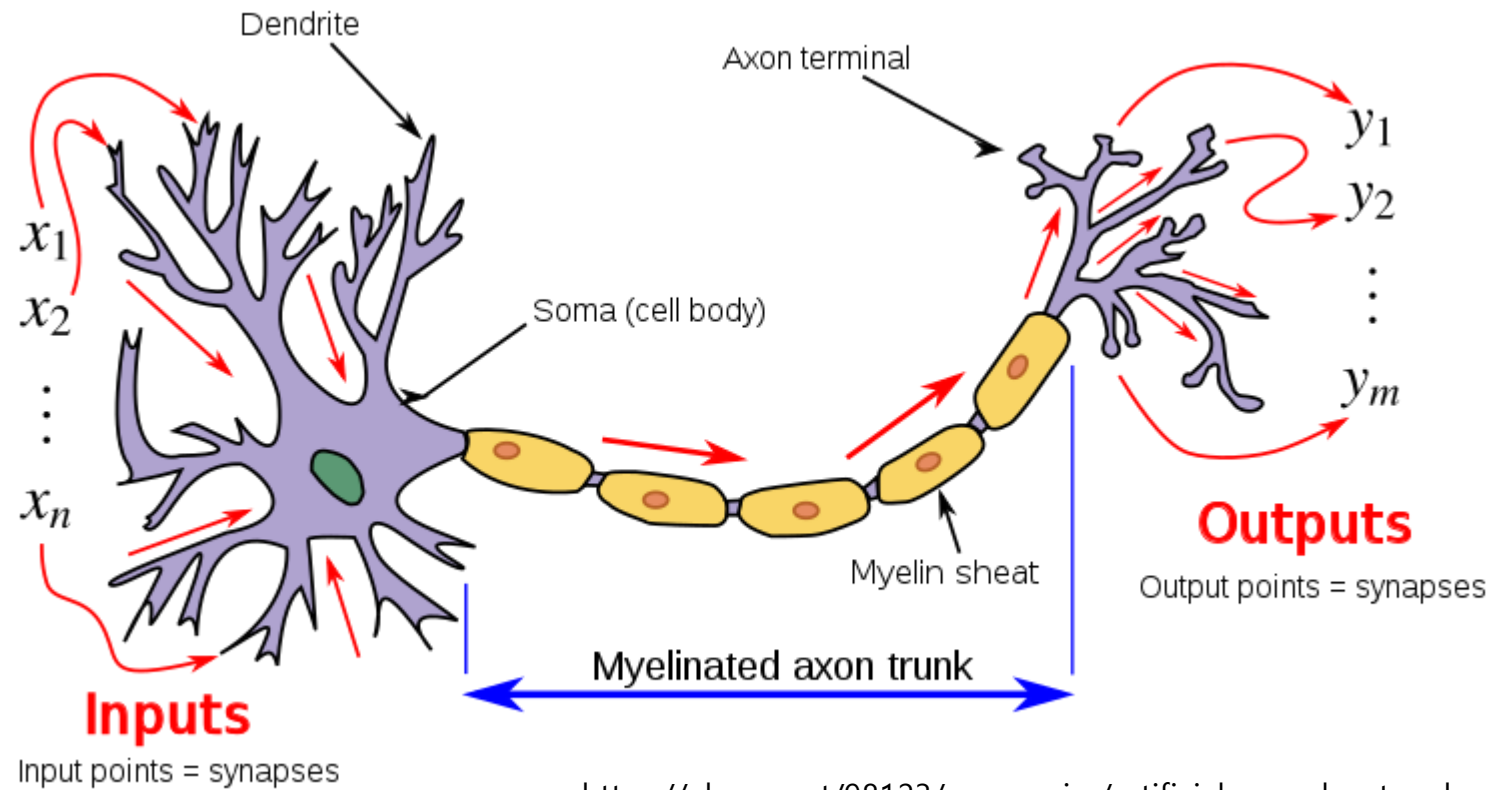




Neural Network

Neural Network



https://ebrary.net/98123/economics/artificial_neural_networks

참고자료

- 밑바닥부터 시작하는 딥러닝(사이토 고키, 2017)
- CS231n(<http://cs231n.stanford.edu/>)
- <https://github.com/Harry24k>

NEURAL NETWORK

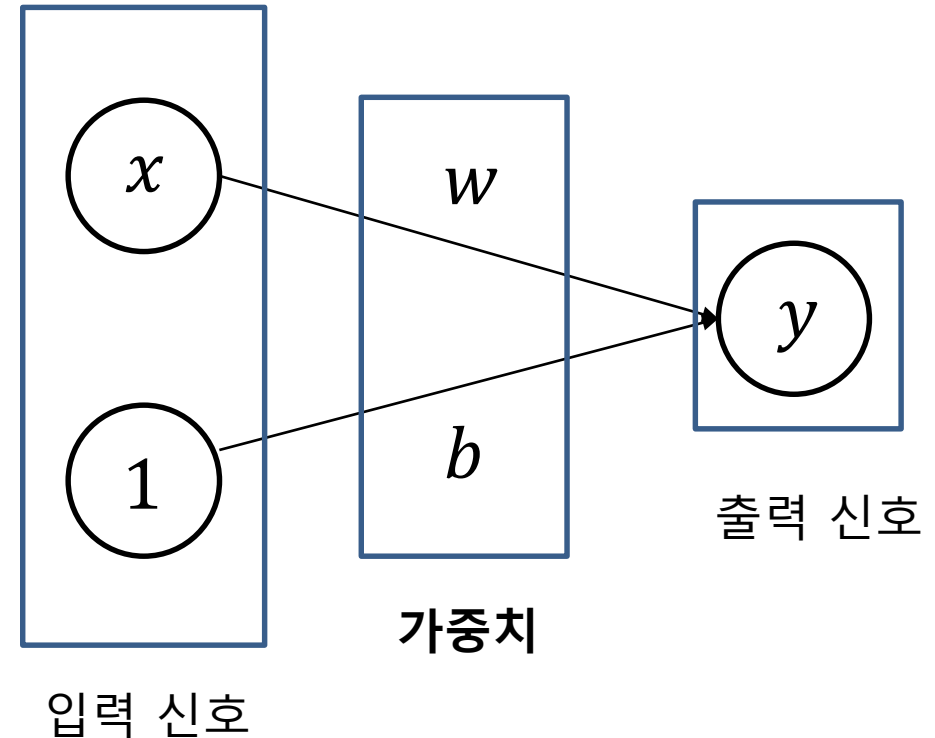
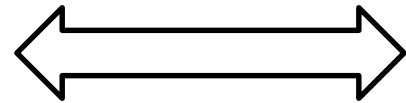
선형회귀와 신경망

Source: <https://blockchain.berkeley.edu/decal/>

■ 퍼셉트론

- 다수의 신호를 입력 받아 하나의 신호를 출력
- 선형회귀는 하나의 퍼셉트론으로 구현 가능

$$wx + b = y$$



역전파와 순전파

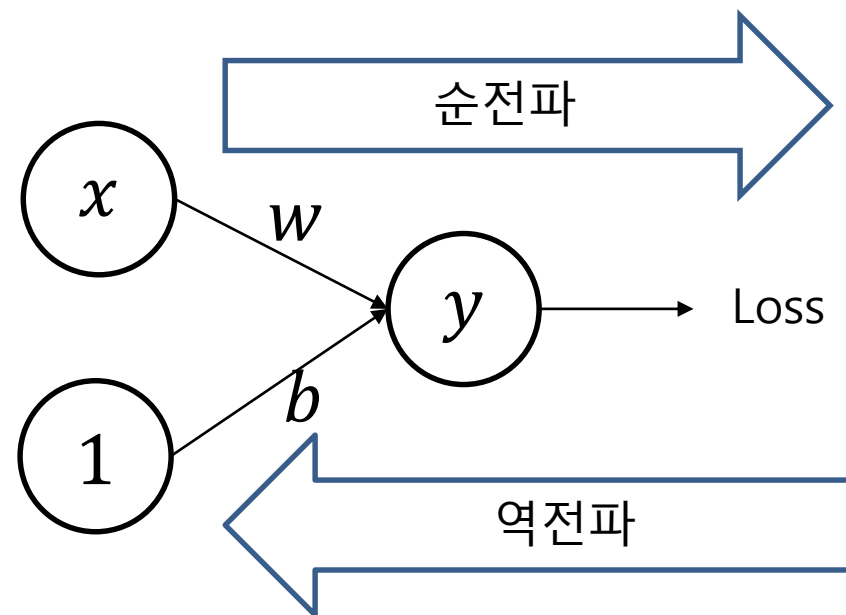
목표: 역전파를 통해 좋은 가중치를 찾자!

■ 순전파

- 데이터처리 → 모델 구현 → 예측값 도출 → 손실함수 계산
- 손실함수는 문제에 따라 정의됨

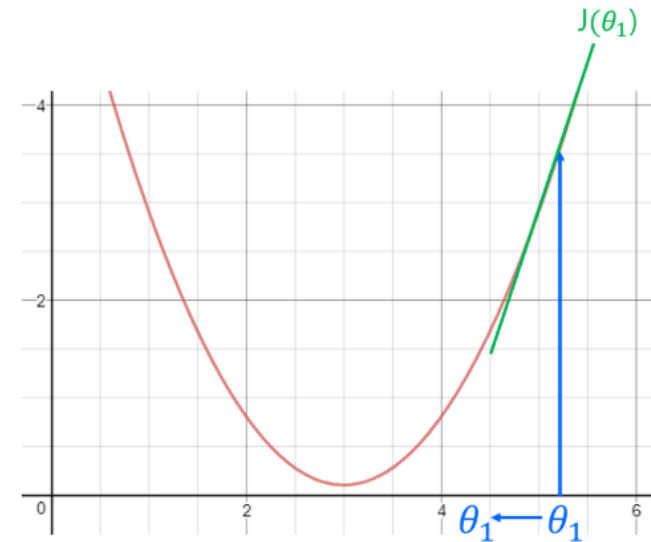
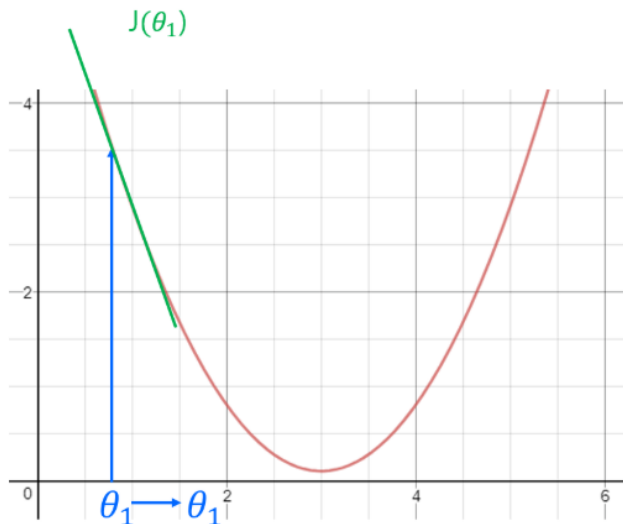
■ 역전파

- 기울기계산 → 개선방향 구하기 → **가중치 개선**
- 개선 방향 = loss를 감소시키는 방향
- 기울기 방향의 (-)는 loss를 감소시키는 방향이다



Gradient Descent

- 손실함수의 기울기를 통해 손실함수를 줄이는 방법
 - 초기점에서 출발
 - 초기점에서 기울기를 구함
 - 기울기의 반대 방향으로 움직임
 - 움직이는 정도는 learning rate를 통해 조절함.



Gradient

■ Gradient

- $f: \mathbb{R}^n \rightarrow \mathbb{R}$ real valued function에 대해서,

$$\frac{\partial f}{\partial \mathbf{x}} := \nabla f(\mathbf{x}) \stackrel{def}{=} \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{n \times 1}$$

- ∂ : partial derivative

선형 신경망

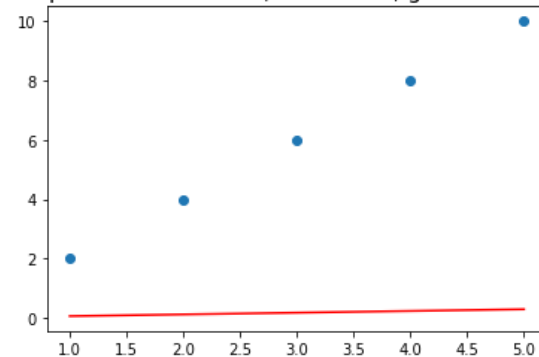
■ 실습 1

```
lr = 0.01

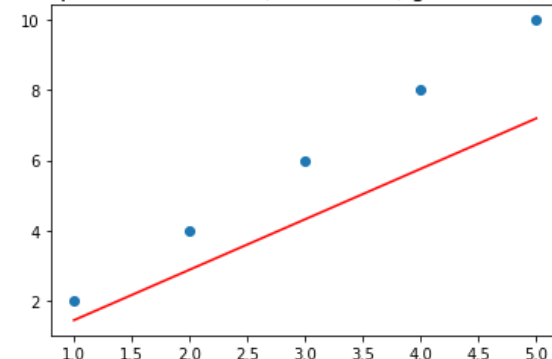
for step in range(20):
    pre = w*x
    cost = ((pre - y) ** 2).sum() / len(x)
    # (wx-y)^2 0/분 1/ 2(wx-y)*x
    grad = 2*(pre-y).view(5).dot(x.view(5))/len(x)
    w -= lr*grad

    if step % 5 == 0:
        plt.scatter(x.numpy(), y.numpy())
        plt.plot(x.numpy(), pre.numpy(), 'r-')
        # w.size() = 1*1, grad.size() = 1
        plt.title('step %d : cost=%.4f, w=%.4f, grad=%.4f' % (step, cost.item(), w.item(), grad.item()))
        plt.show()
```

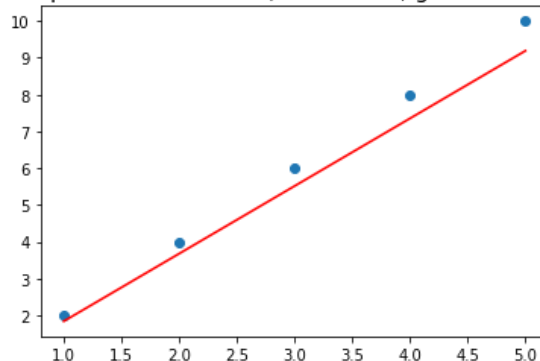
step 0 : cost=41.4864, w=0.4852, grad=-42.7247



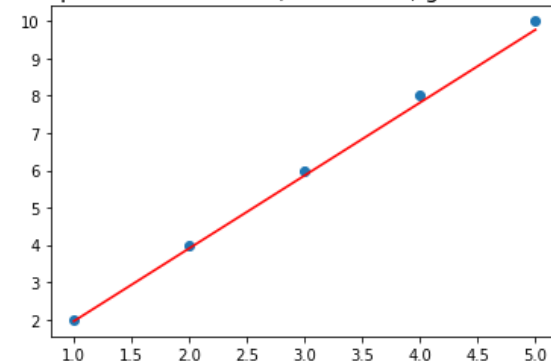
step 5 : cost=3.4582, w=1.5627, grad=-12.3354



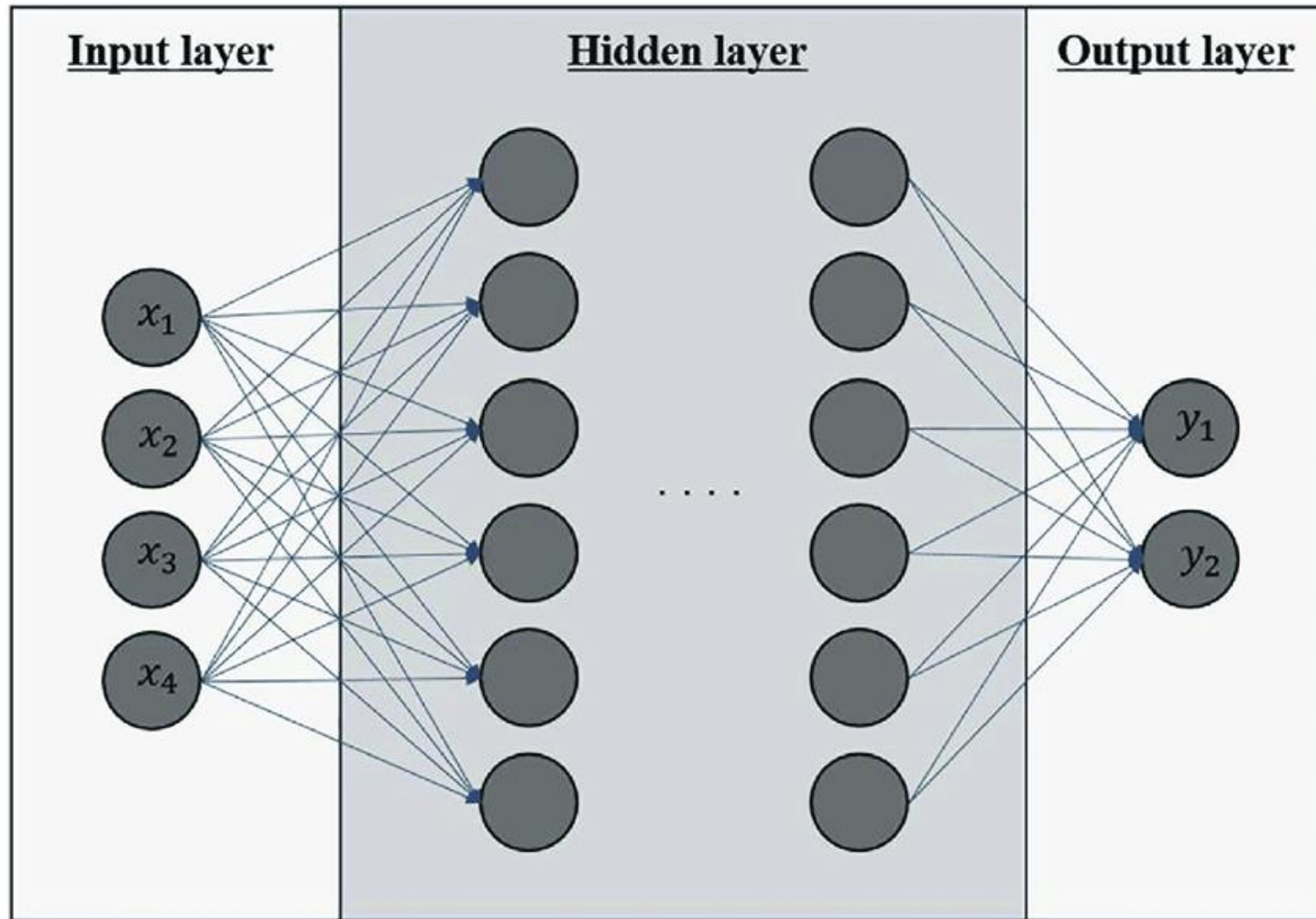
step 10 : cost=0.2883, w=1.8737, grad=-3.5614



step 15 : cost=0.0240, w=1.9635, grad=-1.0282



선형 신경망

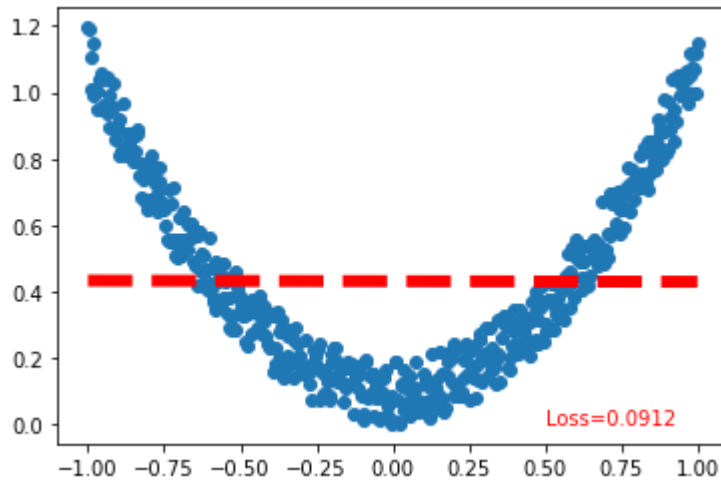


모든 퍼셉트론이 선형이라면?

선형 신경망

■ 실습 2

- $y = x^2$ 그래프를 linear layer 2개를 사용한 모델로 예측하면?



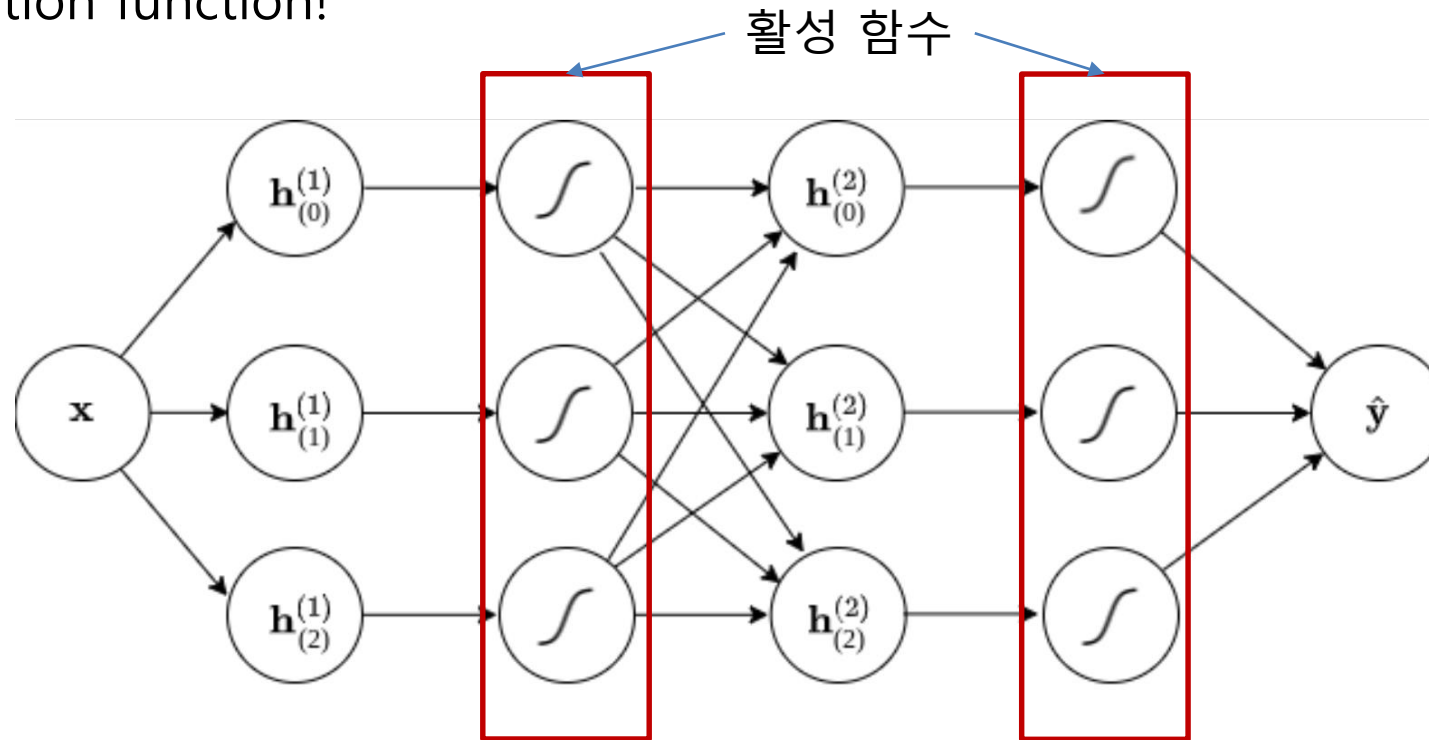
→ Linear layer를 여러 개 쌓는 것은 의미가 없다!

활성함수

■ 비선형성

- 선형결합의 선형결합은 선형결합이기 때문에, linear layer를 여러 번 쌓은 것은 의미가 없음
- 즉, 비선형성을 갖는 함수가 필요함

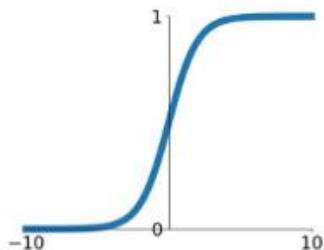
➔ Activation function!



활성함수

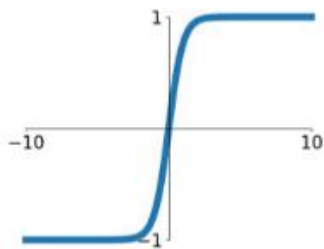
Sigmoid

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



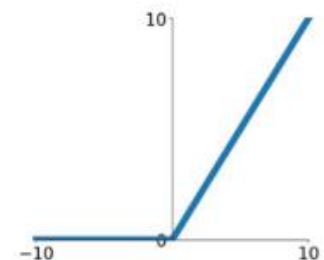
tanh

$$\tanh(x)$$



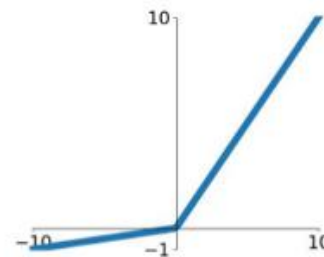
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

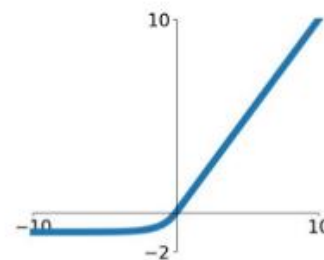


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



활성함수

■ Sigmoid

- 범위는 0과 1 사이 (s형태의 커브 생성)
- 미분 가능 (기울기가 항상 1보다 작음)
- Binary classification에서 출력층에서 사용(0.5보다 작으면 0, 0.5보다 크면 1로 분류)

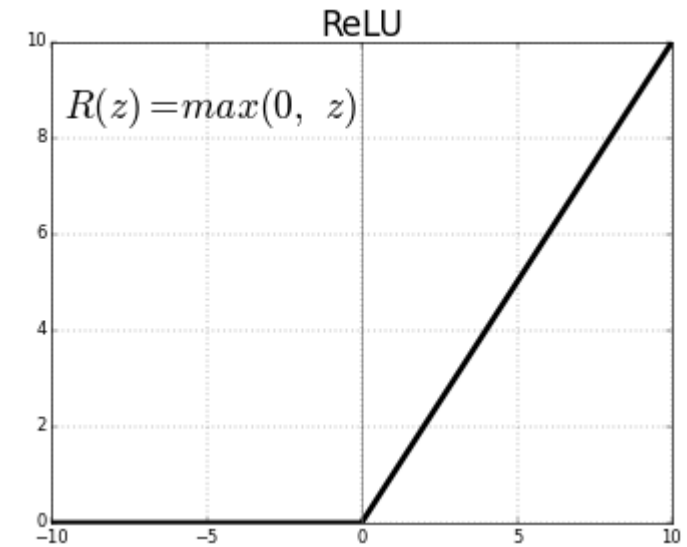
■ Softmax

- N개의 다른 이벤트에 대한 확률 분포
- 일반적으로 각 대상 클래스의 확률 계산, 모든 확률의 합 = 1
- Multiclass classification 문제의 출력층에서 사용

활성함수

■ ReLU

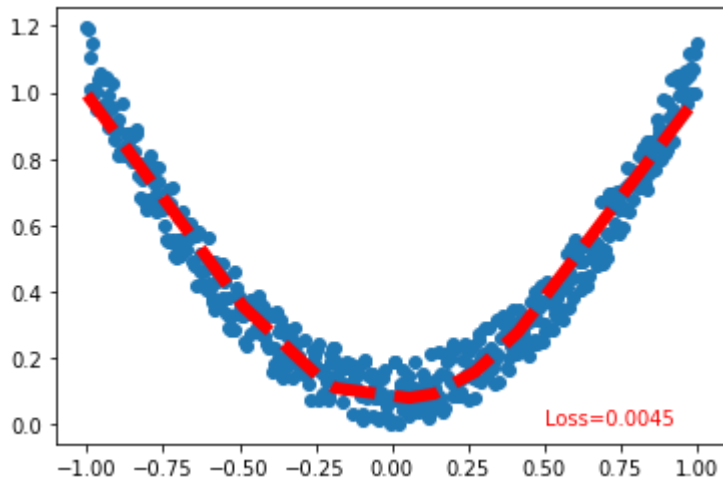
- 입력값이 0보다 크면 입력값을 그대로 출력, 0보다 작으면 0을 출력
- 값을 그대로 보내기 때문에 계산 및 학습이 빠름
- 은닉층(hidden layer)에서 많이 사용함



활성함수

■ 실습 3

- Linear layer와 activation function을 사용해서 $y = x^2$ 그래프를 예측



→ Activation function을 사용하면 비선형성을 나타낼 수 있다.

Loss function

- MSE Loss(mean squared error)
 - 예측한 값과 실제값의 차이를 제공한 것의 평균(앞선 예시에서 사용)
- BCE Loss(Binary Cross Entropy)
 - Binary classification(0, 1의 class로 분류) 문제에서 사용하는 cross entropy
- CE Loss(Cross Entropy)
 - Classification에서 one hot label과의 cross entropy를 이용해 loss를 구함
 - Torch에서는 softmax와 cross entropy를 합쳐 놓은 것으로 제공

Summary

■ Regression

- Activation function(출력층): 항등함수
- Loss function: MSE

■ Binary classification

- Activation function(출력층): sigmoid/softmax
- Loss function: BCE/CEE

■ Multiclass classification

- Activation function(출력층): softmax
- Loss function: CEE

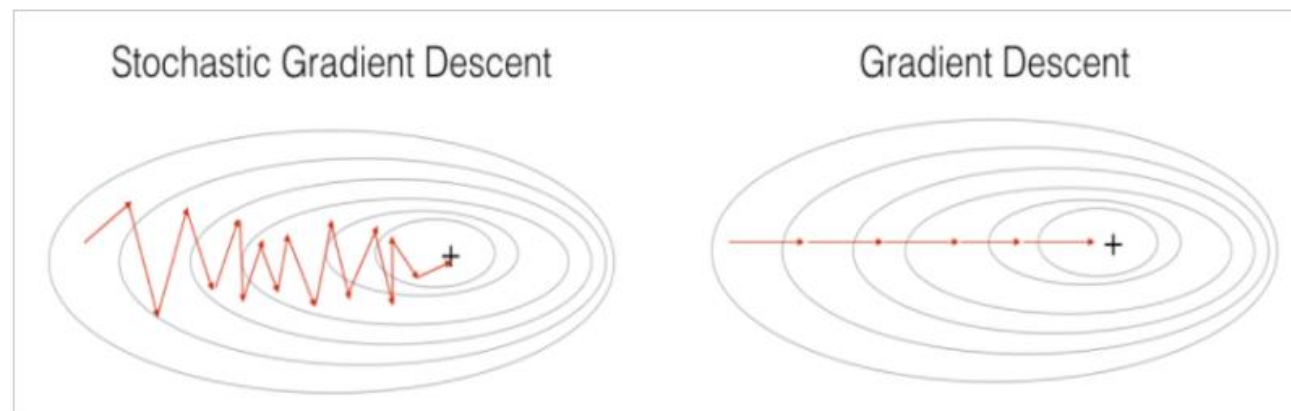
Optimizer

■ SGD(stochastic gradient descent)

- $w' = w - lr * grad$
- 각 sample 혹은 batch에 대해 반복 시행
- 최적 경로가 아닐 수 있음
- 계산량이 적기 때문에 빠른 학습 가능

■ GD(gradient descent)

- $w' = w - lr * grad$
- 모든 training data에 대해 반복 시행
- 최적 경로를 탐색
- 계산량이 큼



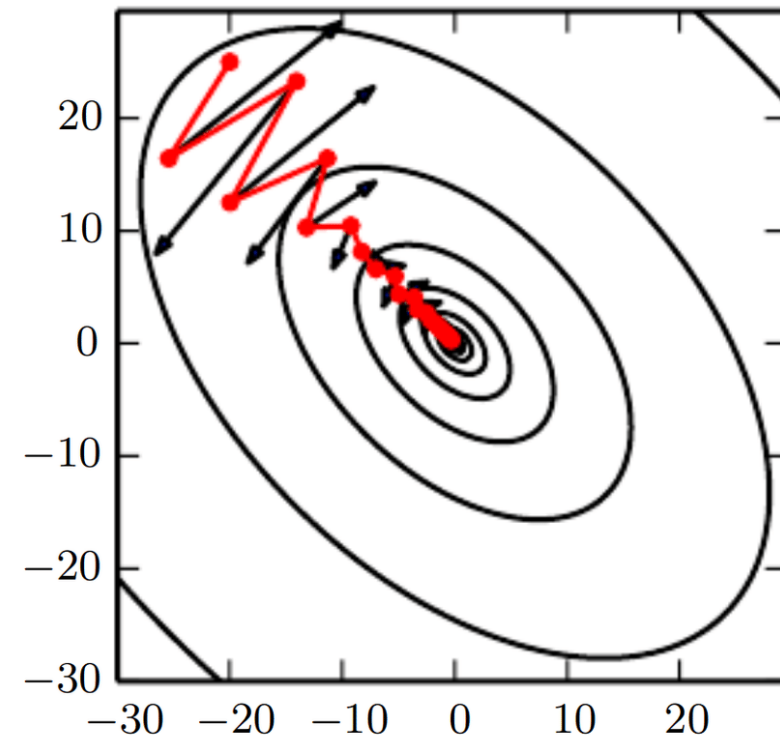
Optimizer

■ SGD의 단점

- 기울기가 최소점을 가르키지 않는 경우 지그재그로 움직임

■ Momentum

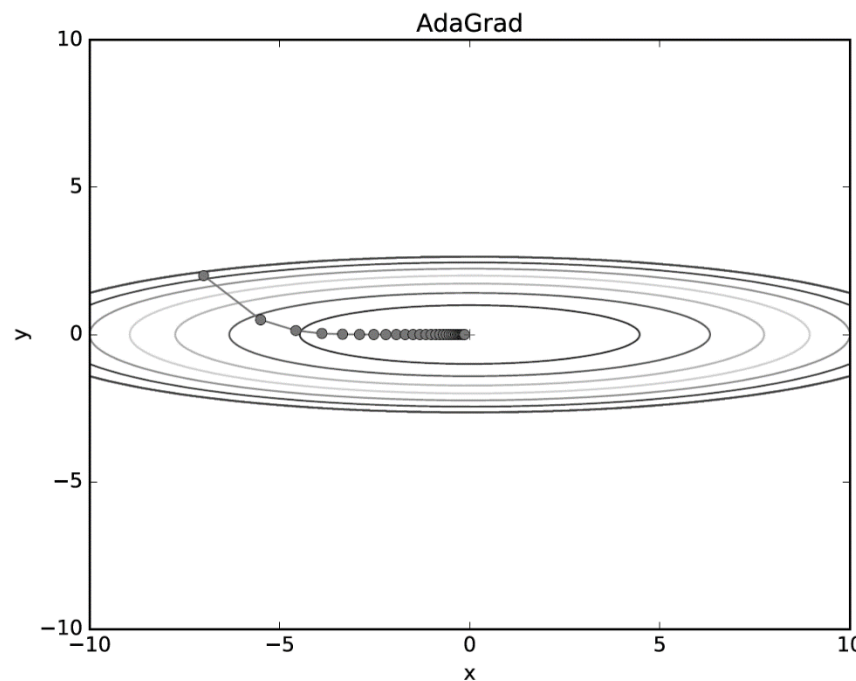
- 관성을 가지는 모델
- 이전 변화량을 가중치 업데이트에 사용
- $v' = \alpha * v - lr * grad$
- $w' = w + v'$



Optimizer

■ AdaGrad

- 학습률을 감소시키면서 학습을 진행함.
- $h' = h + grad \odot grad$
- $w' = w - lr \frac{1}{\sqrt{h' + \epsilon}} grad$
- 크게 움직일수록 학습률 감소
- 단점: 어느 순간 이후 학습률 = 0이 될 수 있음.



Optimizer

■ RMSProp

- 학습률을 감소시키면서 학습을 진행함.
- 가장 가까운 기울기에 가중, 과거 기울기 반영 규모를 기하급수적으로 감소
- $h' = \gamma h + (1 - \gamma) * grad \odot grad$
- $w' = w - lr \frac{1}{\sqrt{h' + \epsilon}} grad$

■ Adam

- **Adagrad** + **Momentum**
- 매개변수 2개 지정
- RMSProp를 일반화

Batch Training

■ Batch Training

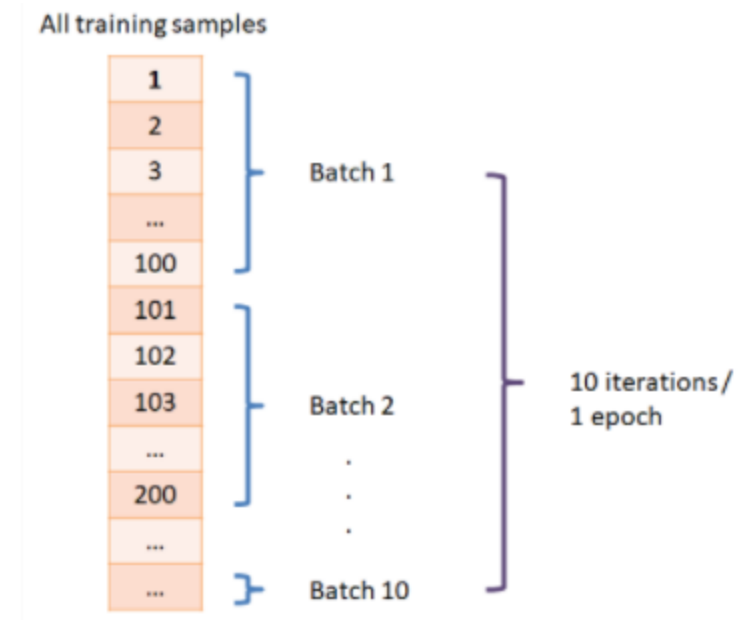
- 한번에 하나의 이미지만 학습하면 학습속도가 느림
- 한번에 전부 다 학습시키면 out of memory
- 적당한 양의 데이터를 묶어 한번에 학습시킨다
- 적당한 양 = Batch size

■ Epoch

- 한 데이터가 총 몇 번 학습에 사용되나?

■ Iteration

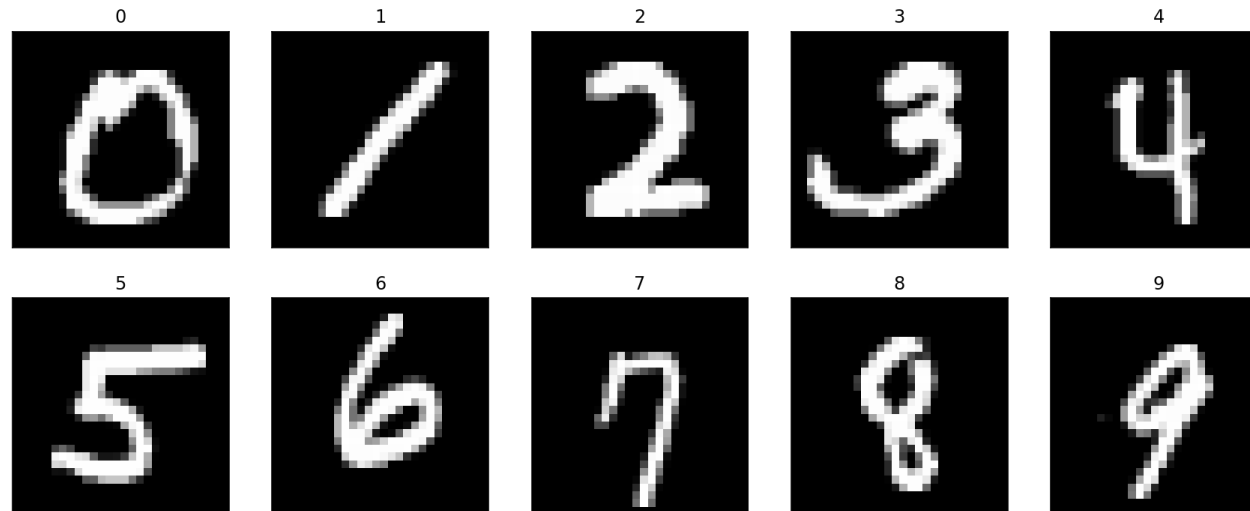
- 1 epoch에 총 몇 개의 batch를 사용하냐?



<https://geniewishescometrue.tistory.com/entry/ML-DL-WIKI-BatchBatch-sizeEpochIteration>

MNIST

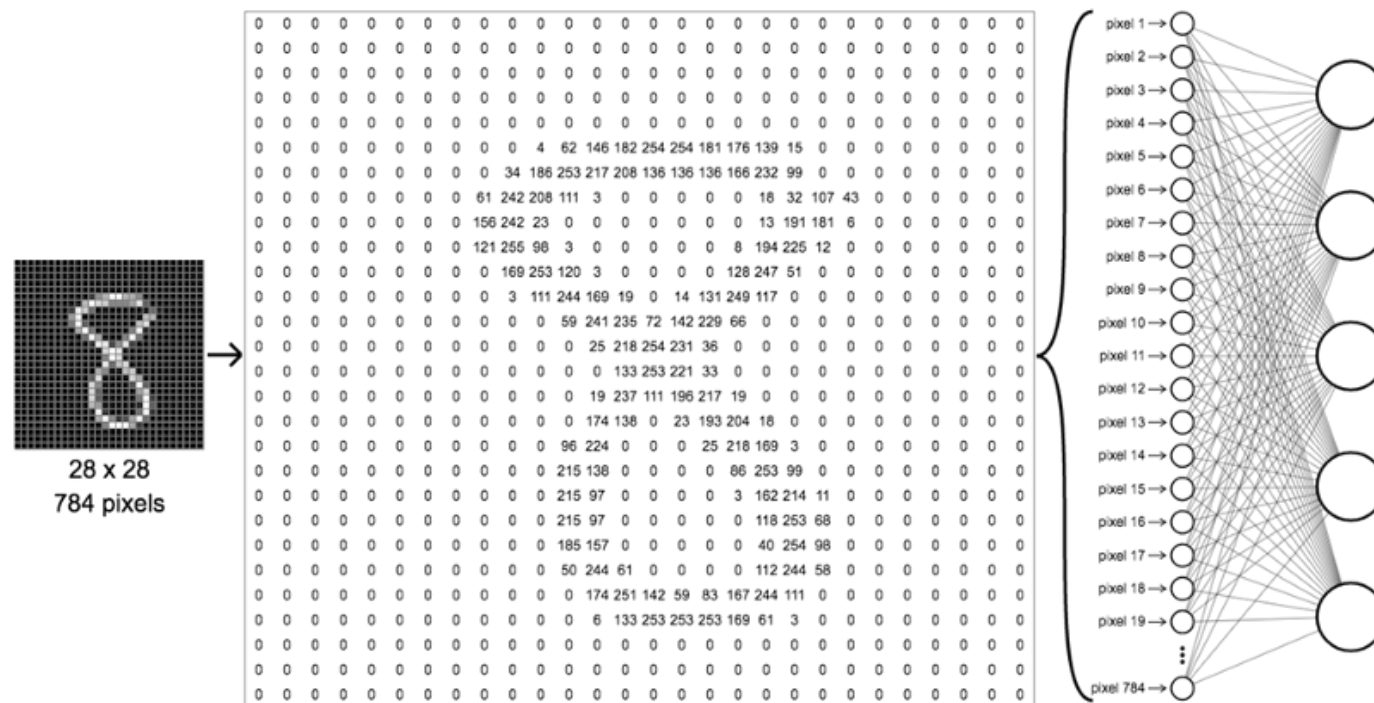
- MNIST: 손글씨 숫자 이미지 데이터
 - Training set: 60000장
 - Test set: 10000장
 - Size: 28 * 28
 - 흑백 이미지



MNIST

■ MNIST를 위한 Neural Network

- Size : $28 \times 28 = 784$
- Output : 10 (이미지가 0일 확률 ~ 9일 확률)



실습