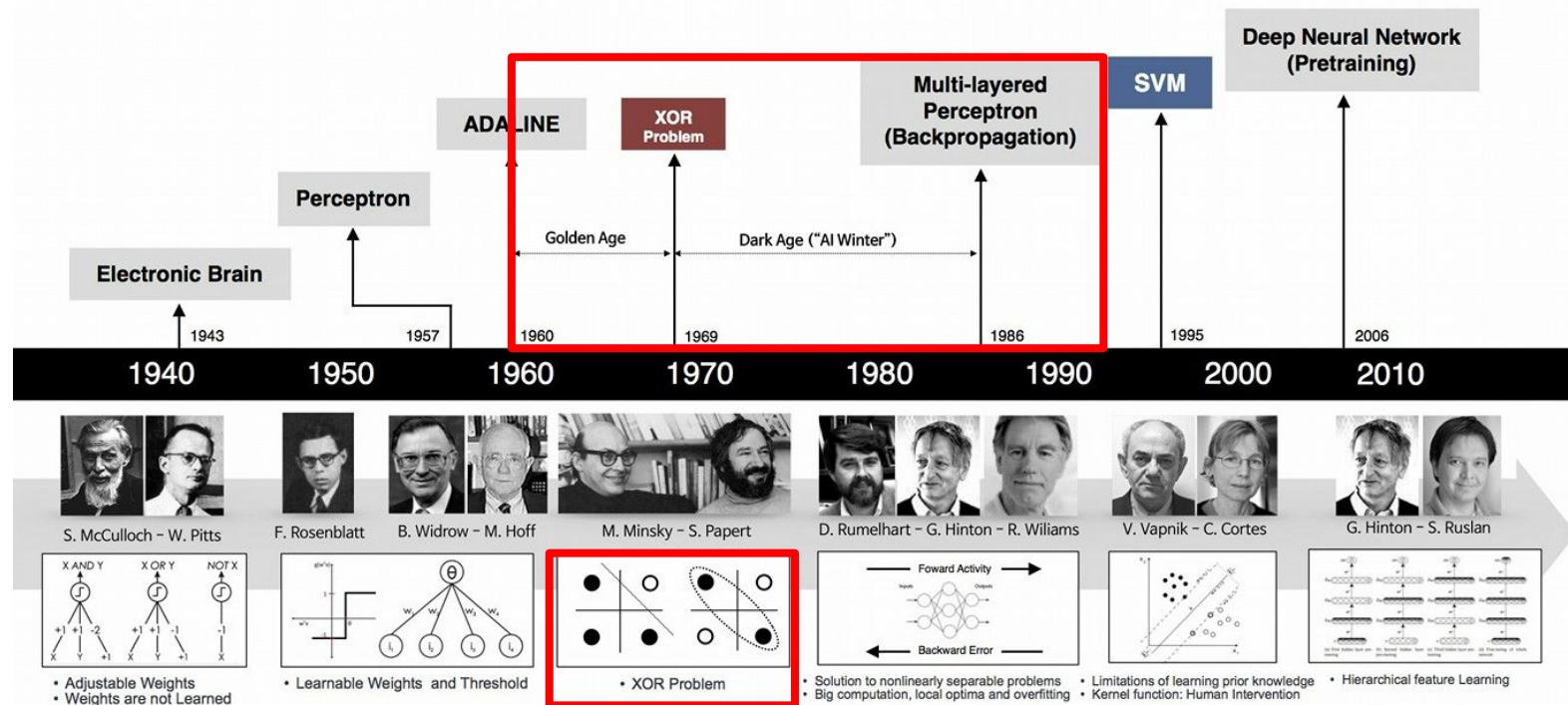


Tensorflow day2

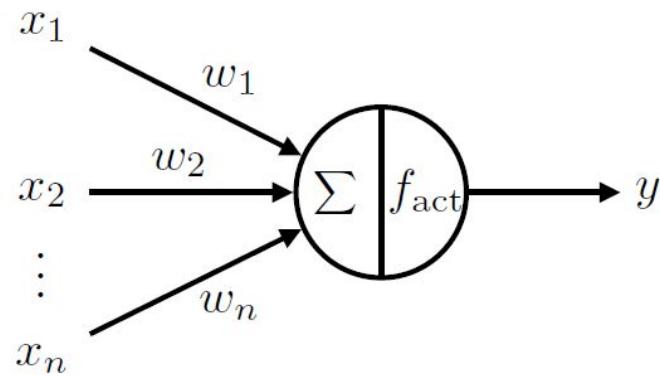
모두의 연구소 Lab. Rubato
소준섭

XOR Problem



Perceptron

- Frank Rosenblatt, 1957
- Neural network의 시초
- 몇 개의 binary inputs: x_1, x_2, \dots, x_n
- 한 개의 binary output: y
- 가중치 (Weights): w_1, w_2, \dots, w_n



$$\sum = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

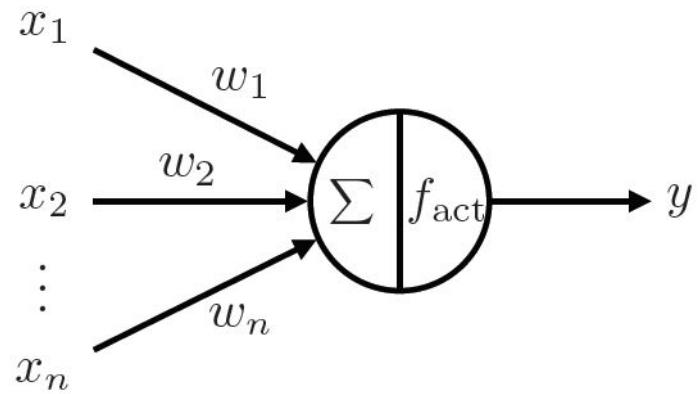
f_{act} : Heaviside step function

Perceptron

- 작동 방식

$$y = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

$$y = f_{\text{act}} \left(\sum_j w_j x_j - \text{threshold} \right)$$



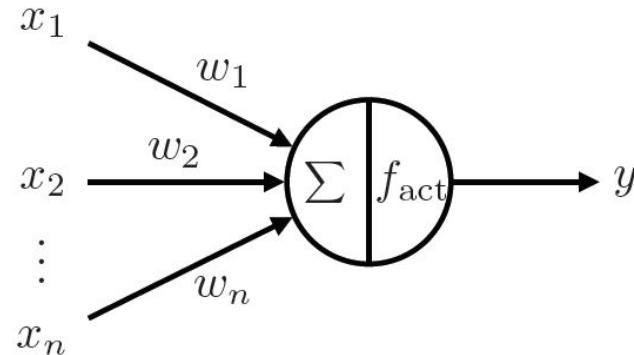
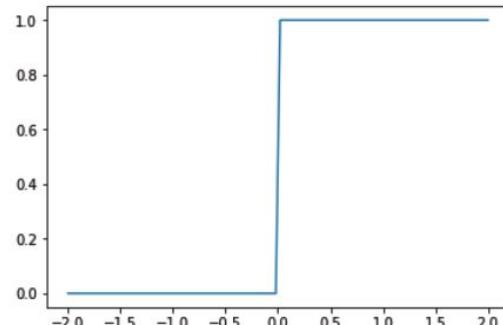
- 신호의 가중합(weighted sum)이 특정 임계값을 넘어서면 활성화(activation)
- 가중치(weights)는 각 입력값(inputs)의 영향력을 조절하는 요소

Perceptron

- 활성화 함수 (activation function)

$$y = f_{\text{act}} \left(\sum_j w_j x_j + b \right)$$

- Heaviside step function

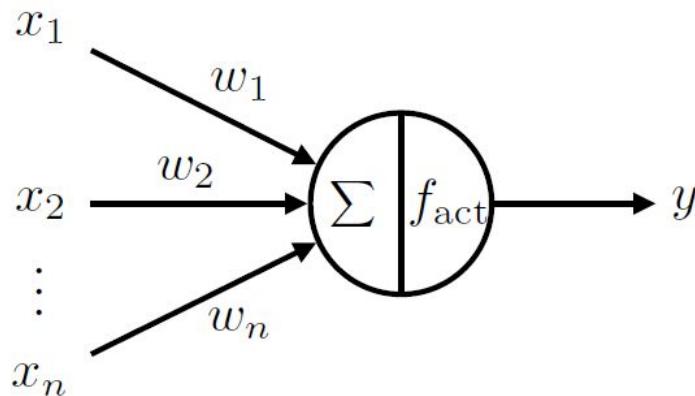


activation function:
Heaviside step function
bias:
-threshold

Perceptron

$$\sum_j w_j x_j + b$$

linear boundary
in d-dimensional space



Perceptron:
Binary classifier

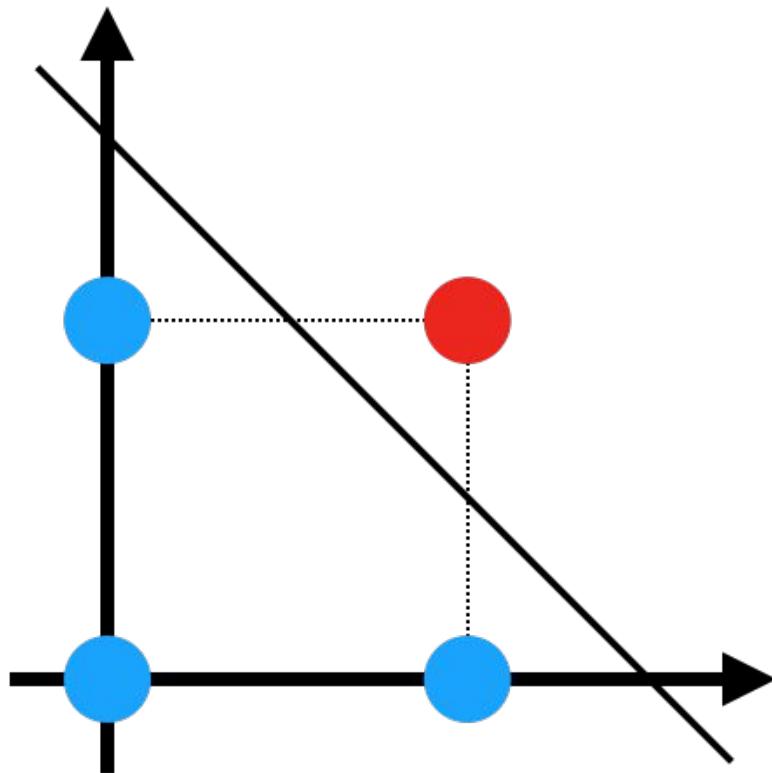
$$y = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

$$y = f_{\text{act}} \left(\sum_j w_j x_j + b \right)$$

$$y = f_{\text{act}} (\mathbf{w} \mathbf{x} + b)$$

where \mathbf{w}, \mathbf{x} : vectors

XOR problem - AND Gate

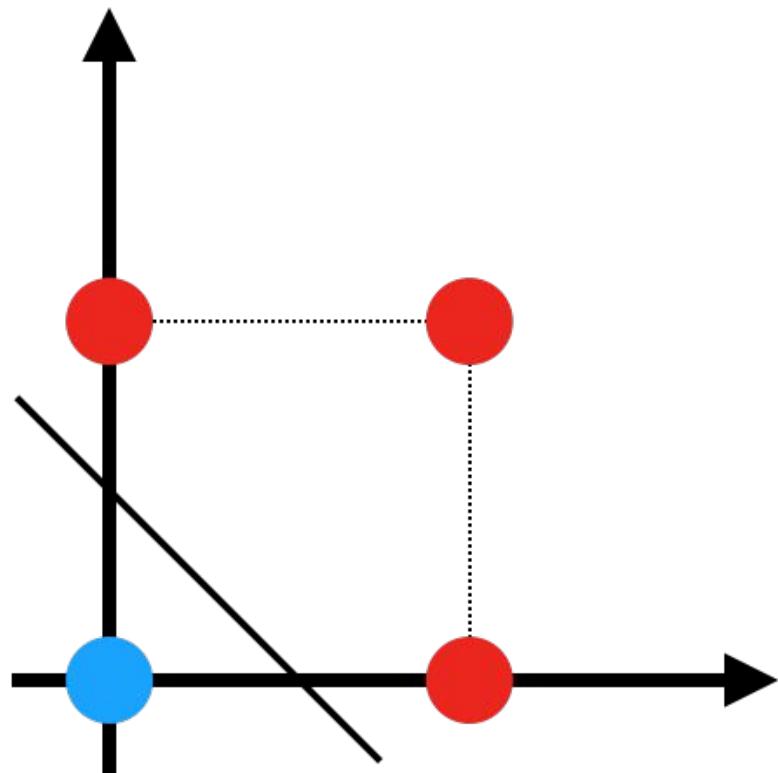


$$w_1 = 2, \ w_2 = 2, \ b = -3$$

x_1	x_2	Σ	y
0	0	-3	0
0	1	-1	0
1	0	-1	0
1	1	1	1

XOR problem - OR Gate

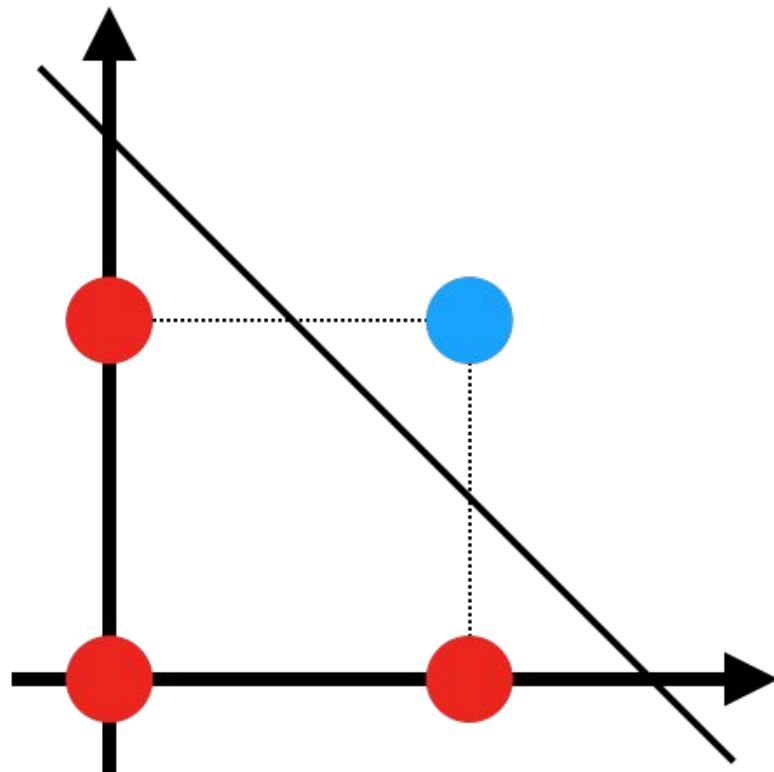
$$w_1 = 2, \ w_2 = 2, \ b = -1$$



x_1	x_2	Σ	y
0	0	-1	0
0	1	1	1
1	0	1	1
1	1	3	1

XOR problem - NAND Gate

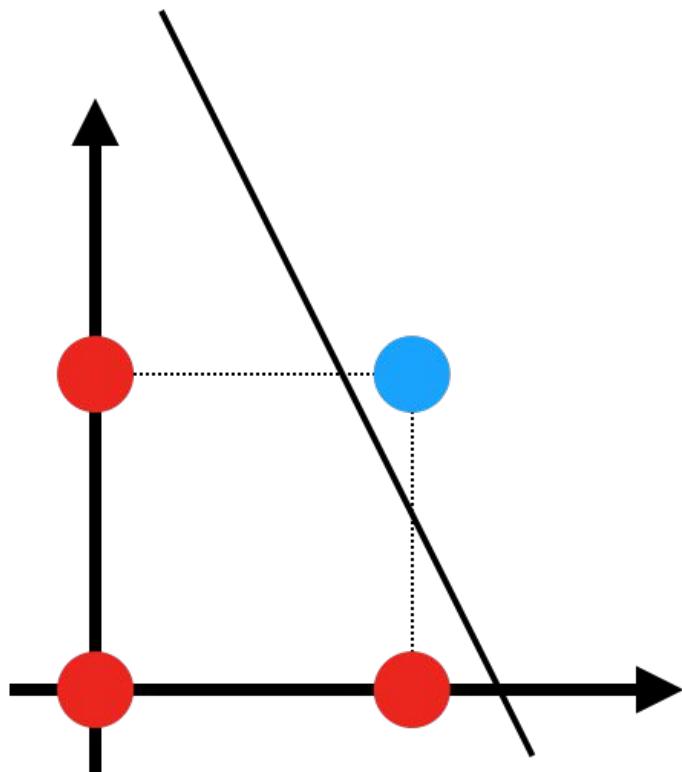
$$w_1 = -2, \ w_2 = -2, \ b = 3$$



x_1	x_2	Σ	y
0	0	3	1
0	1	1	1
1	0	1	1
1	1	-1	0

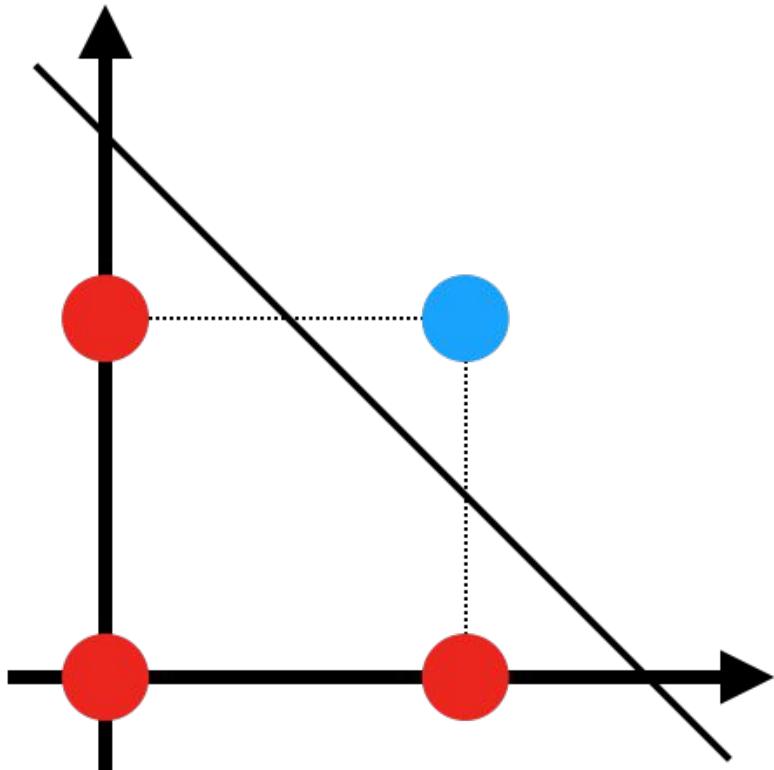
XOR problem - NAND Gate

$$w_1 = -2, \ w_2 = -1, \ b = 2.5$$



x_1	x_2	Σ	y
0	0	2.5	1
0	1	1.5	1
1	0	0.5	1
1	1	-0.5	0

XOR problem - NAND Gate

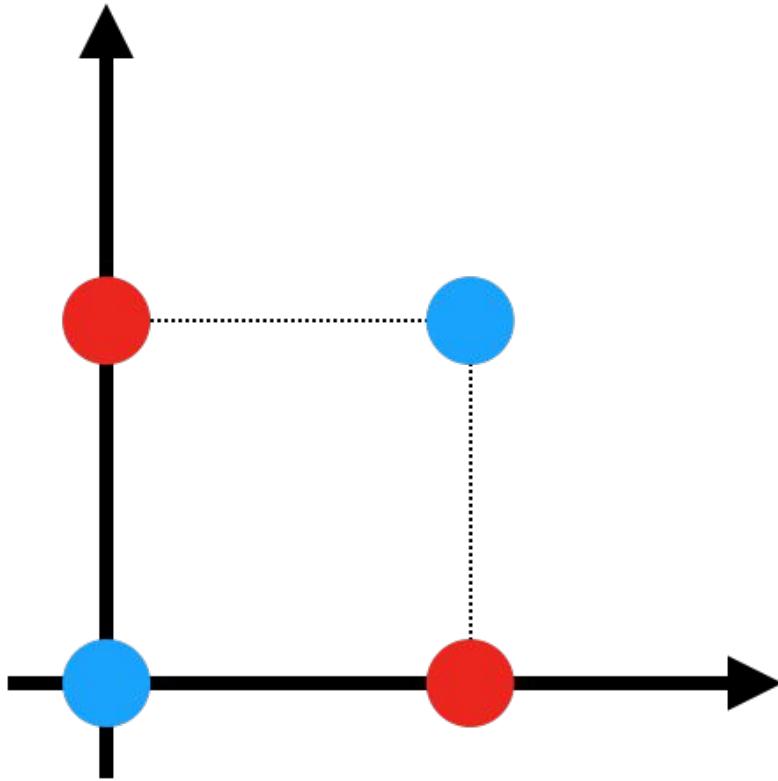


$$w_1 = -2, \ w_2 = -2, \ b = 3$$

$$w_1 = -2, \ w_2 = -1, \ b = 2.5$$

⋮

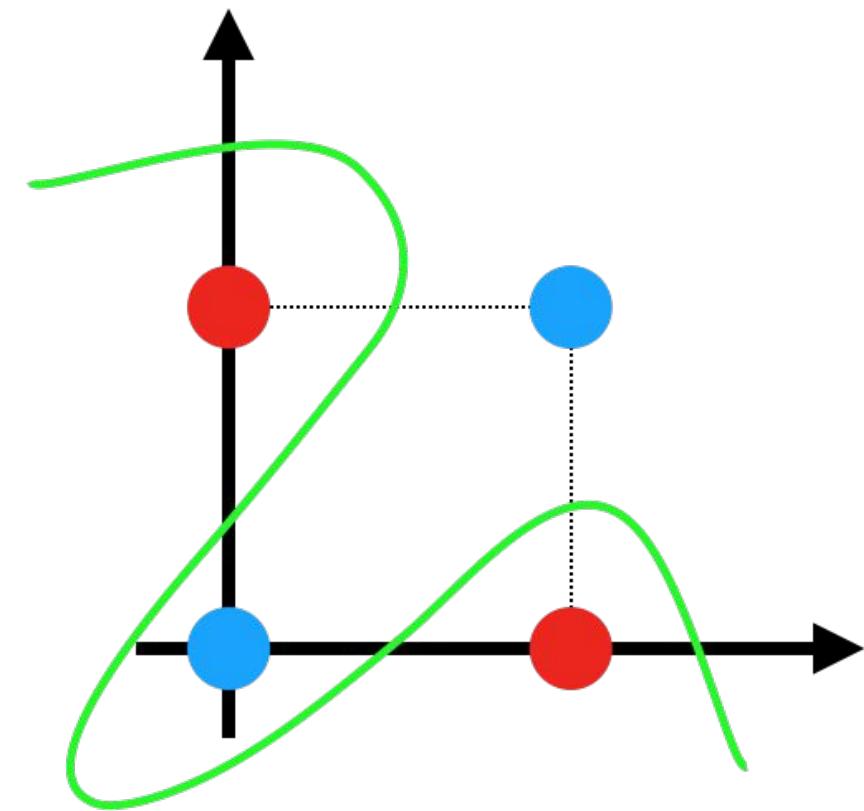
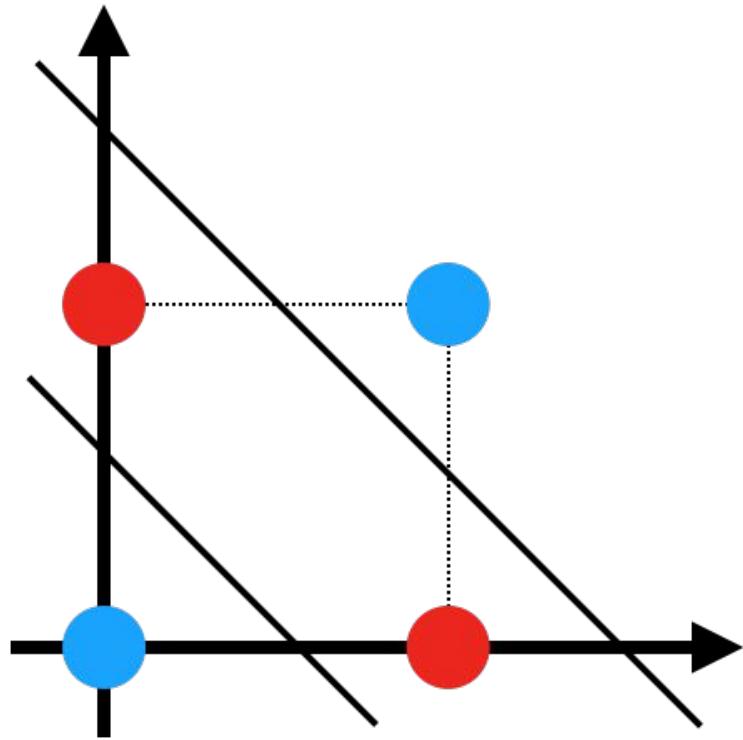
XOR problem



$w_1 = ?$, $w_2 = ?$, $b = ?$

x_1	x_2	\sum	y
0	0		0
0	1		1
1	0		1
1	1		0

XOR problem



XOR problem

$$w_1 = -2, \ w_2 = -2, \ b = 3$$

x_1	x_2	Σ	s_1
0	0	3	1
0	1	1	1
1	0	1	1
1	1	-1	0

$$w_1 = 2, \ w_2 = 2, \ b = -1$$

x_1	x_2	Σ	s_2
0	0	-1	0
0	1	1	1
1	0	1	1
1	1	3	1

$$w_1 = 2, \ w_2 = 2, \ b = -3$$

s_1	s_2	Σ	y
1	0	-1	0
1	1	1	1
1	1	1	1
0	1	-1	0

XOR problem

$$w_1 = -2, \ w_2 = -2, \ b = 3$$

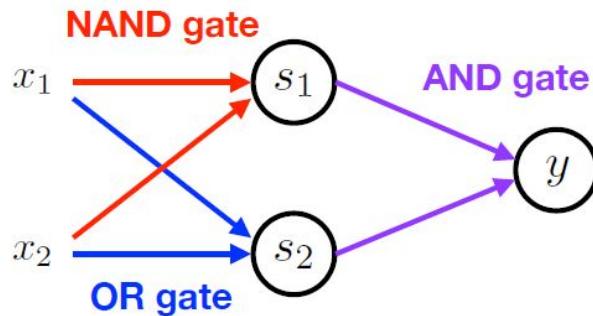
x_1	x_2	\sum	s_1
0	0	3	1
0	1	1	1
1	0	1	1
1	1	-1	0

$$w_1 = 2, \ w_2 = 2, \ b = -1$$

x_1	x_2	\sum	s_2
0	0	-1	0
0	1	1	1
1	0	1	1
1	1	3	1

$$w_1 = 2, \ w_2 = 2, \ b = -3$$

s_1	s_2	\sum	y
1	0	-1	0
1	1	1	1
1	1	1	1
0	1	-1	0



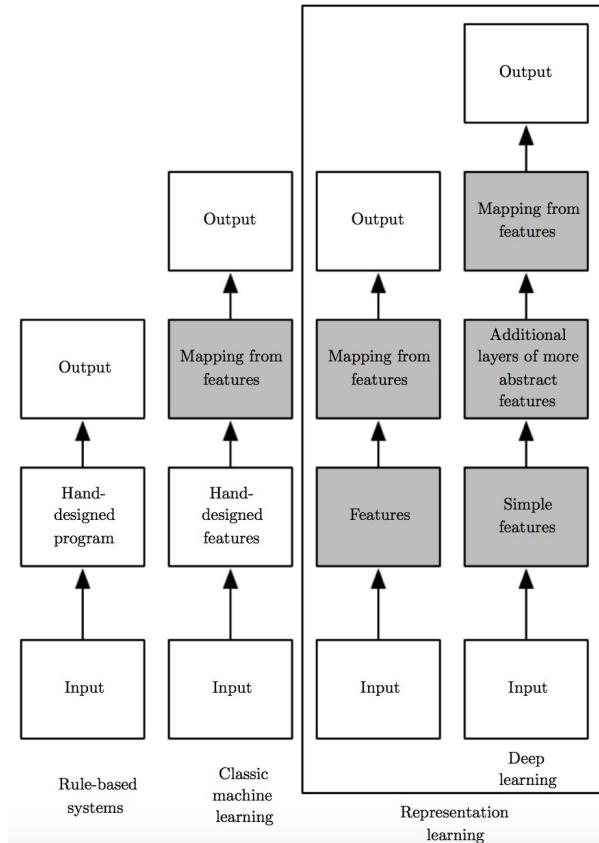
XOR problem



to map space X to space Y

Deep Neural Networks to Deep Learning

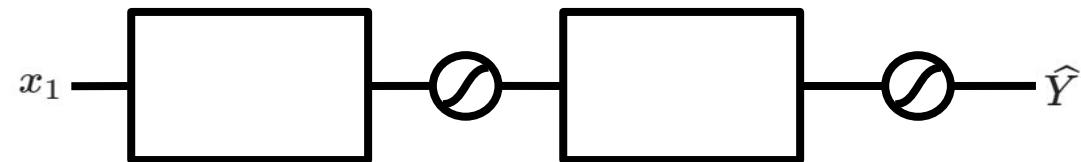
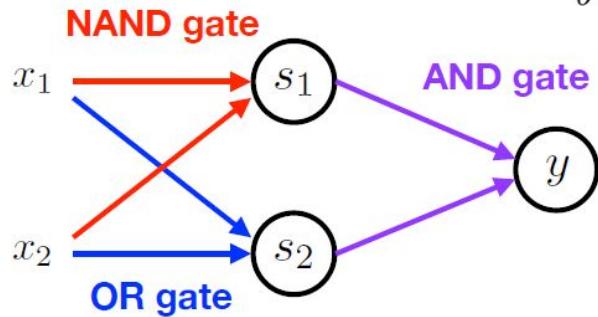
- Learning multiple levels of representation
- Representation / Feature learning
- Generally many layers
- From high dimensional vector to low dimensional vector (vice versa)
- F. Chollet's quotation: **to map space X to space Y using a continuous geometric transform**



XOR problem with NN

$$W = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$
$$b = -3$$

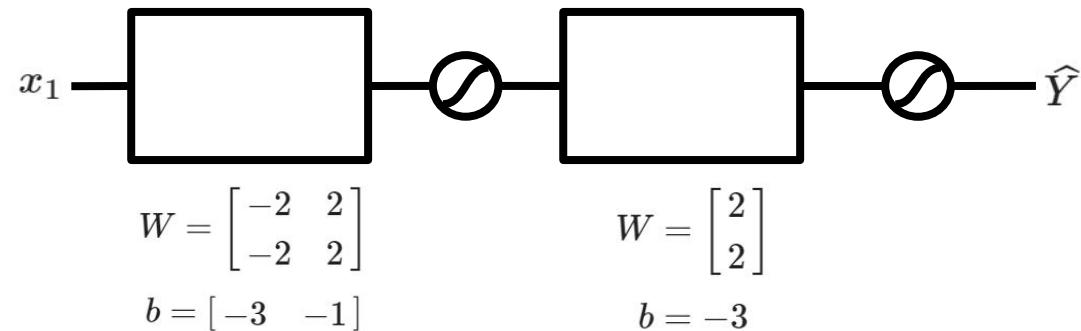
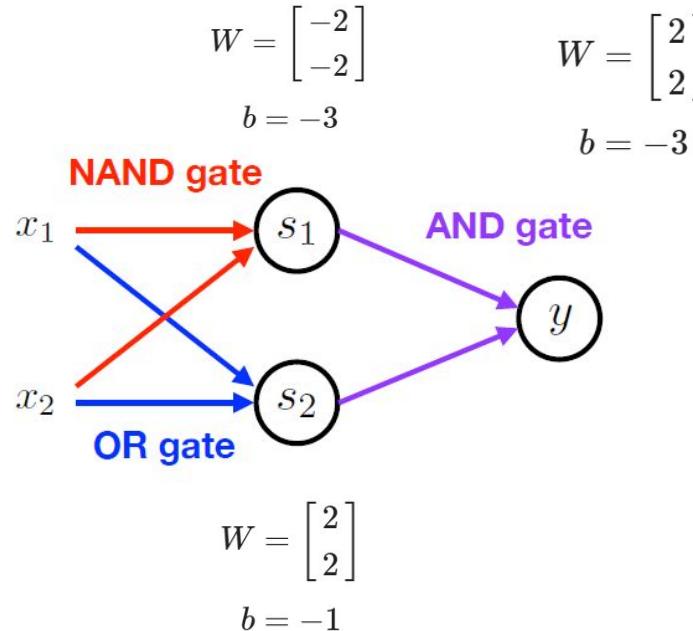
$$W = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$
$$b = -3$$



$$W = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$b = -1$$

XOR problem with NN



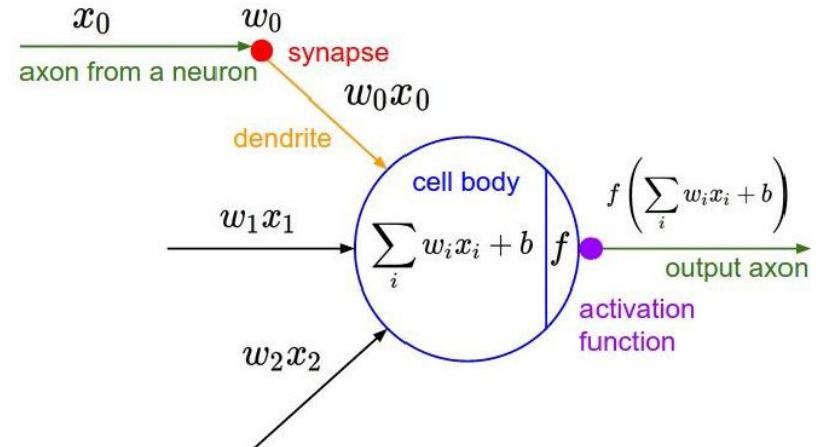
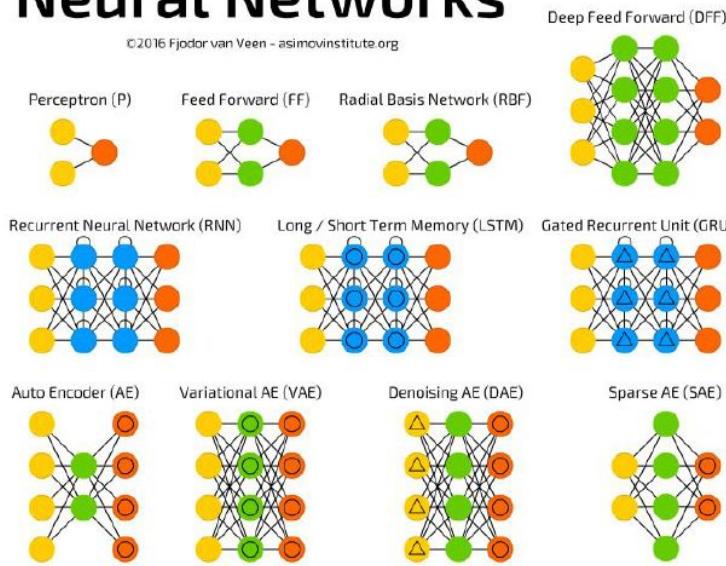
Neural Networks

A mostly complete chart of

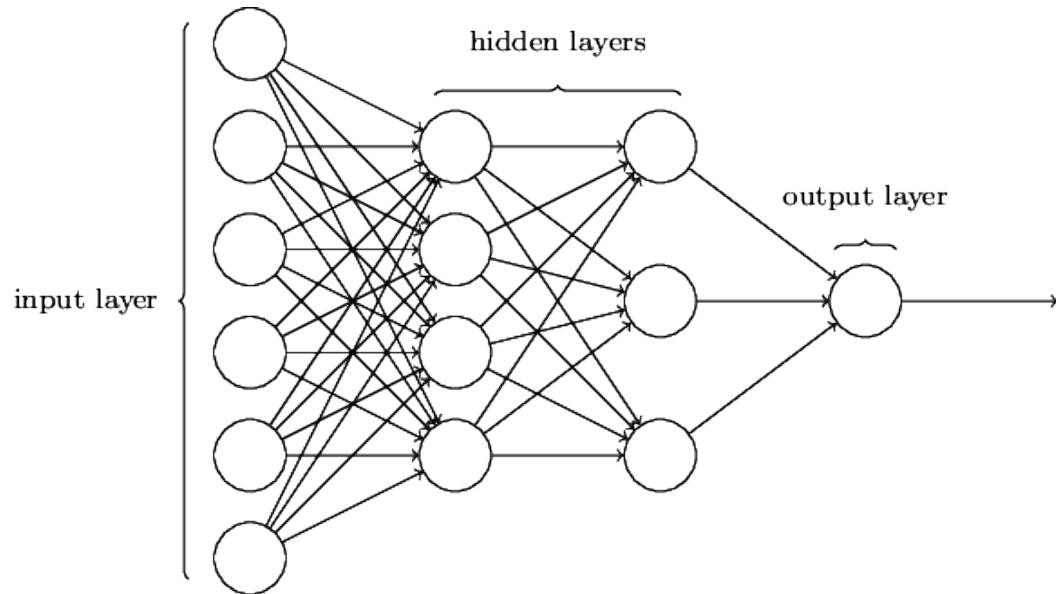
Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

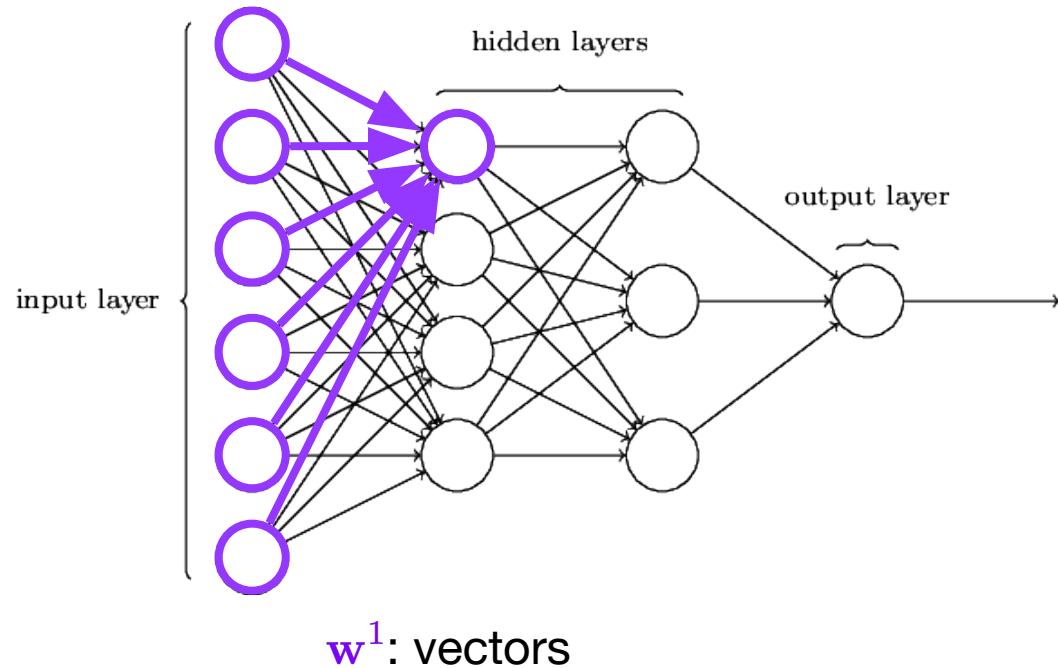
- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool



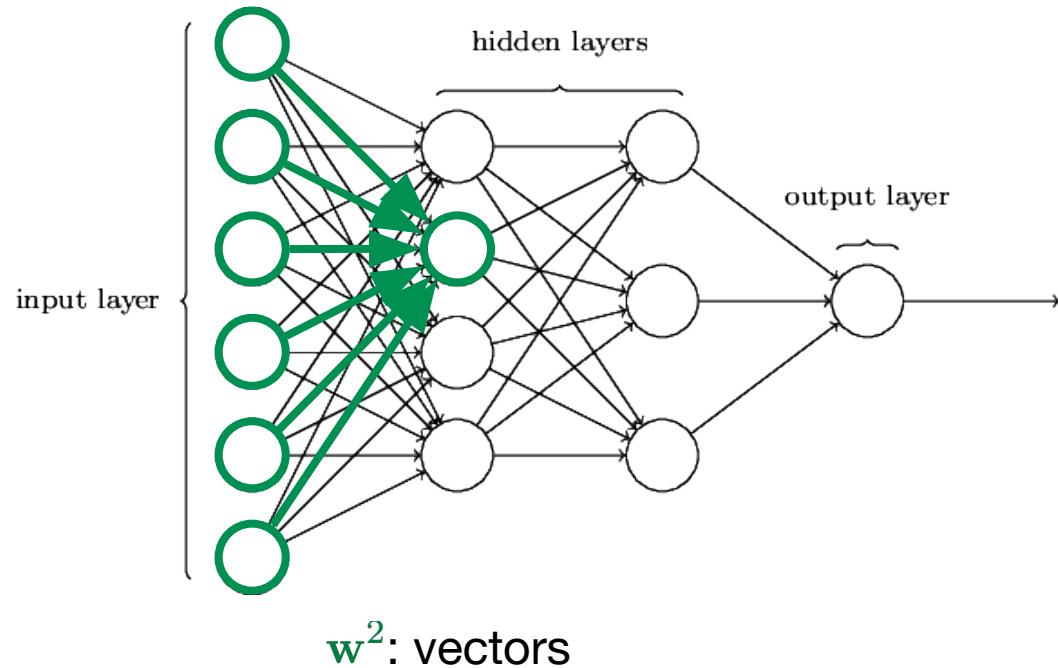
Neural Networks



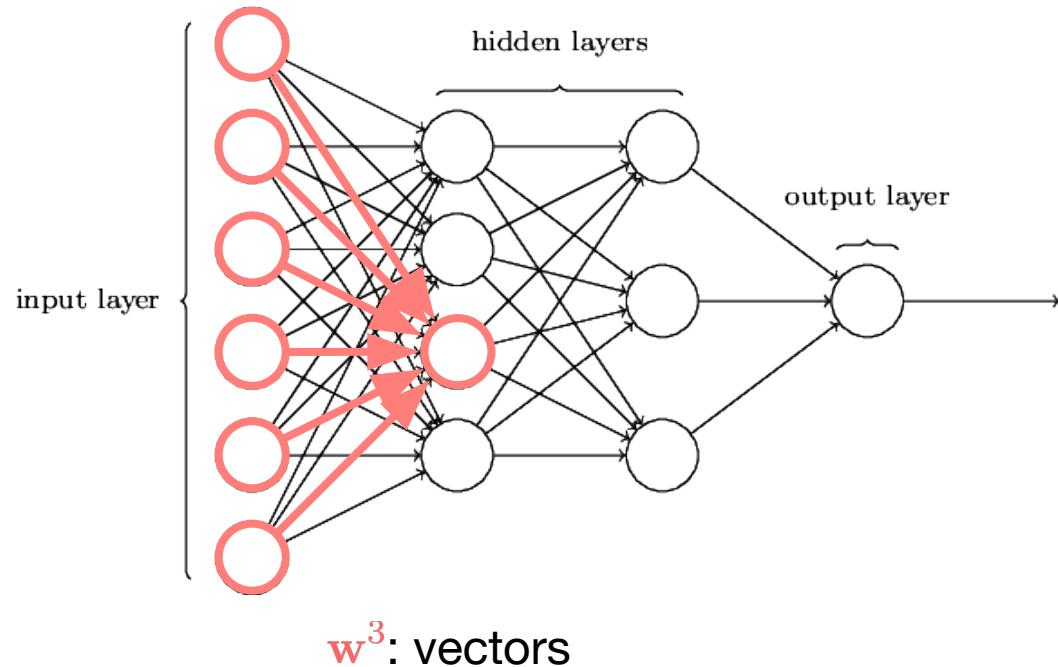
Neural Networks



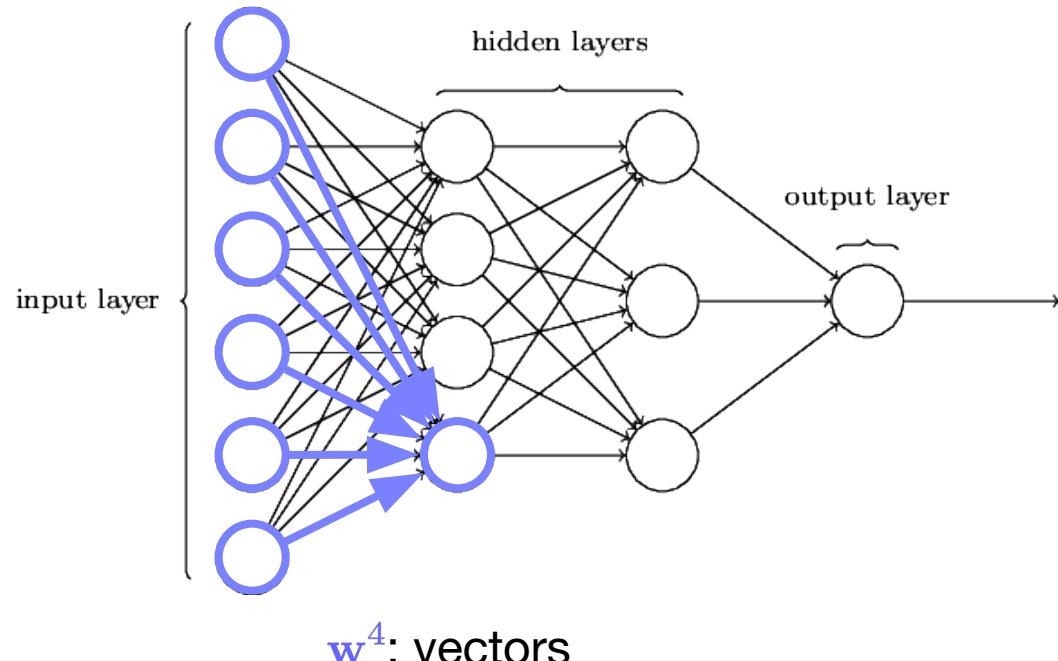
Neural Networks



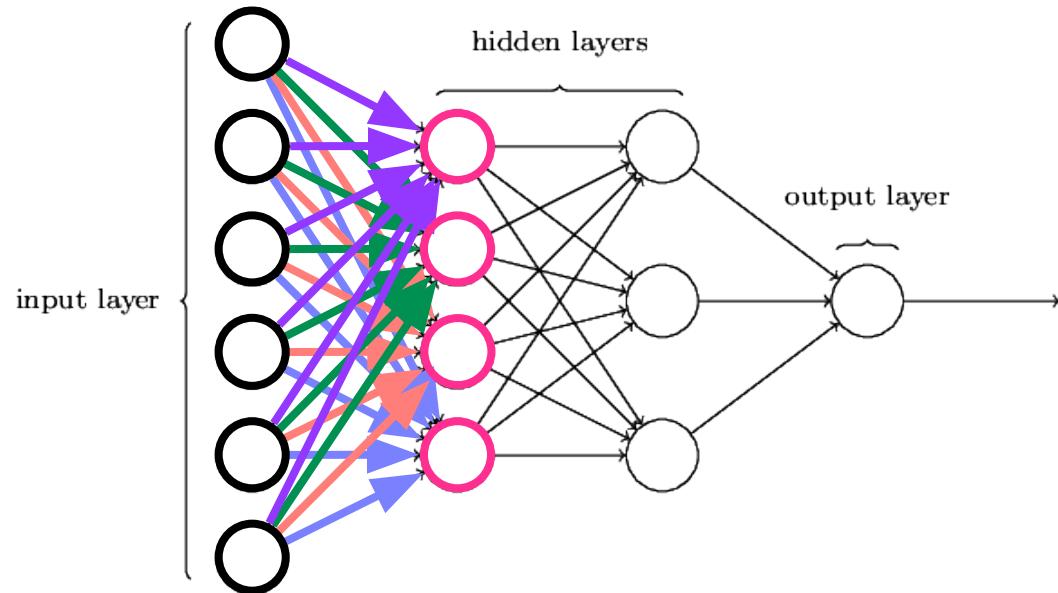
Neural Networks



Neural Networks



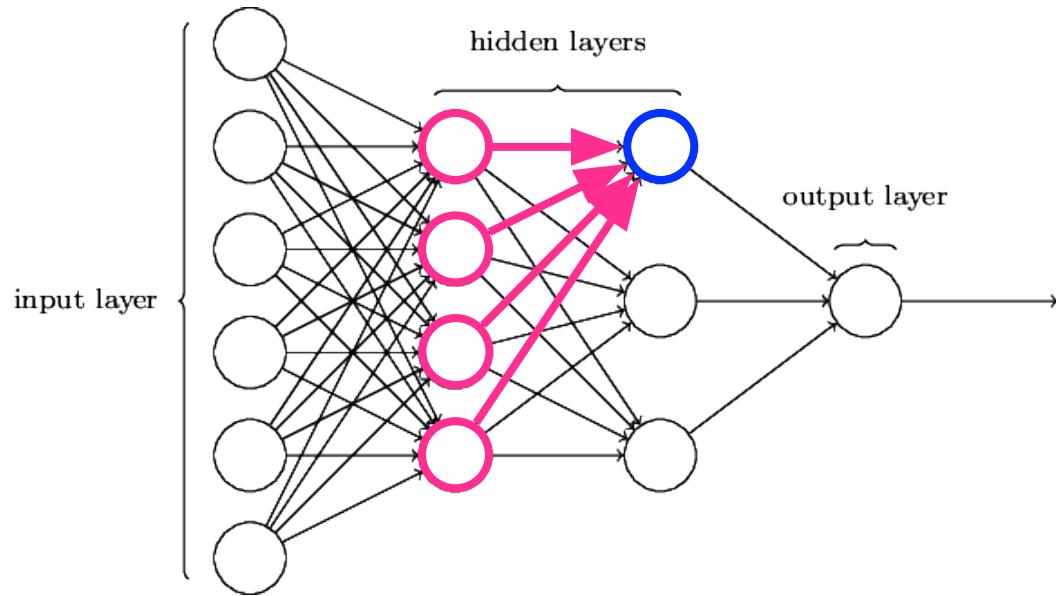
Neural Networks



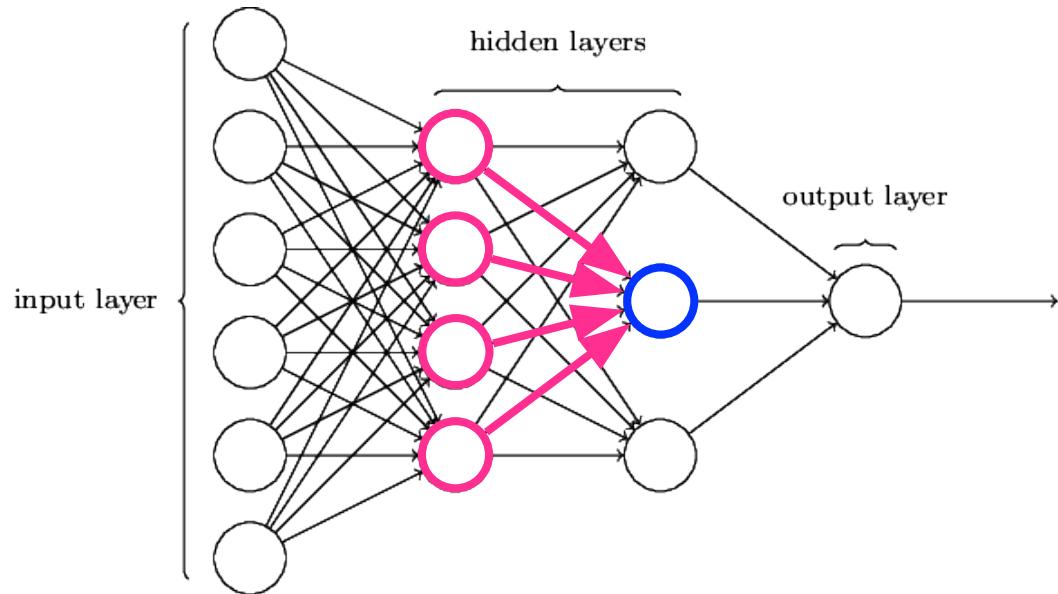
\mathbf{x} : vector \mathbf{W} : matrix \mathbf{h} : vector

$$\mathbf{h} = f_{\text{act}} (\mathbf{W}\mathbf{x} + \mathbf{b})$$

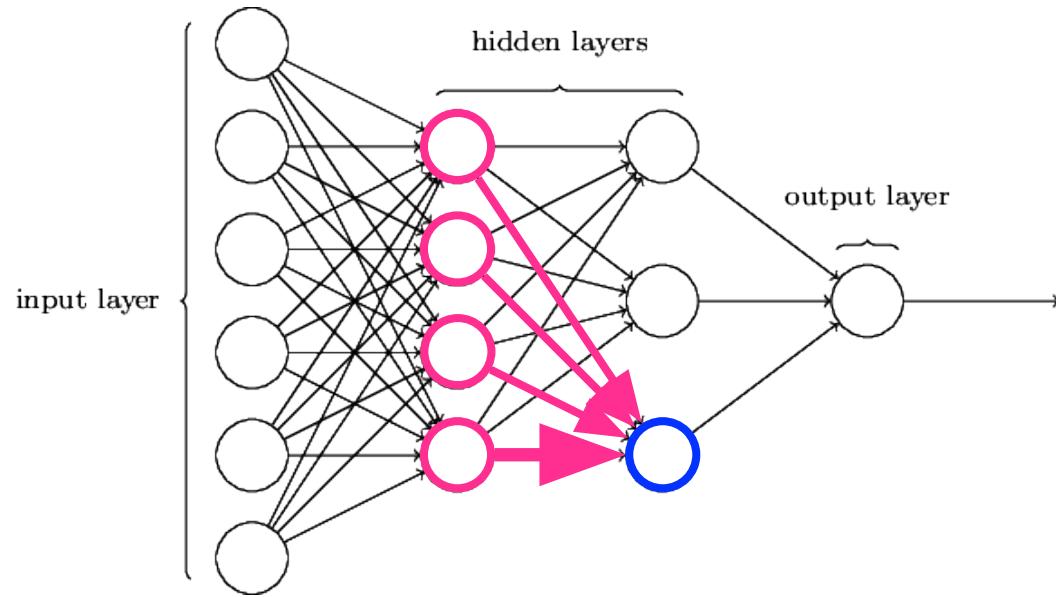
Neural Networks



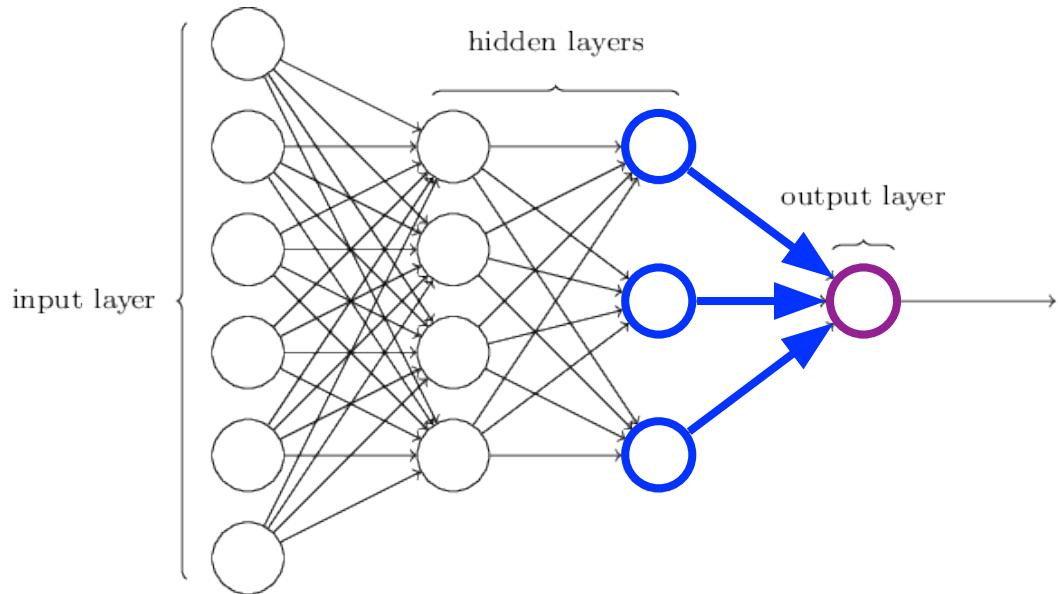
Neural Networks



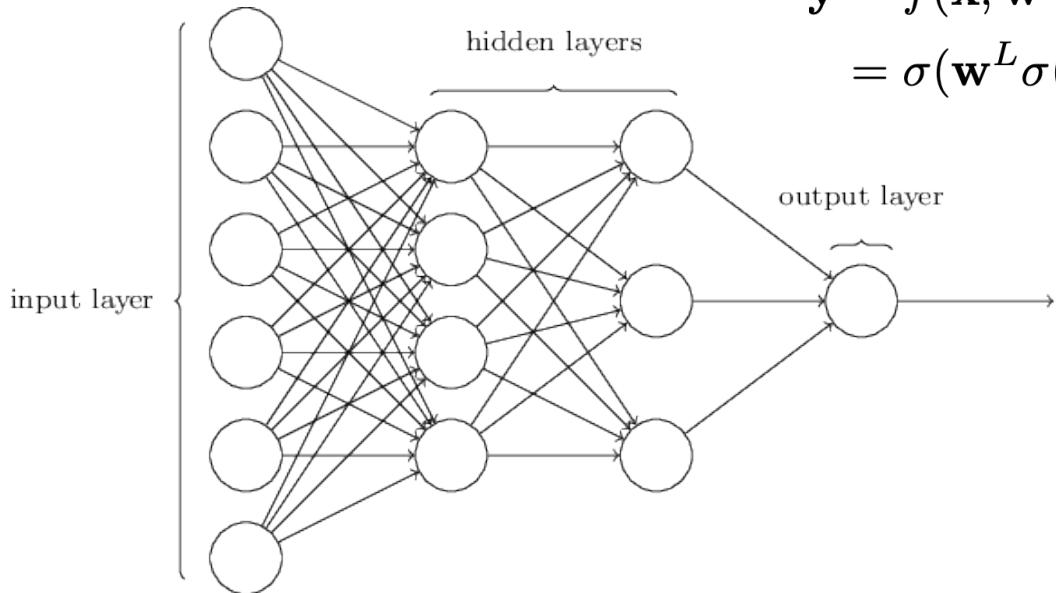
Neural Networks



Neural Networks



Neural Networks

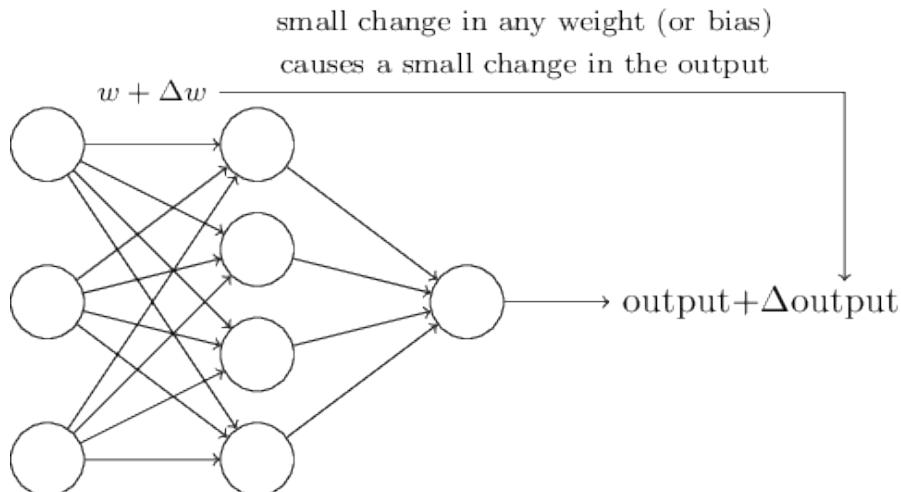


$$\begin{aligned}\hat{\mathbf{y}} &= f(\mathbf{x}; \mathbf{w}^1, \dots, \mathbf{w}^L, \mathbf{b}^1, \dots, \mathbf{b}^L) \\ &= \sigma(\mathbf{w}^L \sigma(\mathbf{w}^{L-1} \dots \sigma(\mathbf{w}^1 \mathbf{x} + \mathbf{b}^1) \dots) + \mathbf{b}^L)\end{aligned}$$

activation function:
sigmoid function

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

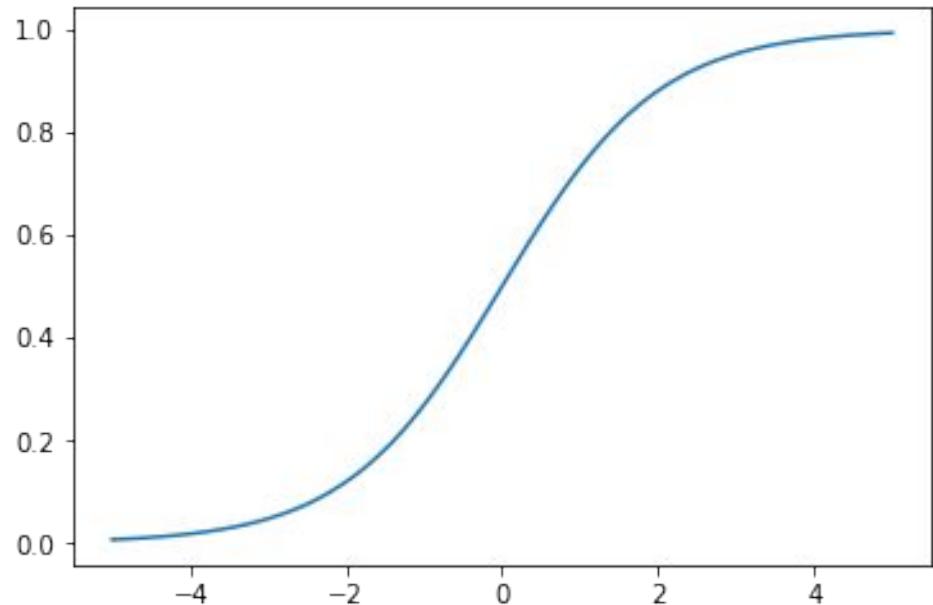
Activation Function



- 네트워크의 가중치를 약간만 변경 했을때 출력도 조금만 변하게 하고 싶다
- 가중치를 조금씩 (반복하여) 변화시켜 더 좋은 출력(output)을 내도록 한다
- 퍼셉트론은 가중치가 변해도 출력이 잘 변하지 않음
- Sigmoid neuron을 사용

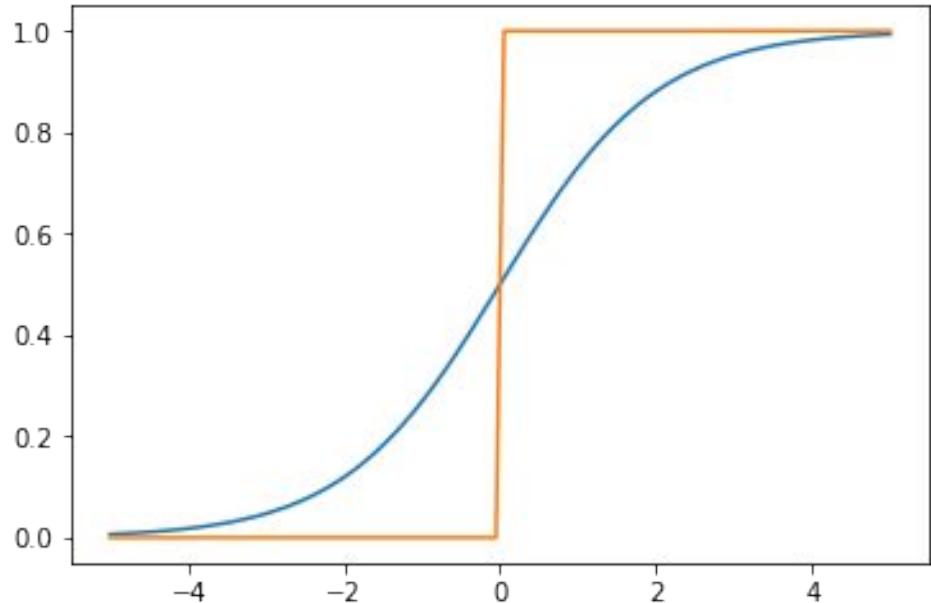
Sigmoid Function

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

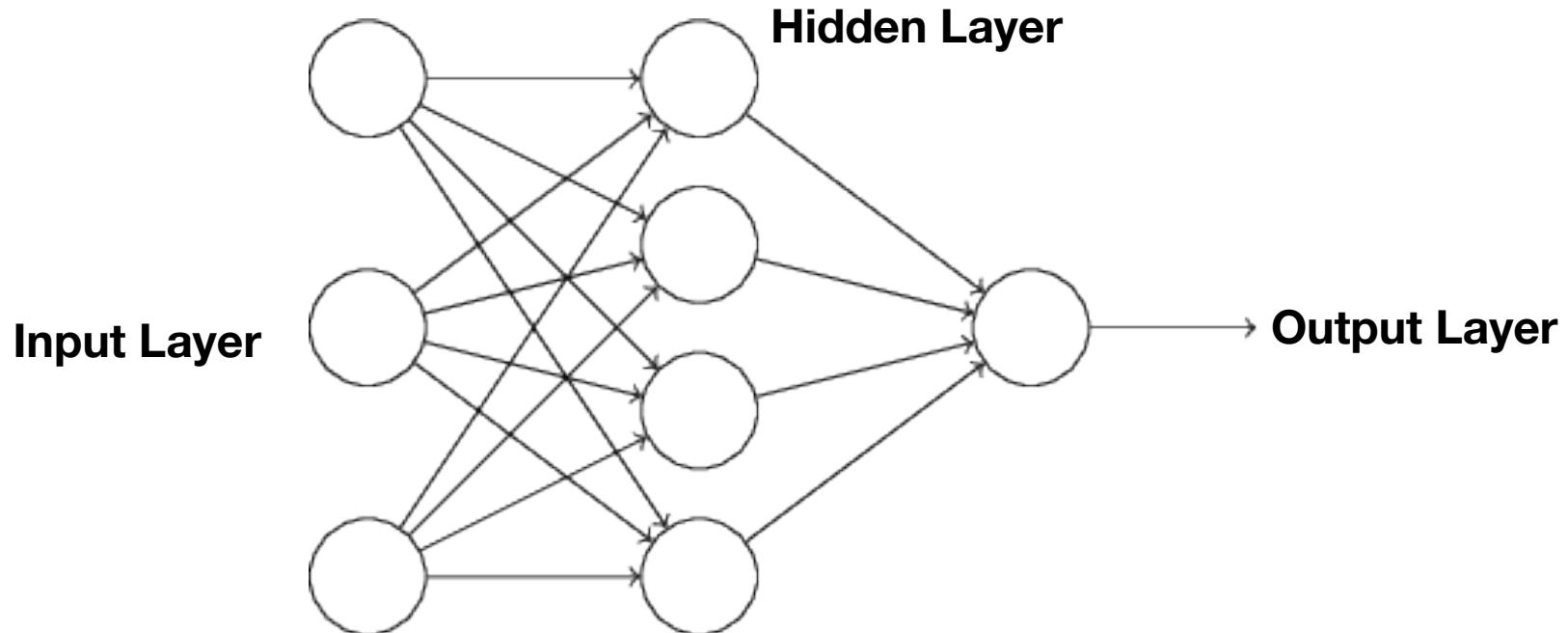


Sigmoid vs Step Function

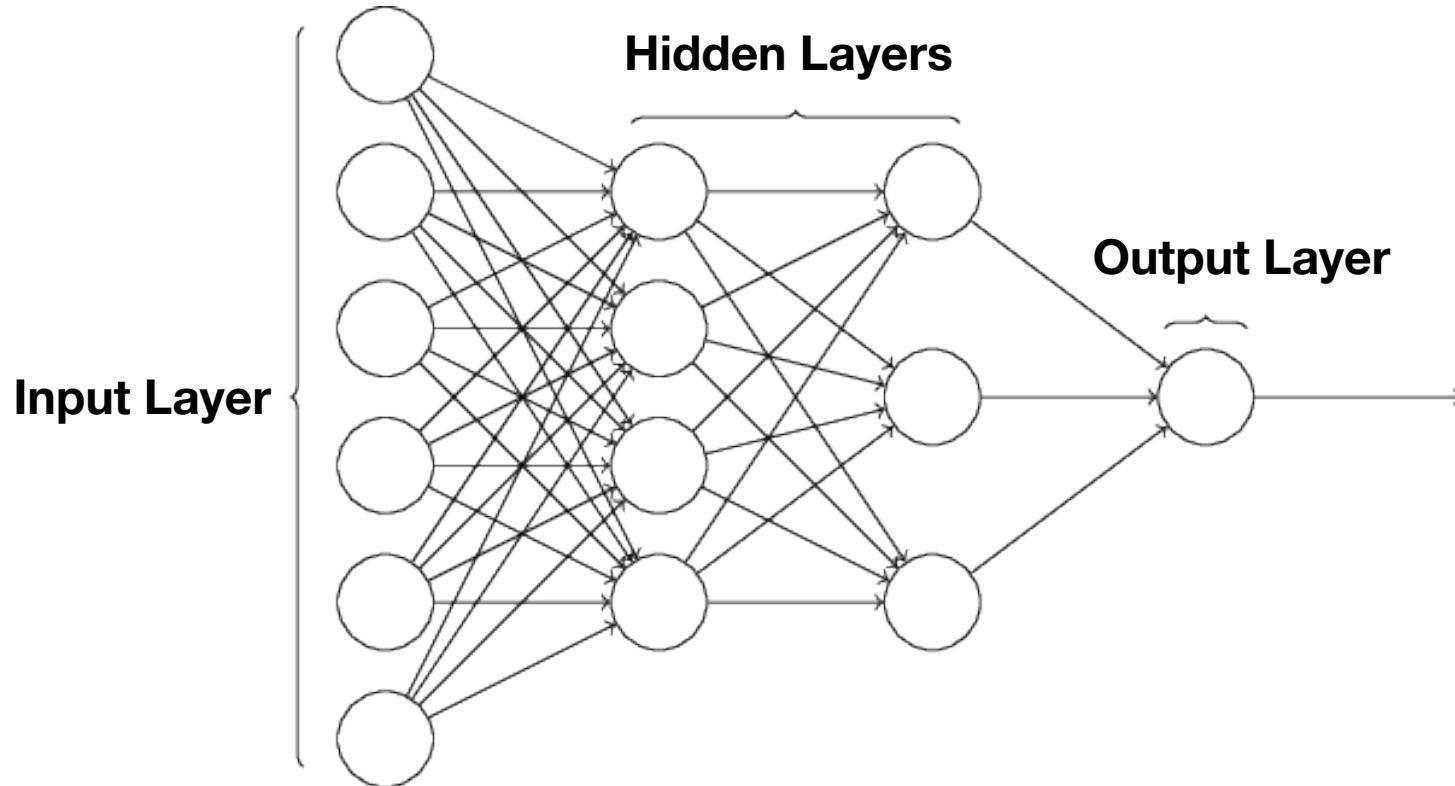
- 비슷한 점
- Sigmoid function은 입력값이 크면 1에 가깝고 작으면 0에 가까워져서 step function과 비슷하다
- 둘 다 $[0, 1]$ 사이의 값을 갖는다
- 차이점
- Sigmoid function은 부드러운 곡선을 갖는 (smoothed) step function이다
- Sigmoid의 매끄러움 때문에 작은 가중치 변화로 작은 출력값의 변화가 생긴다



Structures of Neural Networks



Structures of Neural Networks

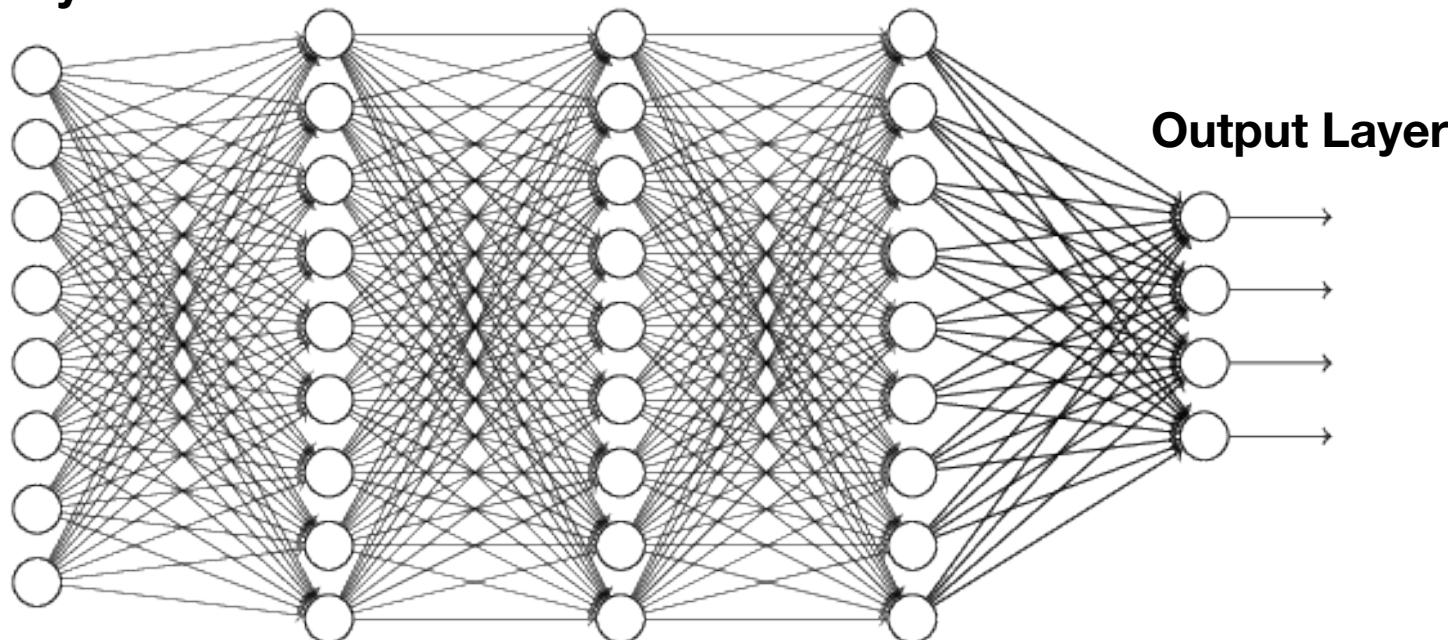


Structures of Neural Networks

Input Layer

Hidden Layers: can be very many

Output Layer



Optimizer

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

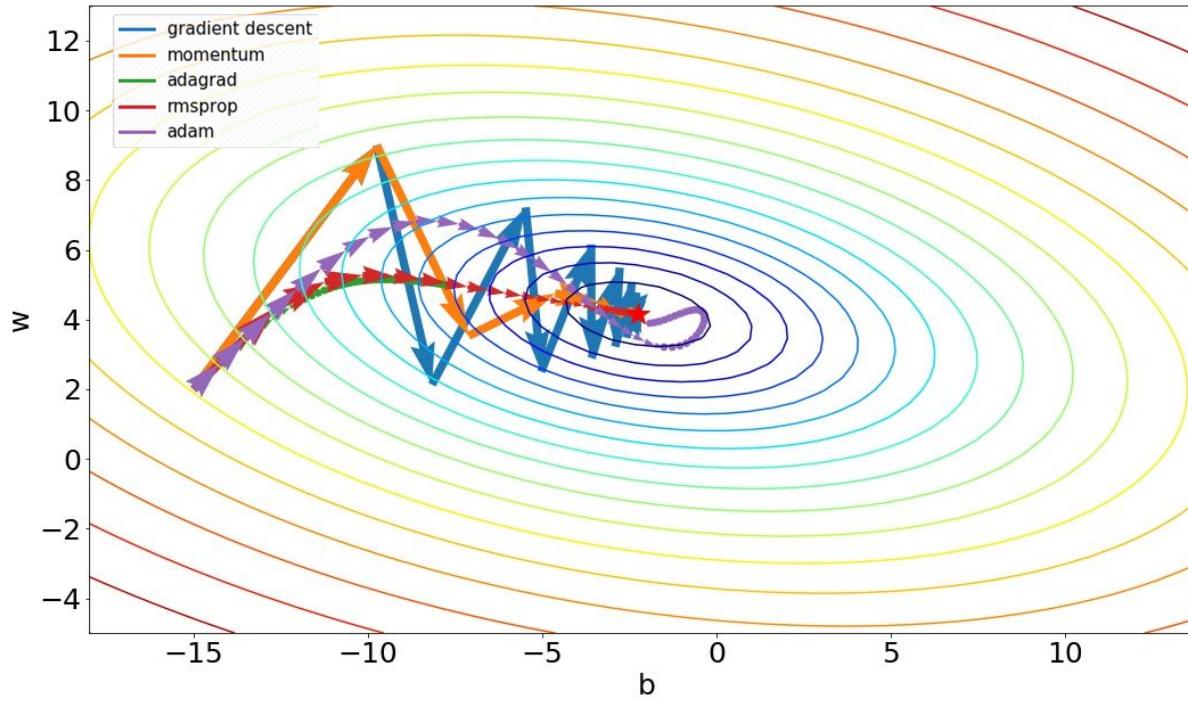
learning rate :

한번에 얼마나 학습할지

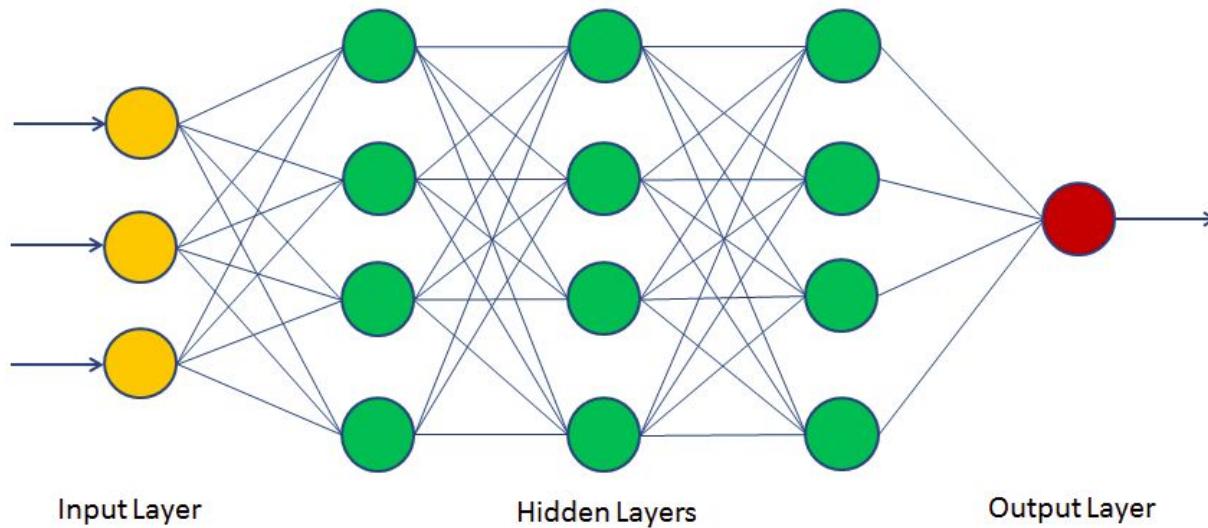
gradient :

어떤 방향으로 학습할지

Optimizer

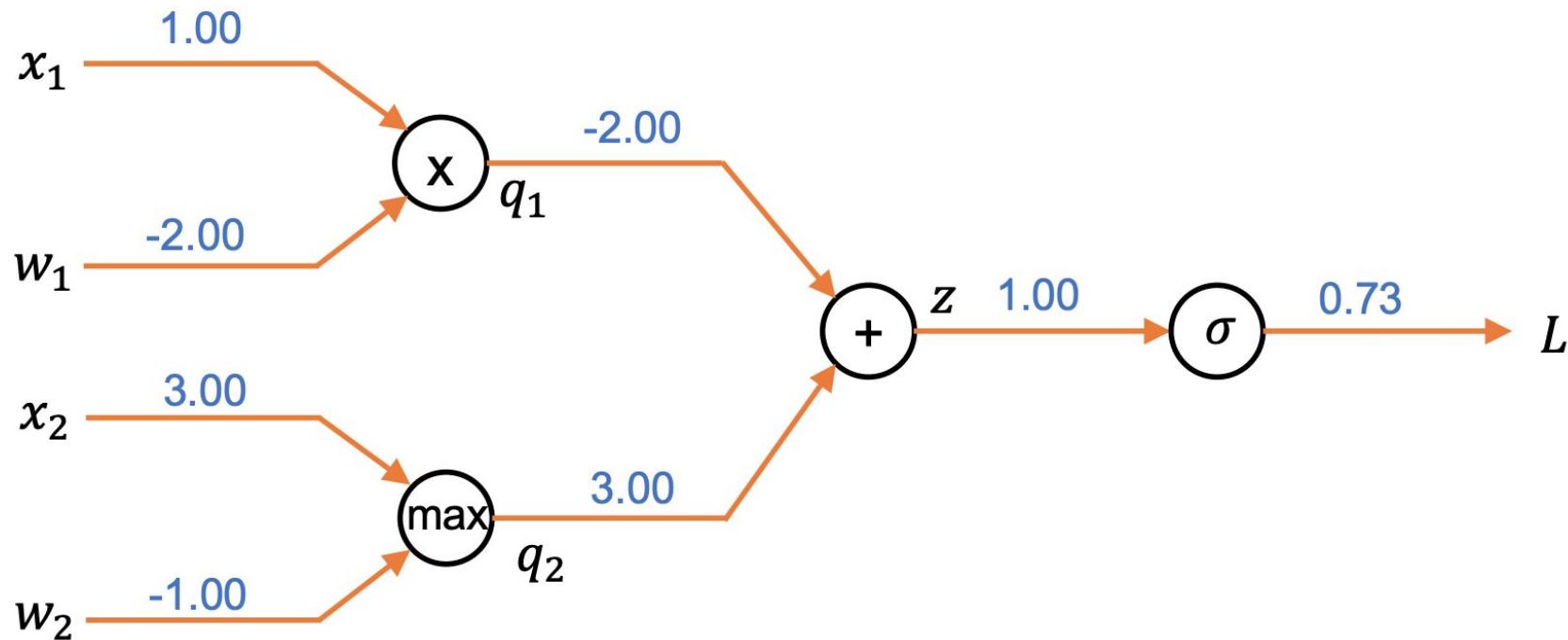


Backpropagation

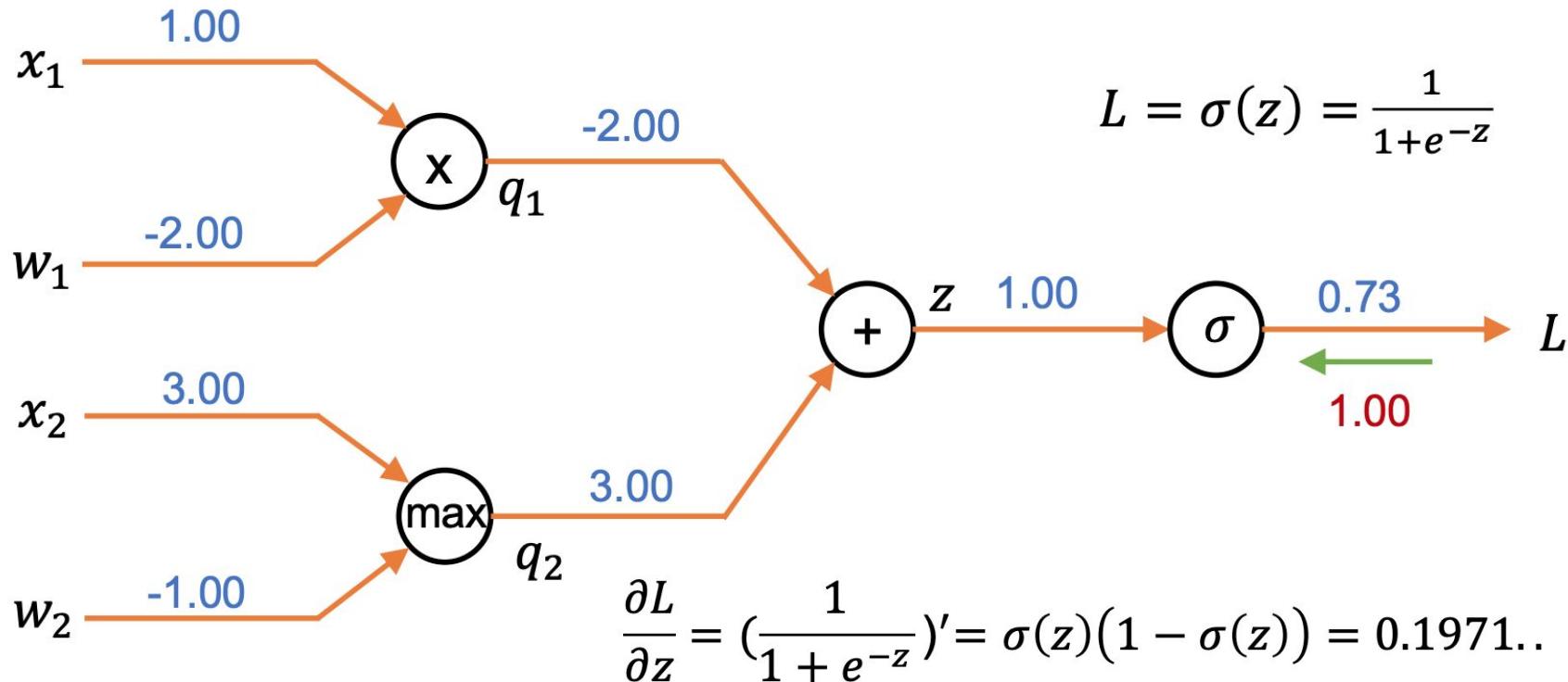


$$\frac{\partial L(\theta)}{\partial W_{ij}^{(1)}} \leftarrow \frac{\partial L(\theta)}{\partial W_{ij}^{(2)}} \leftarrow \frac{\partial L(\theta)}{\partial W_{ij}^{(3)}} \leftarrow \frac{\partial L(\theta)}{\partial W_{ij}^{(4)}}$$

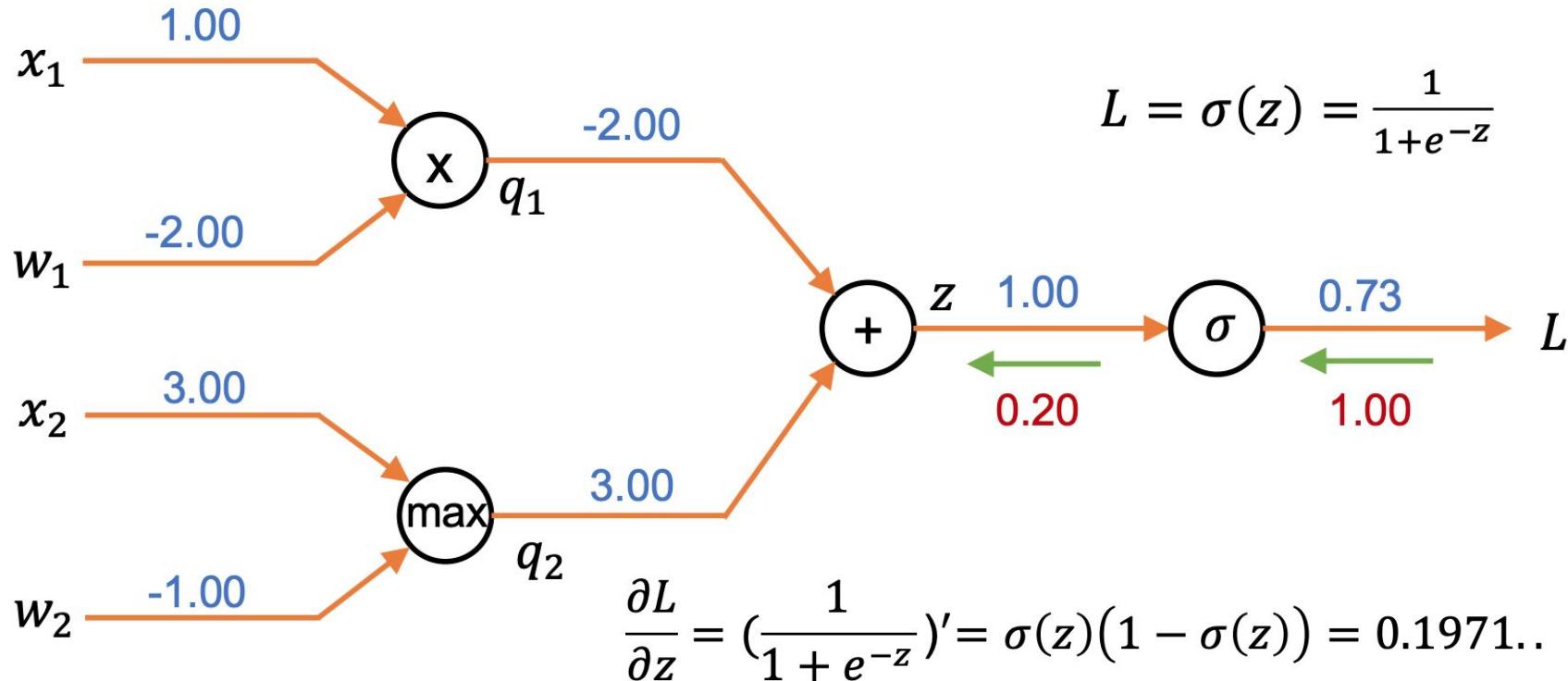
Backpropagation



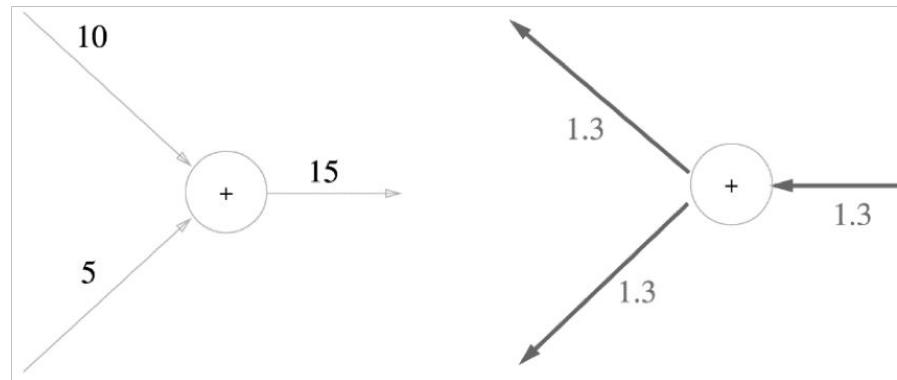
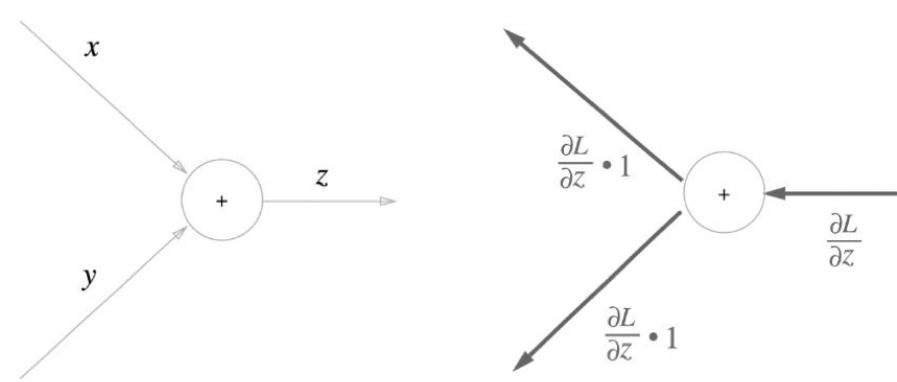
Backpropagation



Backpropagation



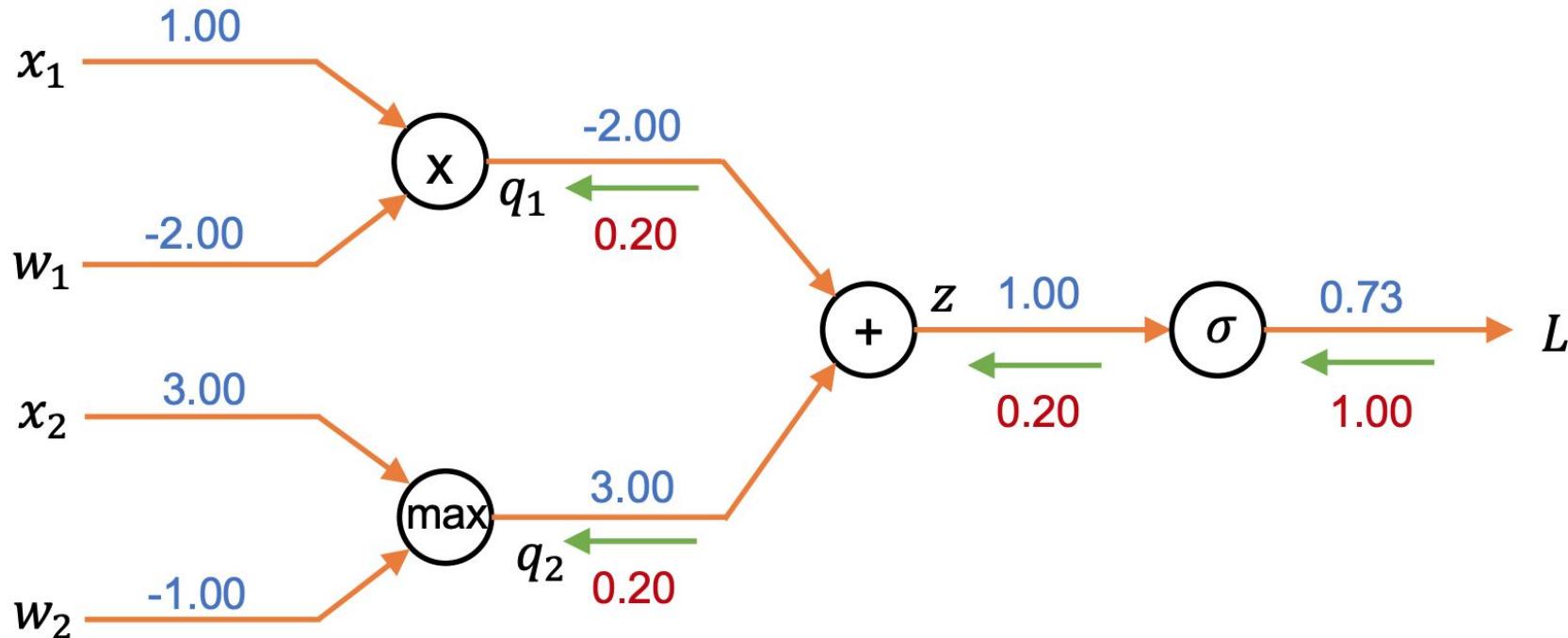
Backpropagation



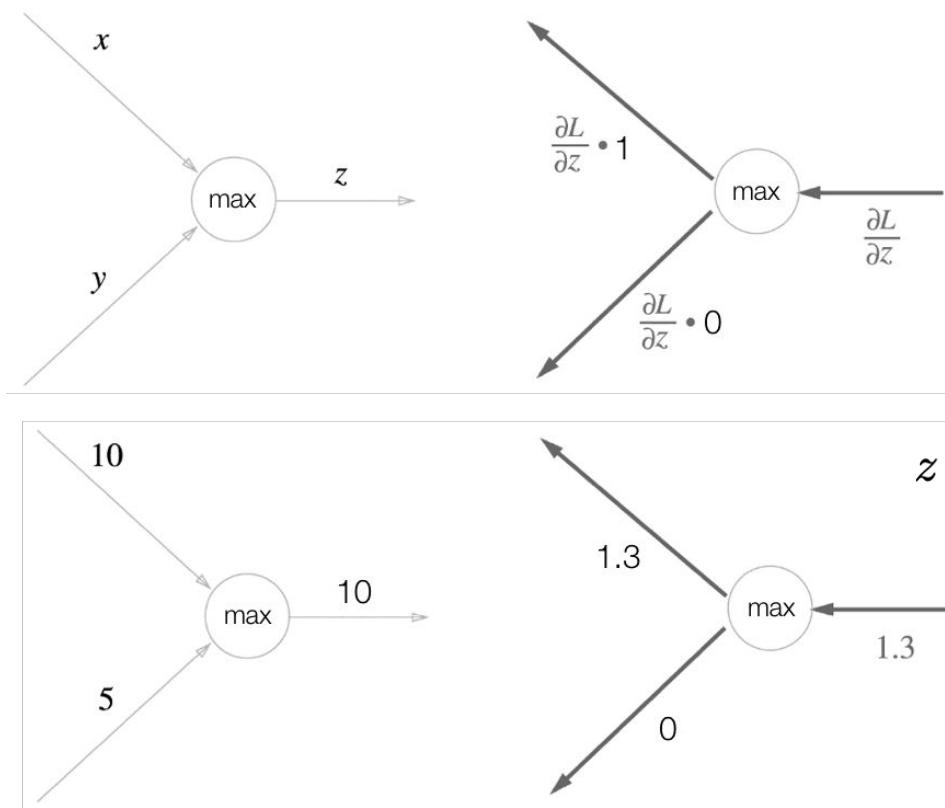
$$z = x + y \Rightarrow \begin{cases} \frac{\partial z}{\partial x} = 1 \\ \frac{\partial z}{\partial y} = 1 \end{cases}$$

gradient distributor

Backpropagation



Backpropagation

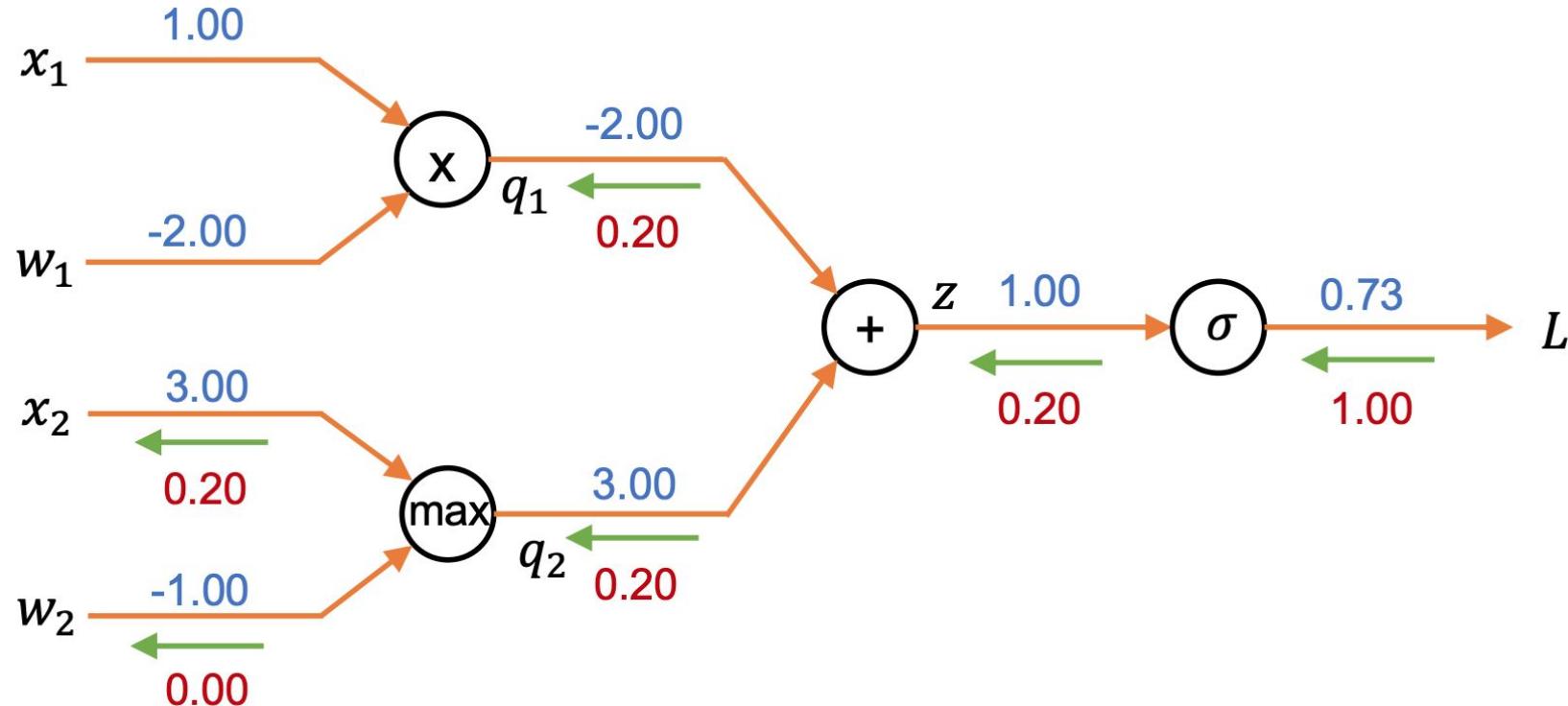


gradient router

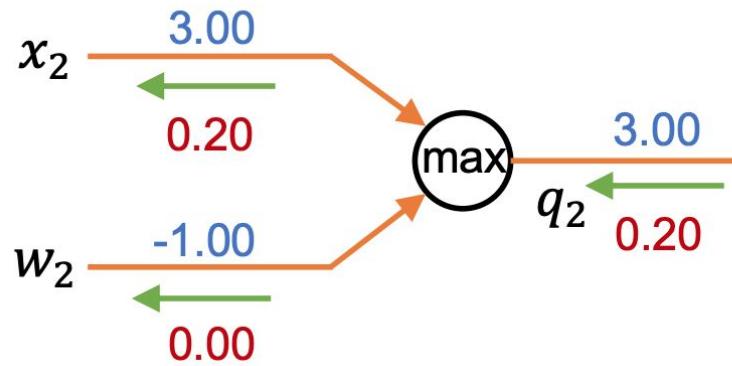
$$z = \max(x, y) \Rightarrow$$

$$\begin{cases} \frac{\partial z}{\partial x} = 1 & \text{if } x > y \\ \frac{\partial z}{\partial y} = 0 & \text{if } x < y \end{cases}$$

Backpropagation



Backpropagation



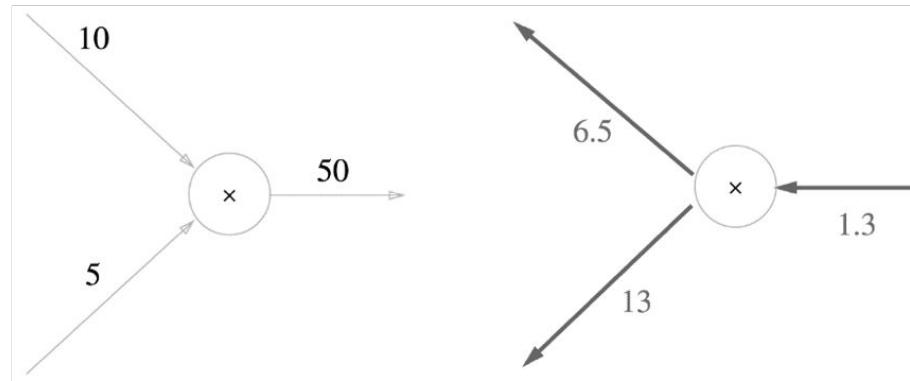
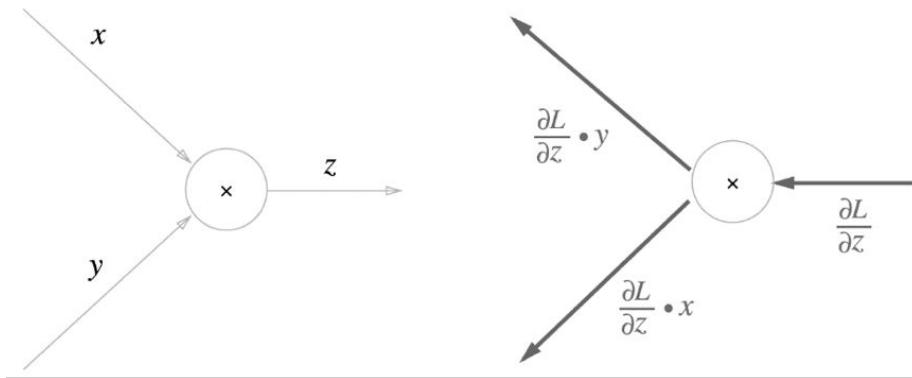
$$\frac{\partial q_2}{\partial x_2} = \mathbf{1}(x_2 > w_2)$$

$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial q_2} \cdot \frac{\partial q_2}{\partial x_2} = 0.20 * 1.00 = 0.20$$

$$\frac{\partial q_2}{\partial w_2} = \mathbf{1}(w_2 > x_2)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial q_2} \cdot \frac{\partial q_2}{\partial w_2} = 0.20 * 0.00 = 0.00$$

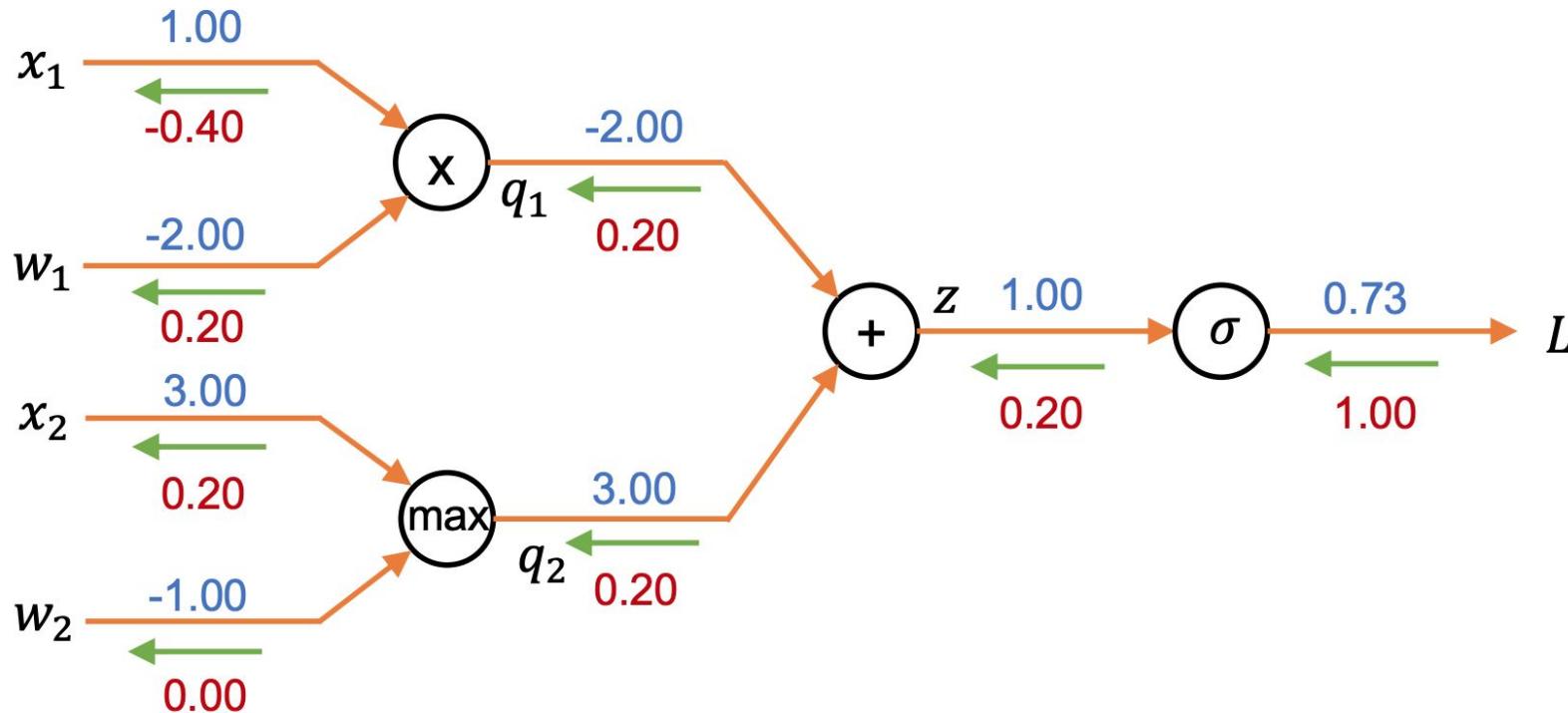
Backpropagation



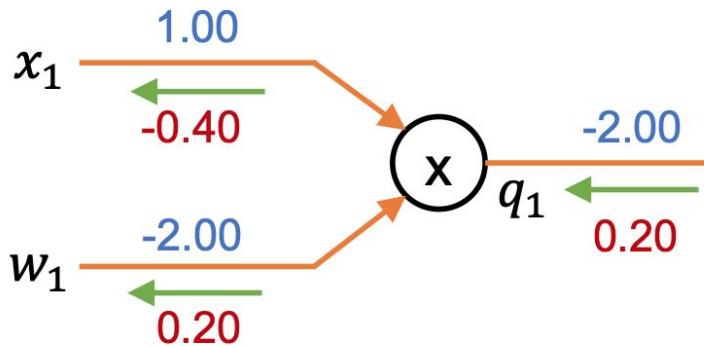
$$z = xy \Rightarrow \begin{cases} \frac{\partial z}{\partial x} = y \\ \frac{\partial z}{\partial y} = x \end{cases}$$

gradient switcher

Backpropagation



Backpropagation



$$\frac{\partial q_1}{\partial x_1} = w_1$$

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial q_1} \cdot \frac{\partial q_1}{\partial x_1} = 0.20 * (-2.00) = -0.40$$

$$\frac{\partial q_1}{\partial w_1} = x_1$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial q_1} \cdot \frac{\partial q_1}{\partial w_1} = 0.20 * 1.00 = 0.20$$

Backpropagation

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

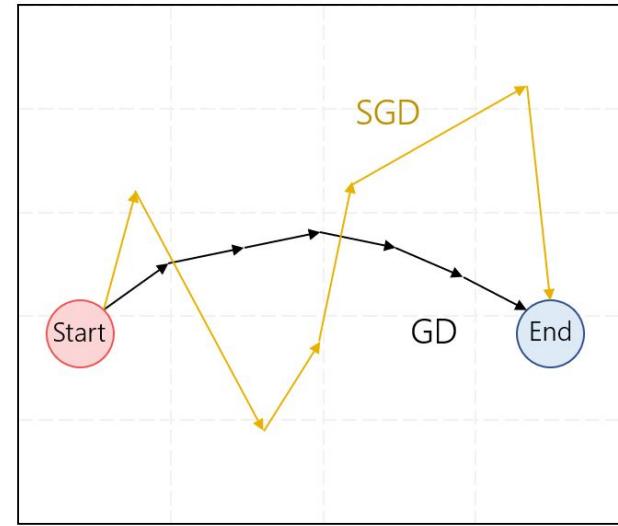
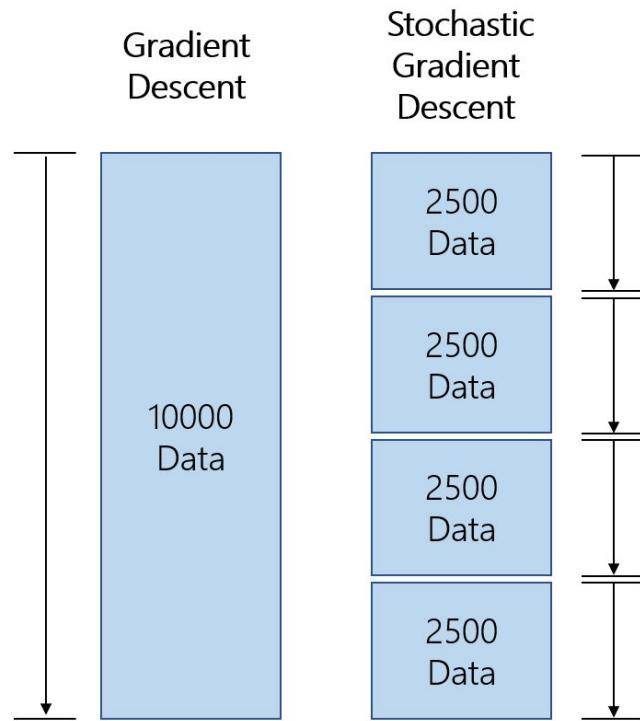
learning rate :

한번에 얼마나 학습할지

gradient :

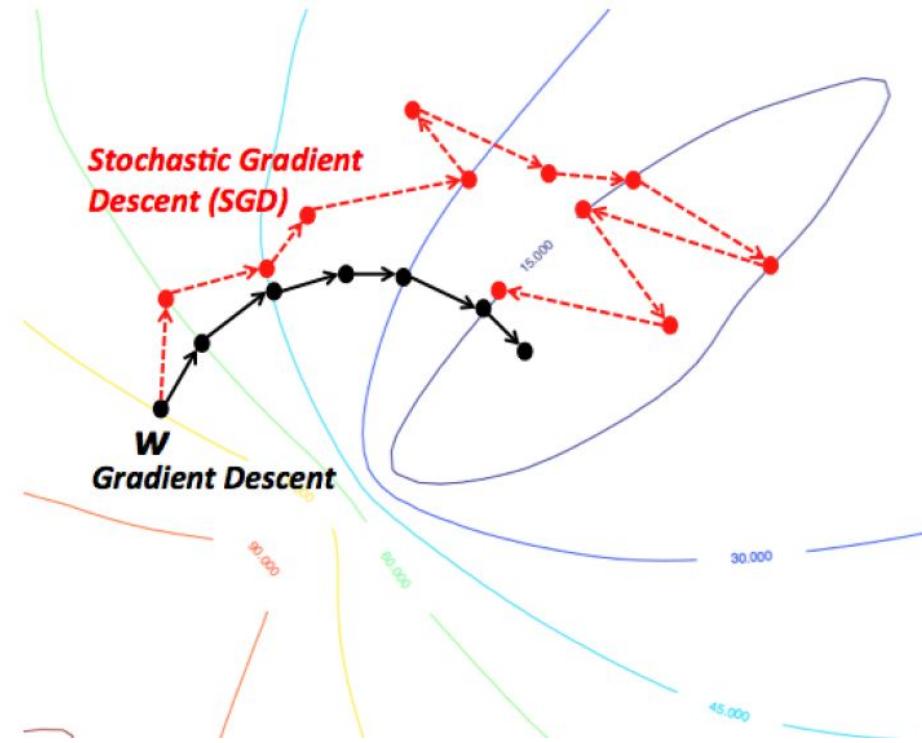
어떤 방향으로 학습할지

Optimizer

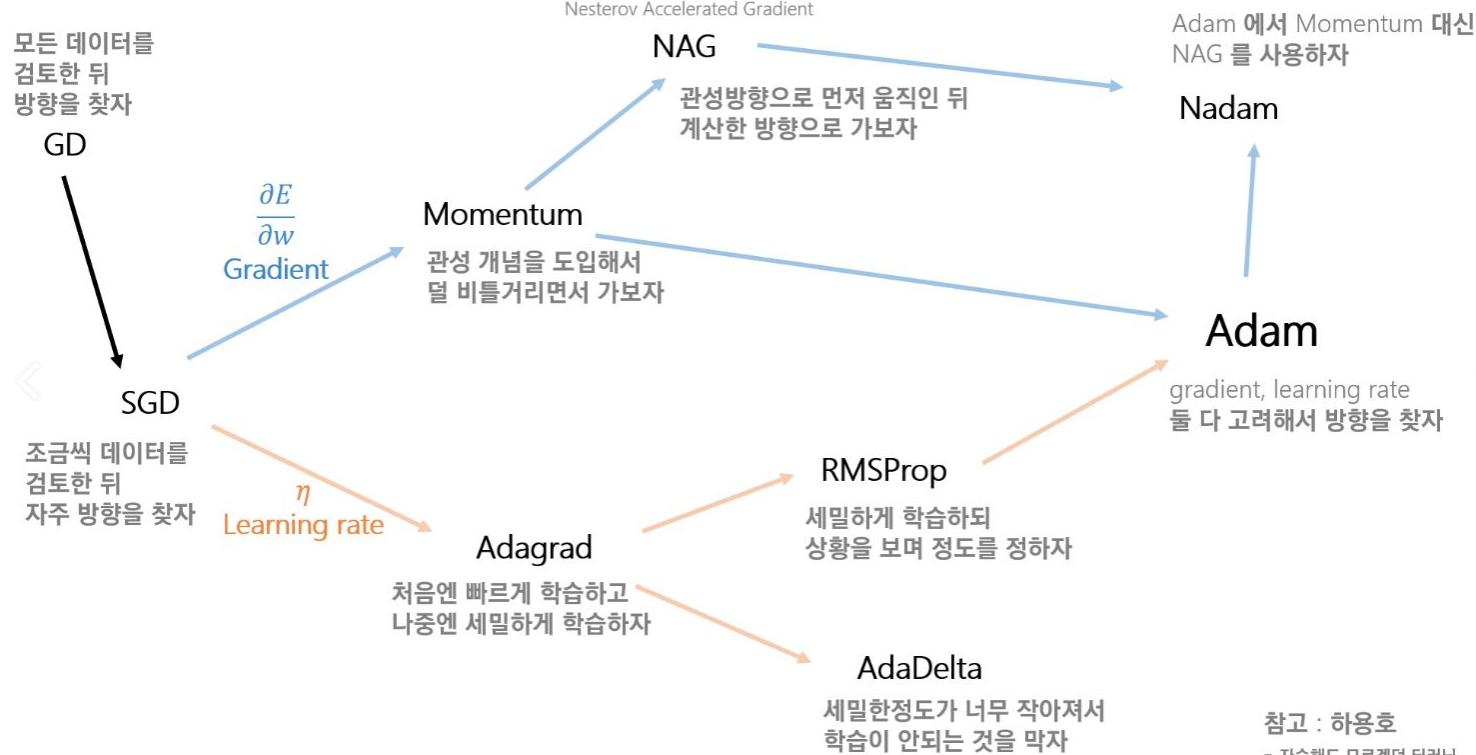


더 비틀비틀 가는 것 같지만
기존 보다 더 빠르게 학습합니다

Optimizer



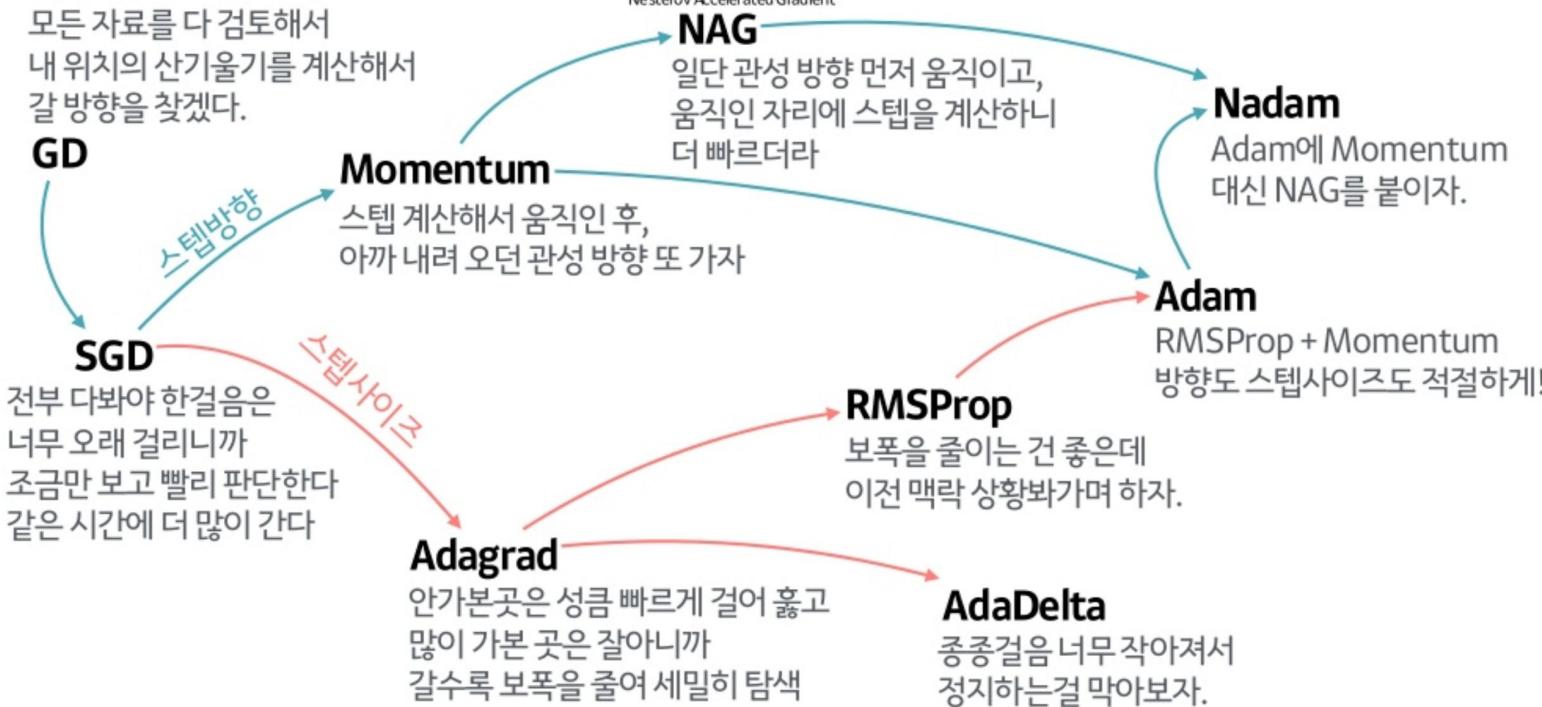
Optimizer



참고 : 하용호

- 자습해도 모르겠던 딥러닝, 머리속에 인스를 시켜드립니다.

Optimizer



Training Process - MNIST

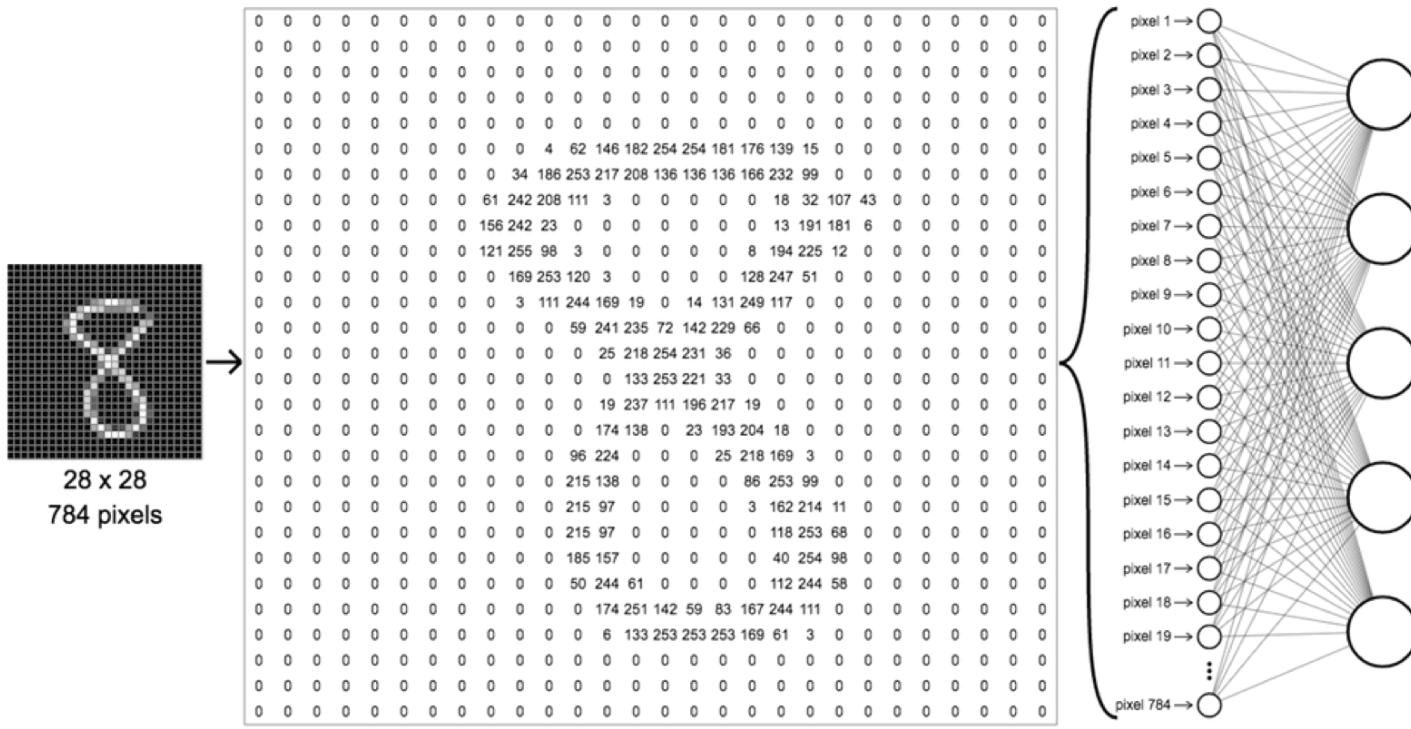
- MNIST dataset
- 0 ~ 9의 손 글씨 데이터
- 28 x 28 grayscale 이미지
- 6만장 학습 데이터, 1만장 테스트 데이터



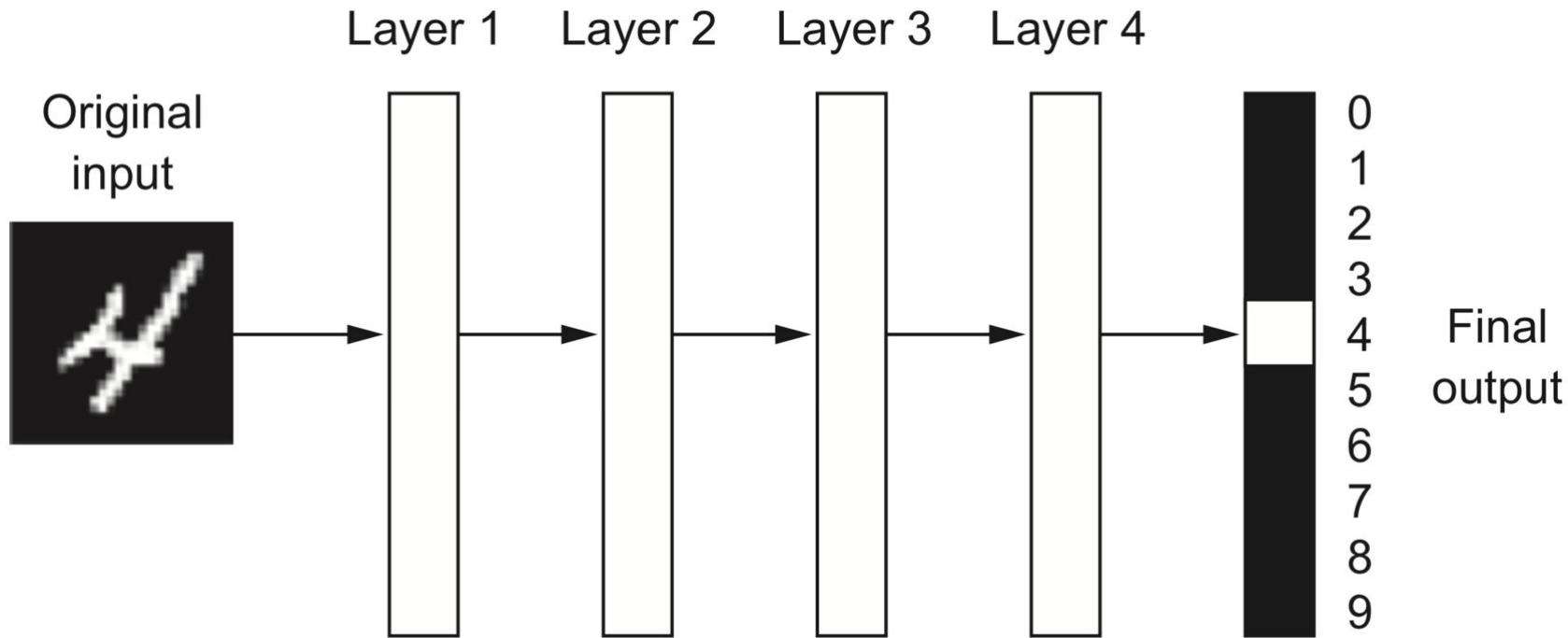
Training Process - MNIST

- 입력 (x)
 - $28 \times 28 = 784$ dimensional vector
- 출력 ($y = y(x)$)
 - 10 dimensional vector
 - ex) 숫자 6 : $y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T$

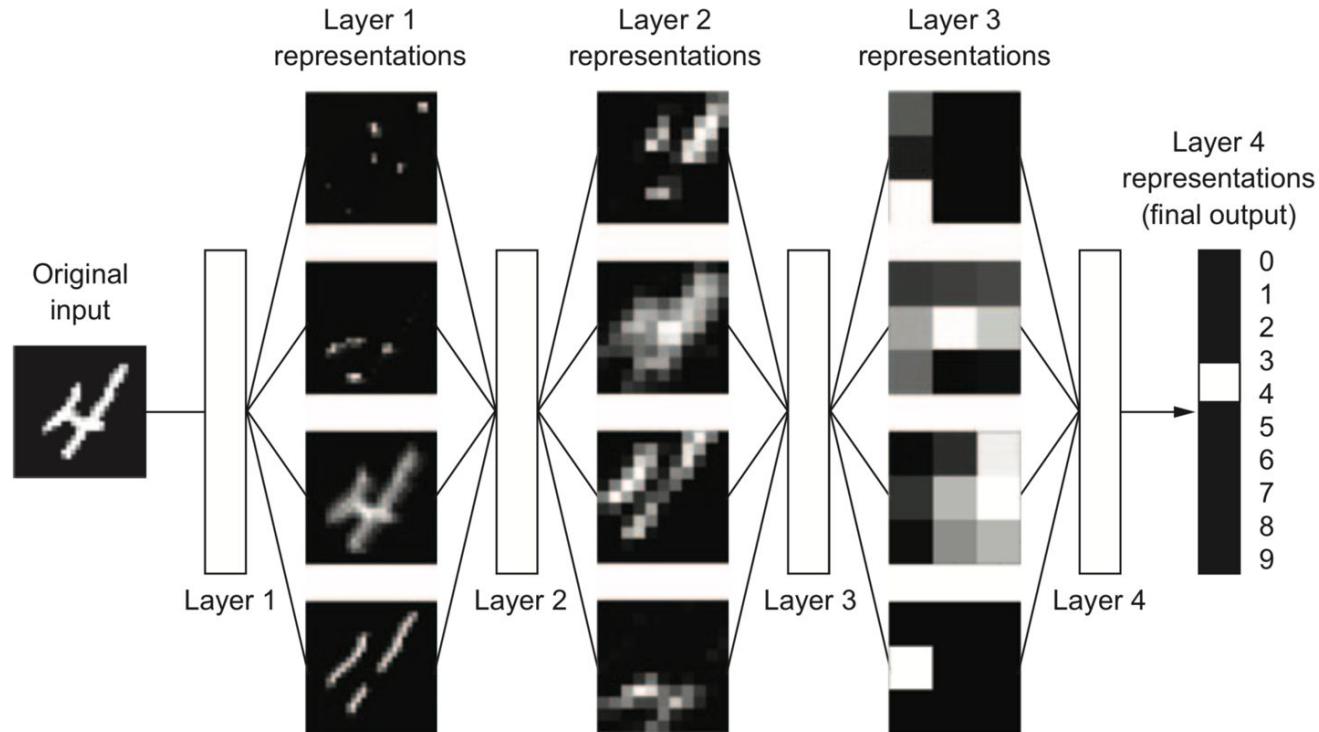
Training Process - MNIST



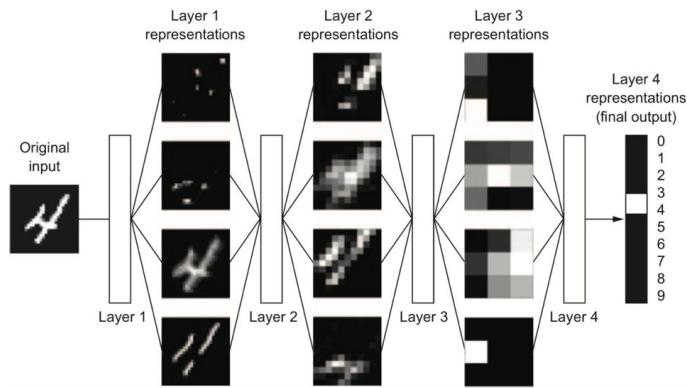
Training Process - MNIST



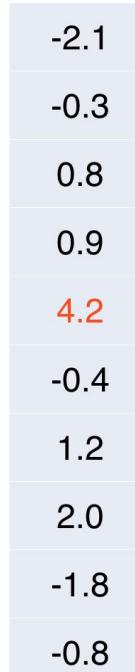
Training Process - MNIST



Training Process - MNIST



logits: z



$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}},$$

$$\text{where } z_i = \sum_j W_{ij}x_j + b_i$$



softmax: a



Training Process - MNIST

prediction	label
0.0015	0
0.0088	0
0.026	0
0.029	0
0.79	1
0.008	0
0.039	0
0.088	0
0.002	0
0.0053	0

Mean Squared Error (MSE)

$$\mathcal{L} \equiv \frac{1}{2N} \sum_i \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|^2 = \frac{1}{2N} \sum_i \sum_k (\hat{y}_{ik} - y_{ik})^2$$

Cross Entropy Loss

$$\mathcal{L} \equiv -\frac{1}{N} \sum_i \mathbf{y}_i \log \hat{\mathbf{y}}_i = -\frac{1}{N} \sum_i \sum_k y_{ik} \log \hat{y}_{ik}$$

label: y

prediction: \hat{y}

dimension: k



Training Process - MNIST

- 주어진 loss function \mathcal{L} 을 **최소화** 하는 parameters Θ 를 찾는 것

$$\arg \min_{\Theta} \mathcal{L} \quad \Theta = \{\mathbf{w}_1, \dots, \mathbf{b}_1, \dots\}$$

- 간단하게 말하면 굉장히 복잡한 함수를 잘 **fitting** 하는 것
- 학습의 목적 (머신러닝의 목적)
 - 주어진 데이터를 이용하여 loss function을 최소화 함과 동시에 새로운 데이터들에 대해서도 잘 맞추도록 (**generalization**) 하는 것 (avoiding overfitting)

Training Process - MNIST

- 주어진 loss function \mathcal{L} 을 **최소화** 하는 parameters Θ 를 찾는 것

$$\arg \min_{\Theta} \mathcal{L} \quad \Theta = \{\mathbf{w}_1, \dots, \mathbf{b}_1, \dots\}$$

- 간단하게 말하면 굉장히 복잡한 함수를 잘 **fitting** 하는 것

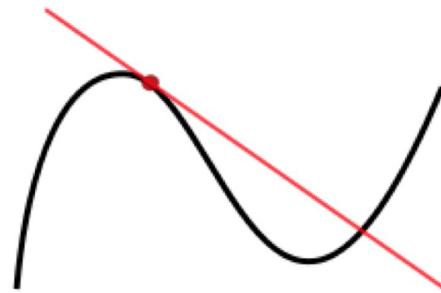
How to?

현재 위치에서 **가장 가파른** (내리막) **방향**으로 **조금씩** 이동하자

Training Process - MNIST

- 미분을 하면 된다

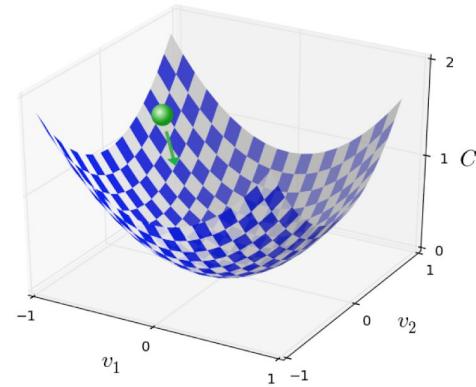
$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$



- 방향이 두 (또는 여러) 방향 : 편미분

$$\frac{\partial f(x, y)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h, y) - f(x, y)}{h}$$

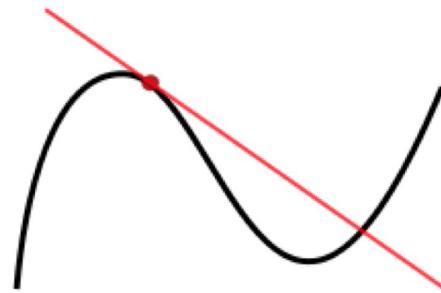
$$\frac{\partial f(x, y)}{\partial y} = \lim_{h \rightarrow 0} \frac{f(x, y + h) - f(x, y)}{h}$$



Training Process - MNIST

- 미분을 하면 된다

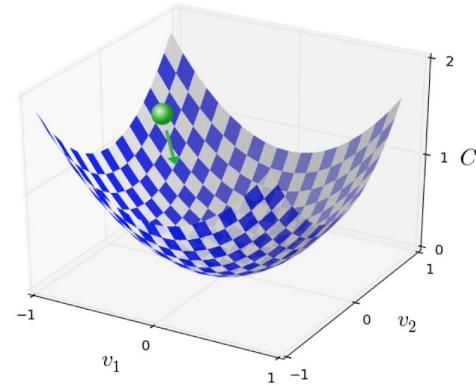
$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$



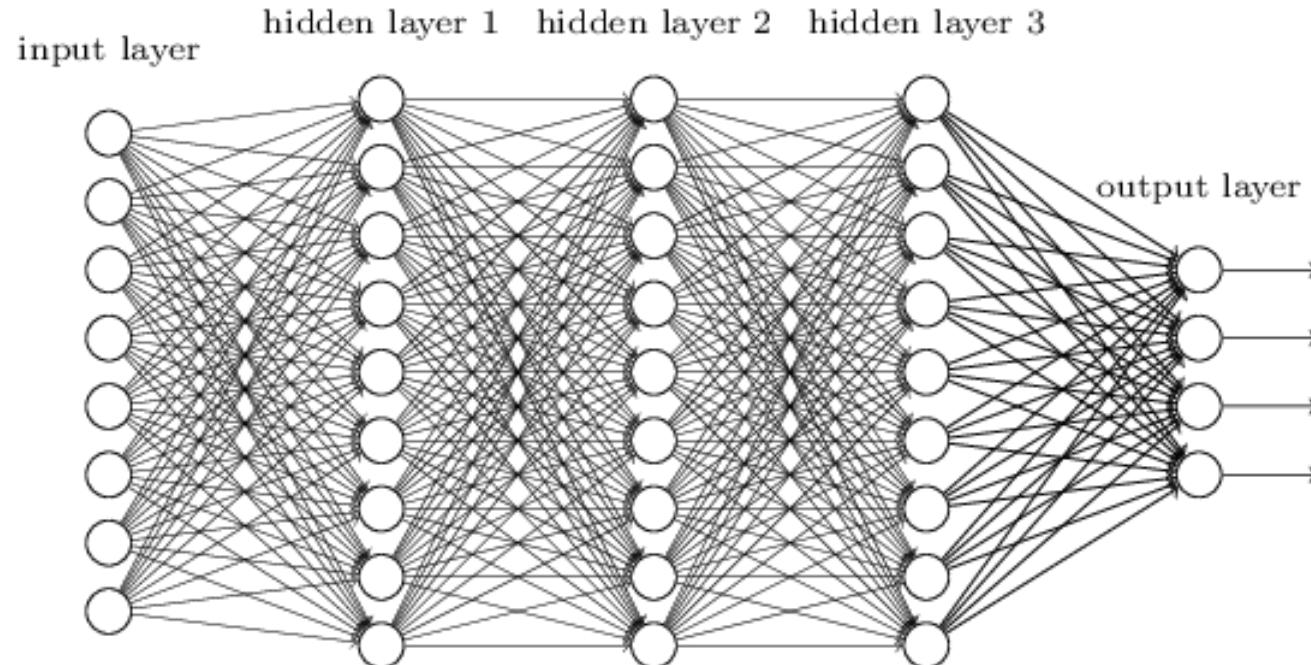
- 방향이 두 (또는 여러) 방향 : 편미분

$$\frac{\partial f(x, y)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h, y) - f(x, y)}{h}$$

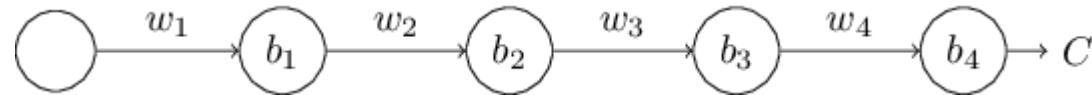
$$\frac{\partial f(x, y)}{\partial y} = \lim_{h \rightarrow 0} \frac{f(x, y + h) - f(x, y)}{h}$$



Vanishing Gradient

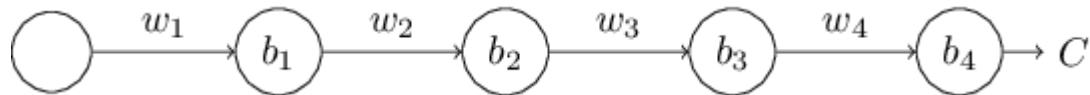


Vanishing Gradient



$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

Vanishing Gradient

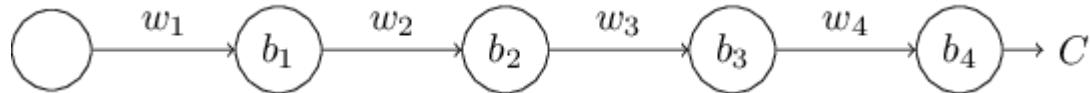


$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

Chain rule

$$\frac{\partial C}{\partial b_1} = \left(\frac{\partial C}{\partial a_4} \frac{\partial a_4}{\partial z_4} \right) \left(\frac{\partial z_4}{\partial a_3} \frac{\partial a_3}{\partial z_3} \right) \left(\frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} \right) \left(\frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial b_1} \right)$$

Vanishing Gradient



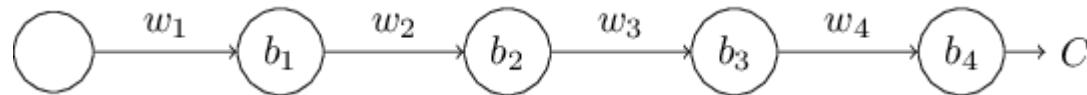
Chain rule

$$\frac{\partial C}{\partial b_1} = \left(\frac{\partial C}{\partial a_4} \frac{\partial a_4}{\partial z_4} \right) \left(\frac{\partial z_4}{\partial a_3} \frac{\partial a_3}{\partial z_3} \right) \left(\frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} \right) \left(\frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial b_1} \right)$$

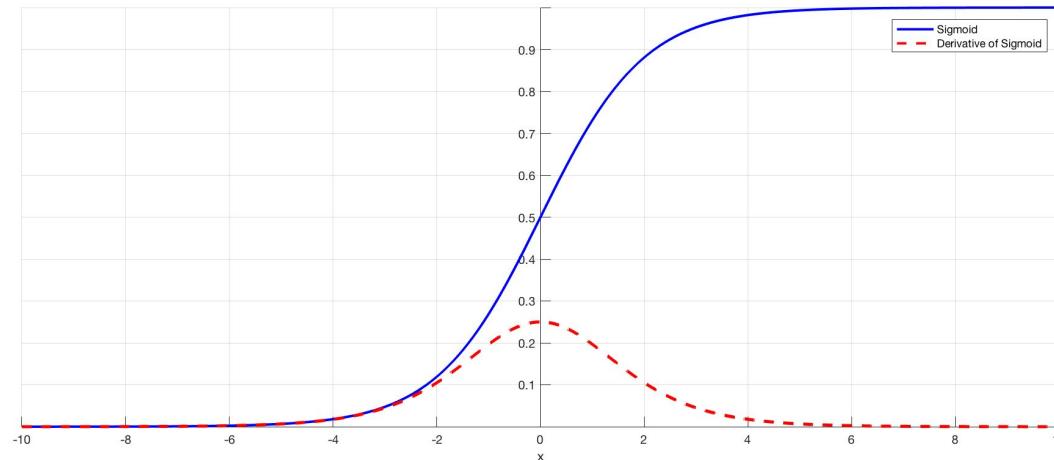
$$a_l = \sigma(z_l) \Rightarrow \frac{\partial a_l}{\partial z_l} = \sigma'(z_l) \quad z_l = w_l a_{l-1} + b_l \Rightarrow \frac{\partial z_l}{\partial a_{l-1}} = w_l$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \sigma'(z_4) \times w_4 \sigma'(z_3) \times w_3 \sigma'(z_2) \times w_2 \sigma'(z_1)$$

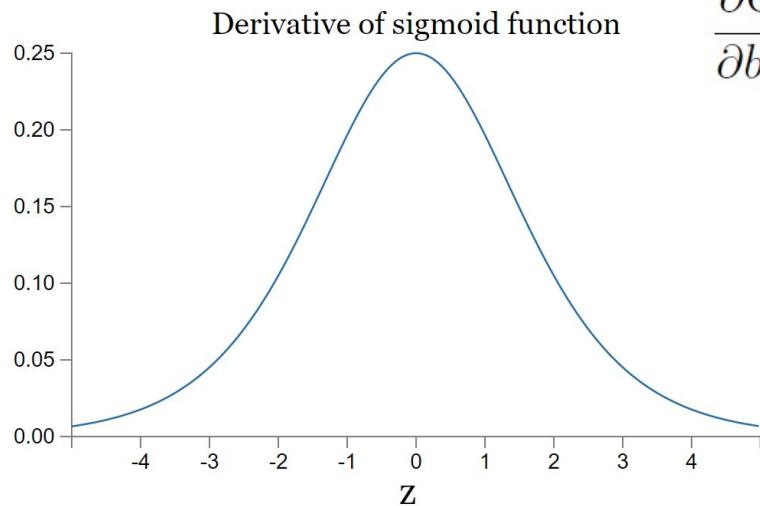
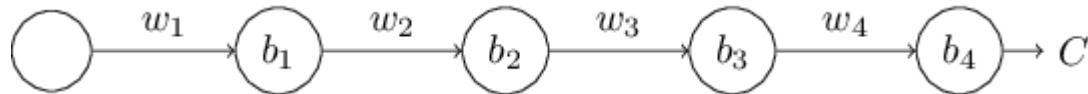
Vanishing Gradient



sigmoid function



Vanishing Gradient



$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \sigma'(z_4) \times w_4 \sigma'(z_3) \times w_3 \sigma'(z_2) \times w_2 \sigma'(z_1)$$

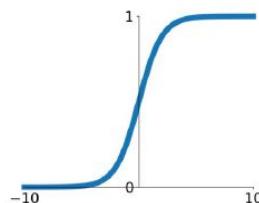
generally $|w_l| < 1$
 $\therefore |w_l \sigma'(z_l)| < 1/4$

아래쪽 layer는
학습이 잘 안됨

Activation Function

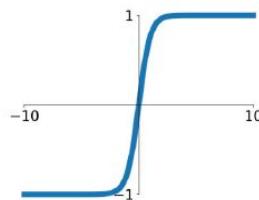
Sigmoid

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



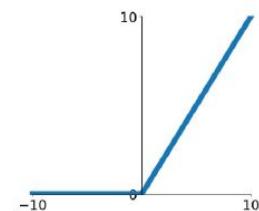
tanh

$$\tanh(x)$$



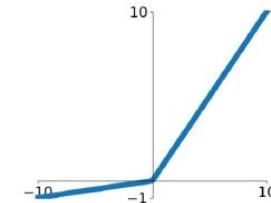
ReLU

$$\max(0, x)$$



Leaky ReLU

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

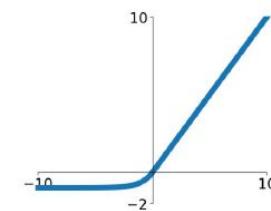


Maxout

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

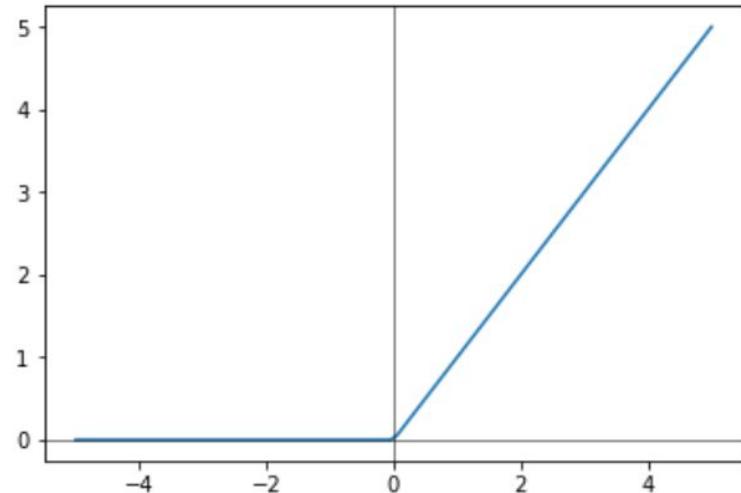
ELU

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



Rectified Linear Unit (ReLU)

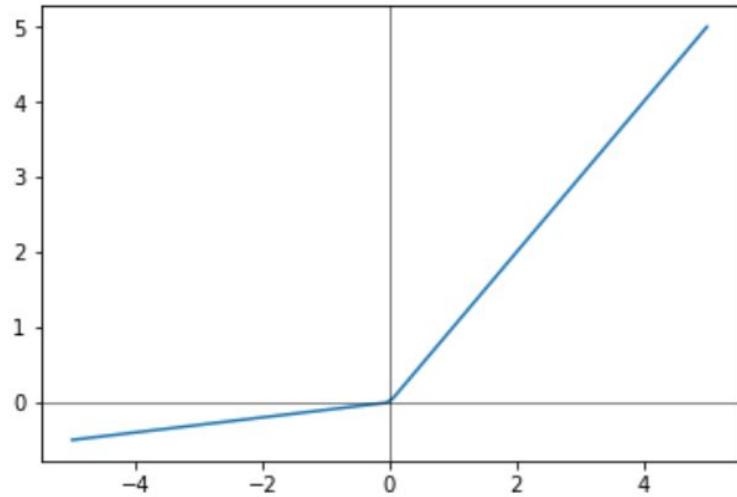
1. Gradient가 죽지 않는다
2. 계산량이 적다 (exponential에 비해)
3. 학습이 빠르게 수렴한다
 1. sigmoid, tanh에 비해 6배 정도 (실험적으로)
4. 실제로는 sigmoid보다 ReLU가 더 생물학적 뉴런에 더 가깝다



$$f_{\text{act}}(x) = \max(x, 0)$$

Leaky ReLU

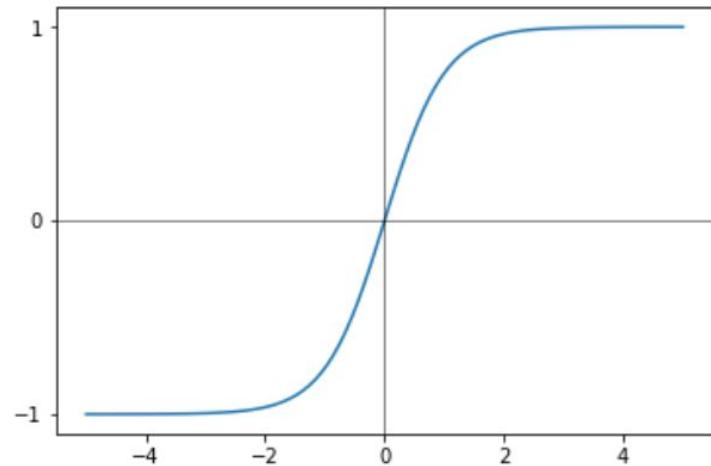
1. Gradient가 죽지 않는다
2. 계산량이 적다 (exponential에 비해)
3. 학습이 빠르게 수렴한다
 1. sigmoid, tanh에 비해 6배 정도 (실험적으로)
4. 노드가 죽지 않는다 (output != 0)



$$f_{\text{act}}(x) = \max(x, 0.01x)$$

Tanh Activation Function

1. 출력값 범위 : [-1, 1]
2. 양끝의 뉴런들은 미분값(gradient)이 0에 가깝다
 - vanishing gradient 원인
3. 출력값이 zero-centered (해결)



$$f_{\text{act}}(x) = \tanh(x)$$

Activation Function

- ReLU를 쓰자
 - Learning rate를 잘 조절하자
- Leaky ReLU / Maxout / ELU 등을 시도해보자
- Sigmoid / tanh 는 쓰지 말자

Preparing Datasets

Idea #1: Choose hyperparameters that work best on the data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train

test

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

train

validation

test

Preparing Datasets

Your Dataset

Idea #4: Cross-Validation: Split data into **folds**,
try each fold as validation and average the results

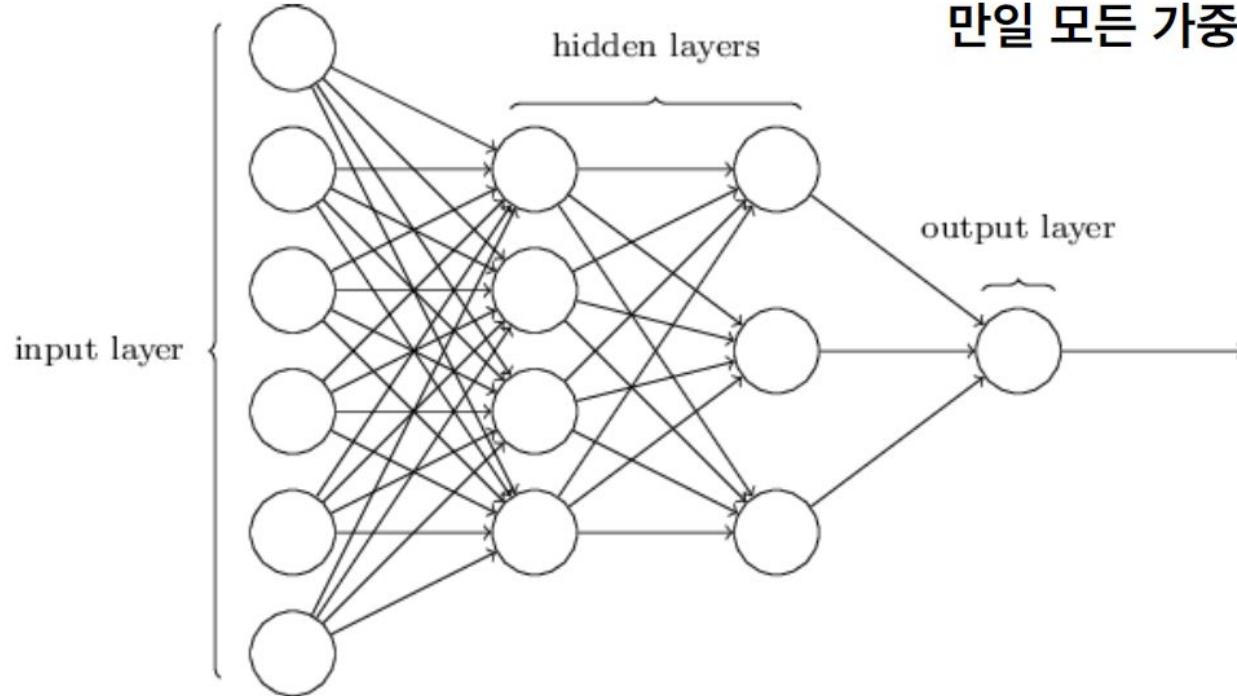
fold 1	fold 2	fold 3	fold 4	fold 5	test
--------	--------	--------	--------	--------	------

fold 1	fold 2	fold 3	fold 4	fold 5	test
--------	--------	--------	--------	--------	------

fold 1	fold 2	fold 3	fold 4	fold 5	test
--------	--------	--------	--------	--------	------

Useful for small datasets, but not used too frequently in deep learning

Weight Initialization



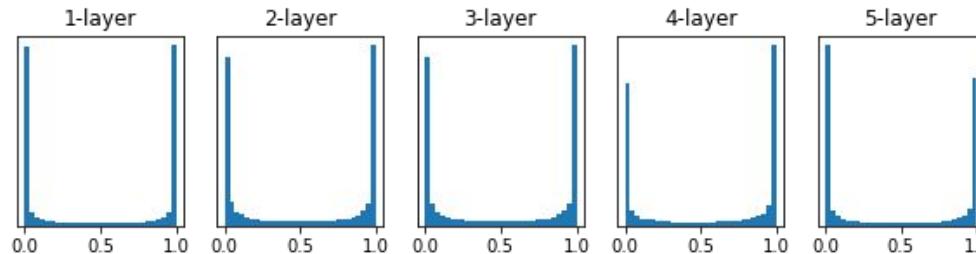
만일 모든 가중치가 0이라면?

Weight Initialization

- 초기값의 변화에 따라 은닉층의 활성화 값 분포가 어떻게 변할까?
 - Test
 - 5개의 layers, 각 100개 Nodes
 - 1000개의 무작위 입력 데이터
 - Activation Function - sigmoid, relu
 - 표준편차를 다르게 해서 실험

Weight Initialization - Sigmoid

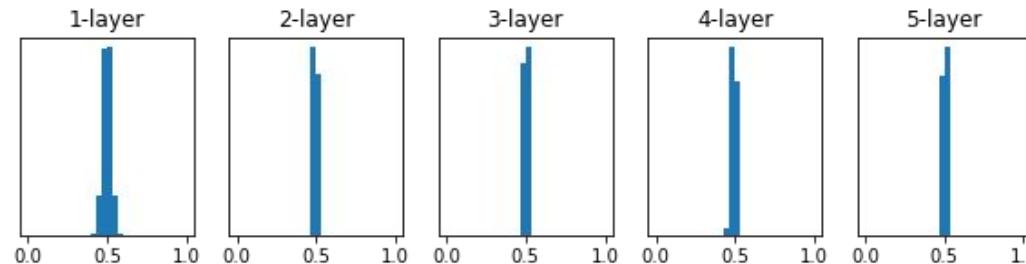
- 가중치를 표준편차가 1인 정규분포로 초기화



- 값이 0과 1에 치우침
- sigmoid 함수의 gradient가 죽는 현상이 일어남 (vanishing gradient)
- 네트워크가 깊어질수록 문제가 심해짐

Weight Initialization - Sigmoid

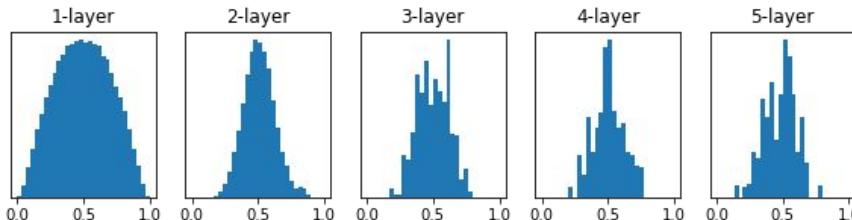
- 가중치를 표준편차가 0.01인 정규분포로 초기화



- 값이 0.5근처에 집중 : gradient가 죽지 않음
- 하지만 값이 몰려 있음 : 표현력이 제한 되어 있는 문제가 생김
 - 많은 뉴런이 같은 값을 출력한다는 것은 100개의 뉴런이 사실상 1개의 뉴런과 같음

Weight Initialization - Sigmoid

- Xavier Normal Initialization

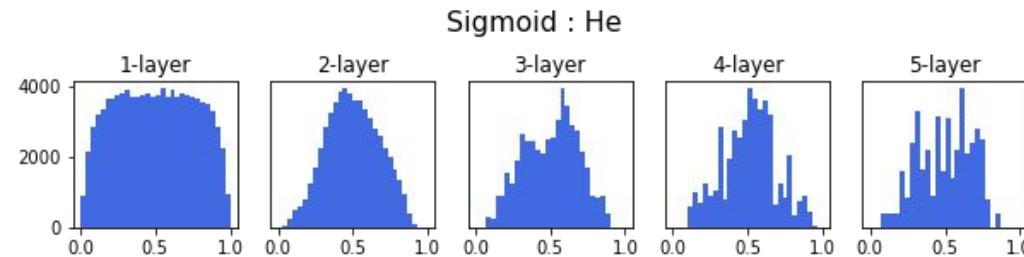
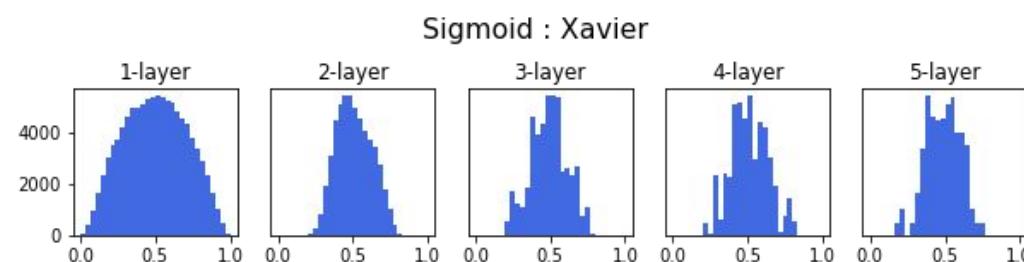
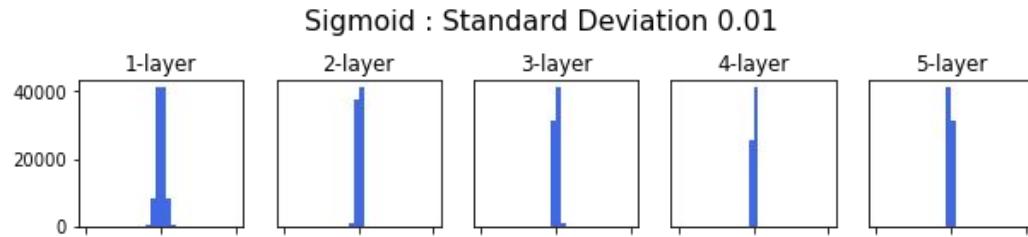


$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

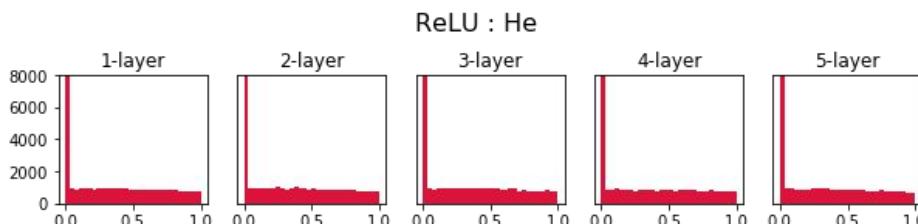
(n_{in} : 이전 layer(input)의 노드 수, n_{out} : 다음 layer의 노드 수)

Weight Initialization - Sigmoid



Weight Initialization - ReLU

- He Normal Initialization



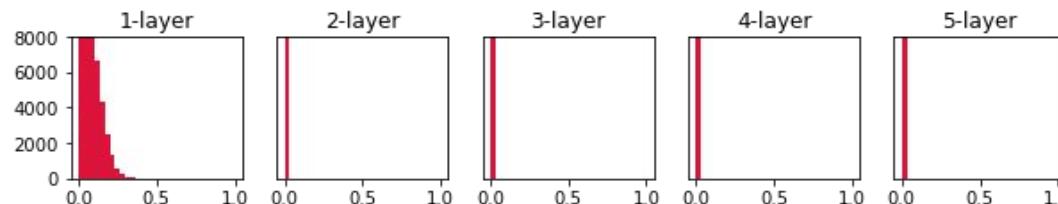
$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

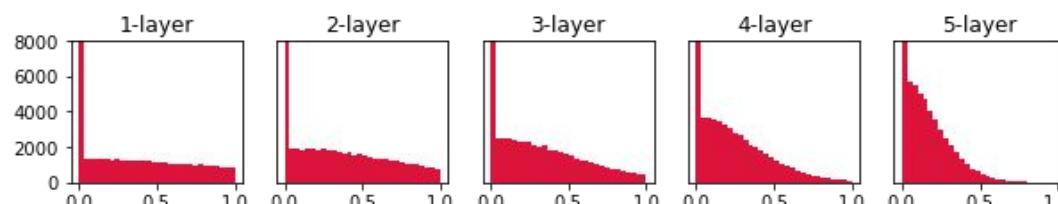
(n_{in} : 이전 layer(input)의 노드 수)

Weight Initialization - ReLU

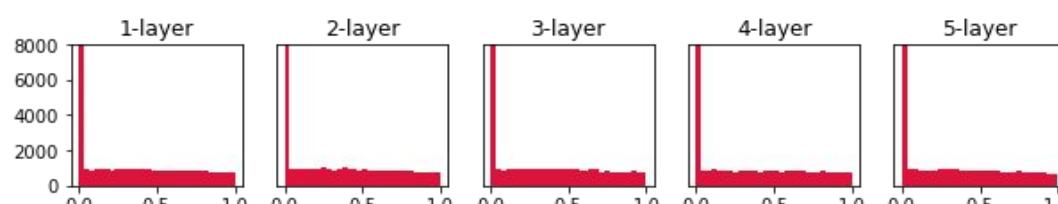
ReLU : Standard Deviation 0.01



ReLU : Xavier



ReLU : He



Weight Initialization

- Activation Function - sigmoid, tanh
 - Xavier initialization
- Activation Function - ReLU
 - He initialization
- Xavier init - default initializer in `tf.contrib.layers`

Regularization

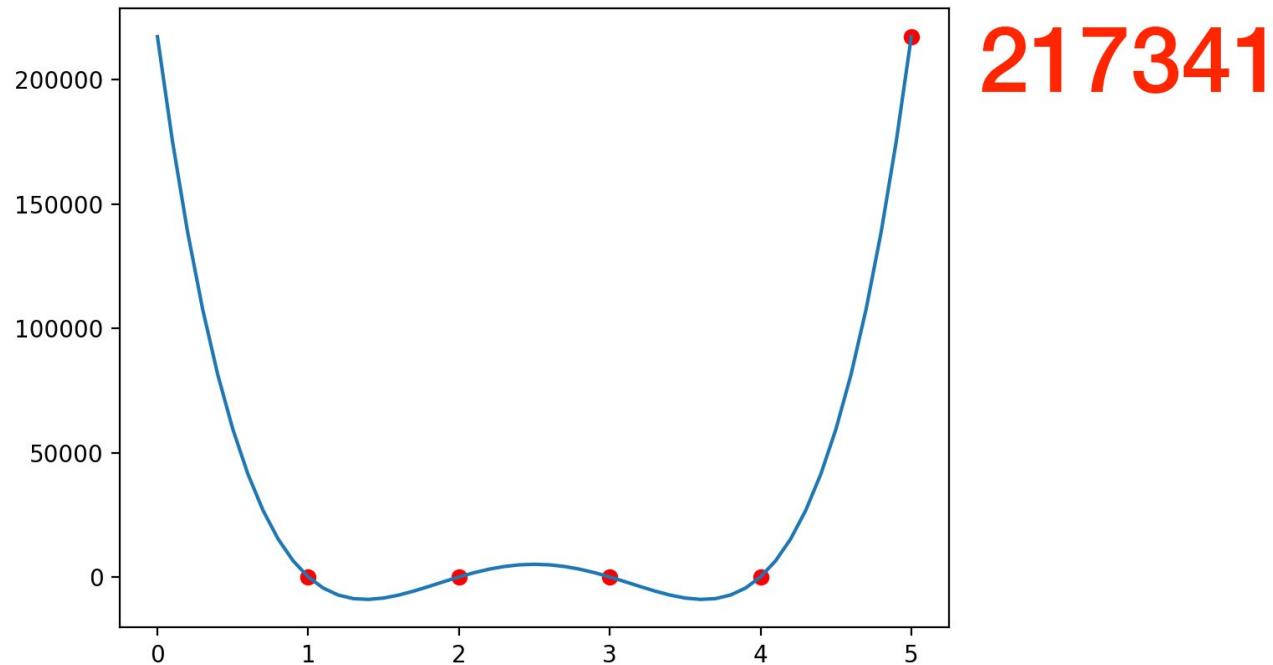
- 예측해 봅시다

1, 3, 5, 7, ?

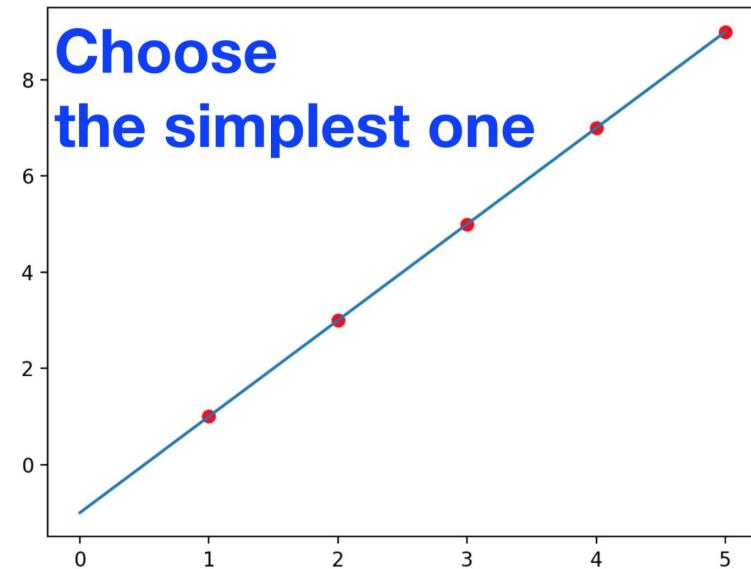
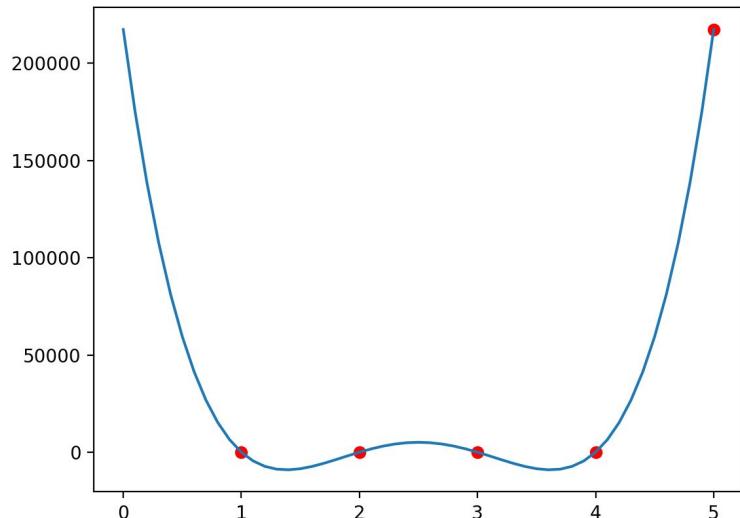
Regularization

217341

Regularization

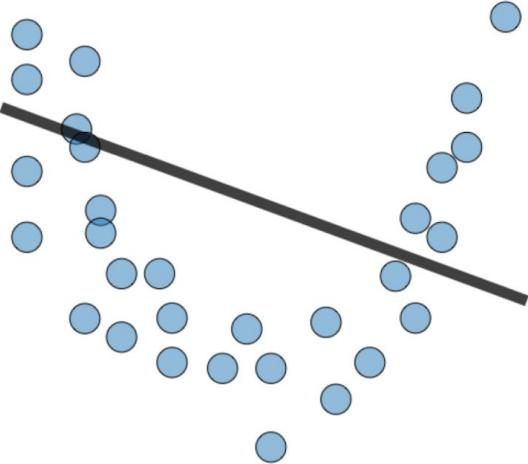


Regularization

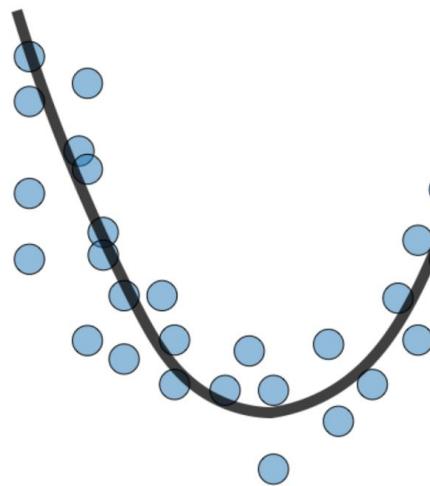


Choose
the simplest one

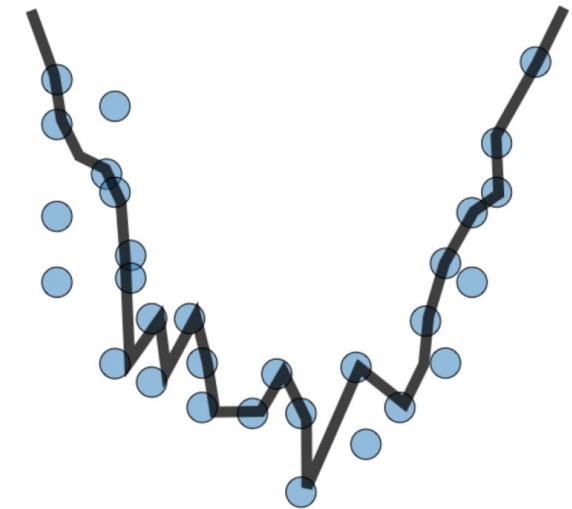
Overfitting and Underfitting



Underfitting



Appropriate fitting



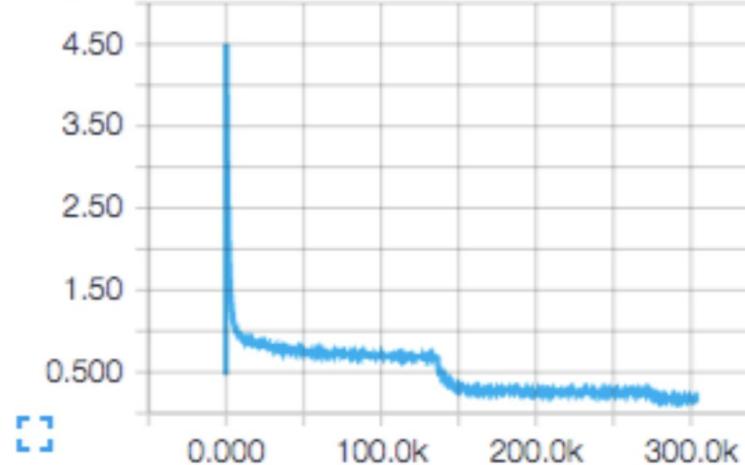
Overfitting

Overfitting

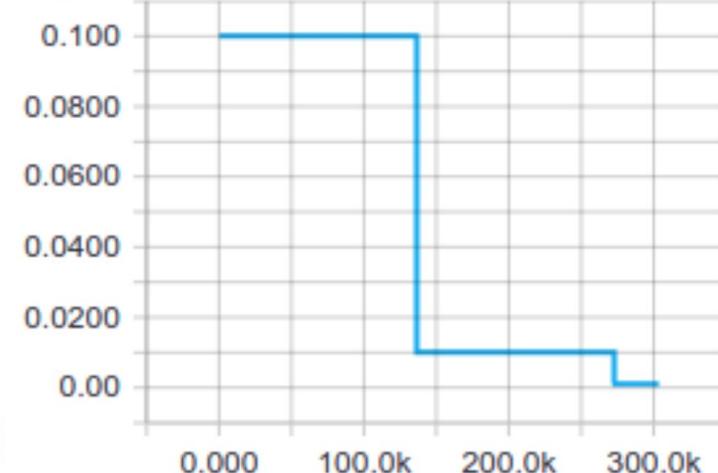
- 훈련 데이터(training data)에만 너무 과하게 학습되어 테스트에서 결과가 좋지 않음
 - 일반화 능력이 떨어진다
- 발생 원인?
 - 모델이 너무 복잡할 때
 - 데이터가 너무 적을 때
 - 혹은 둘 다

Monitoring Overfitting

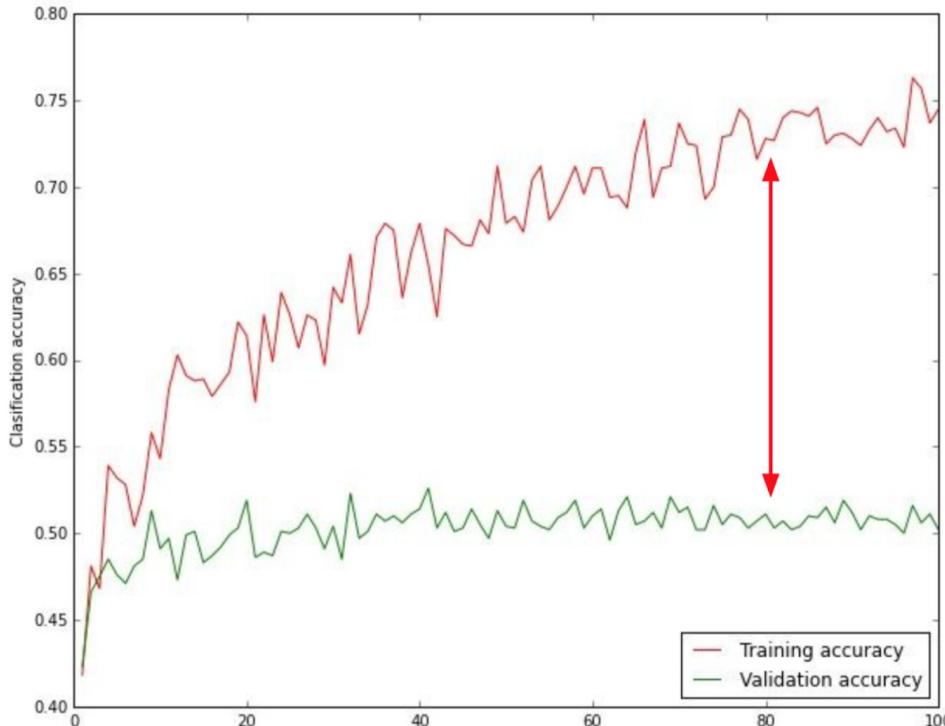
total_loss



learning_rate



Monitoring Overfitting



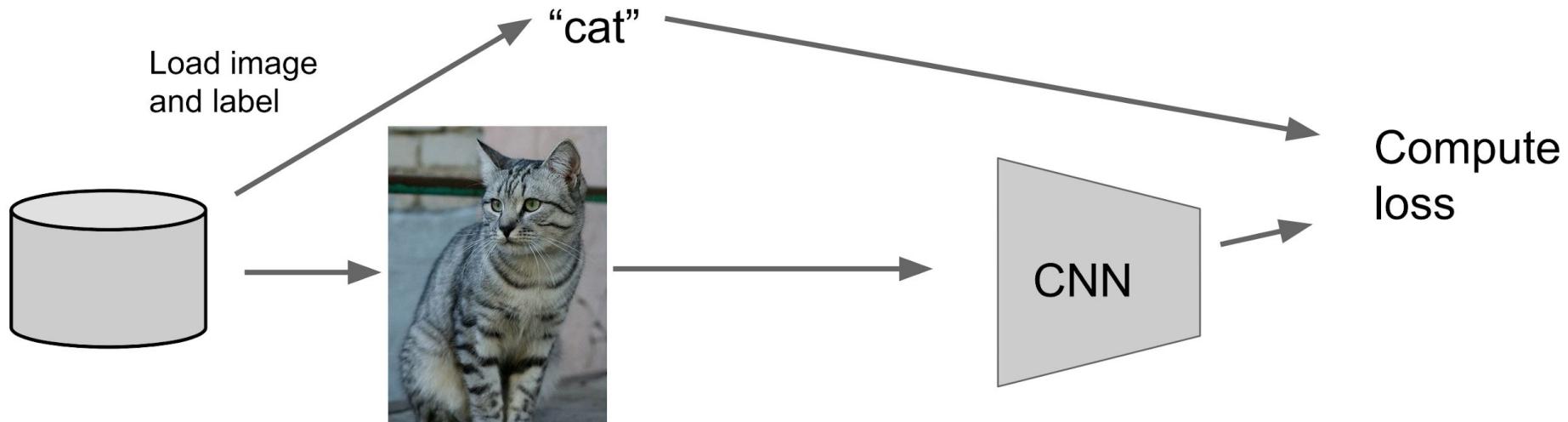
big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

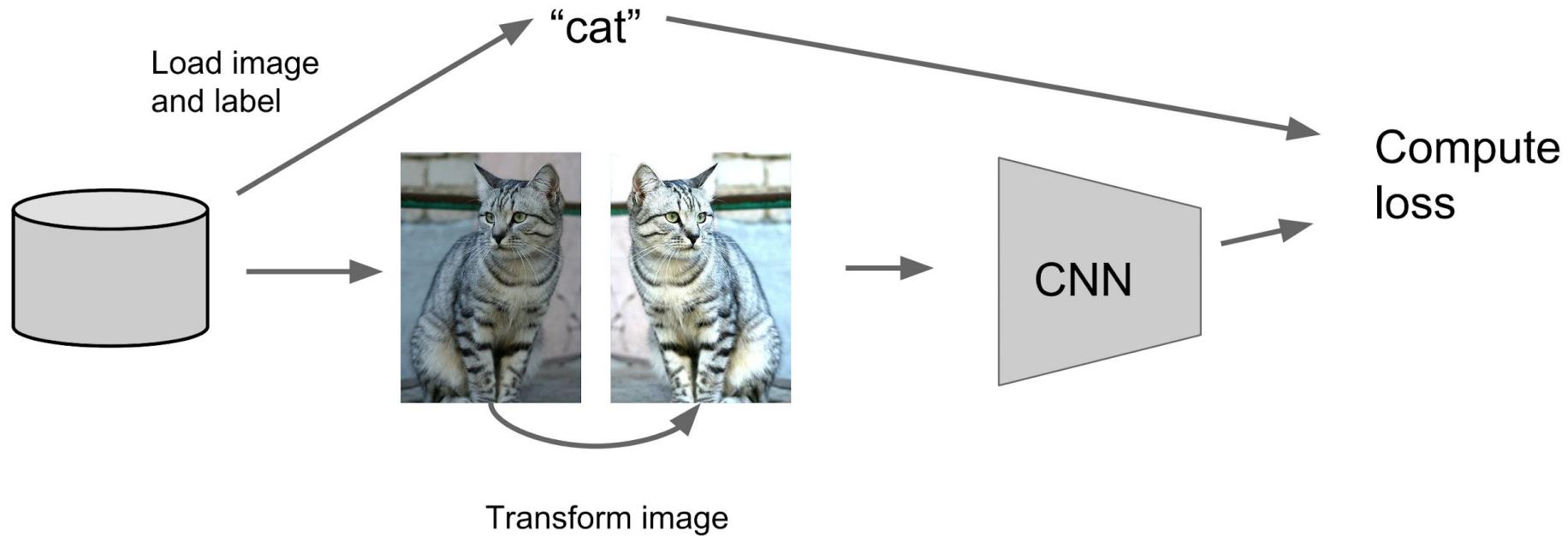
Regularization

- 오버피팅이 발생한다면 어떻게 조정할 수 있을까?

Data Augmentation



Data Augmentation



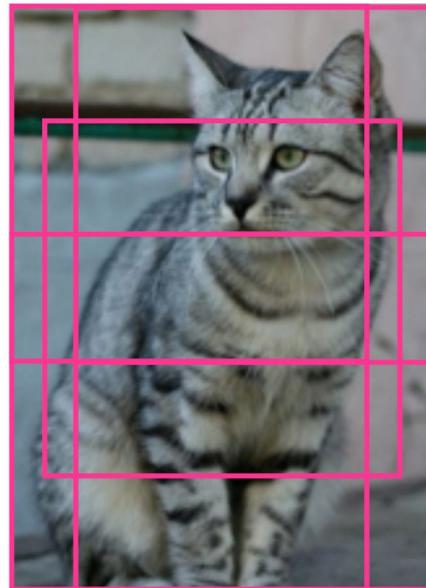
Data Augmentation

Simple: Randomize
contrast and brightness



Data Augmentation

- Training (Inception-V3)
 - aspect_ratio_range: [0.75, 1.33]
 - area_range: [0.05, 1.0]
 - resize: 299 x 299
- Testing (ResNet)
 - Resize image at 5 scales: {224, 256, 384, 480, 640}
 - For each size, use 10 224 x 224 crops:
(4 corners + center) x flips



Data Augmentation

- **Image data**
 - translation
 - rotation
 - stretching
 - shearing
 - lens distortions
 - and so on
- **Voice or music data**
 - add white noise
 - pitch shift
 - time expanding
 - and so on

Weight regularization

$$\mathcal{L} = \frac{1}{2N} \sum_i \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2 + \lambda R(\mathbf{w})$$

regularization strength

- 자주 사용하는 방법

L2 regularization

$$R(\mathbf{w}) = \sum_i \sum_j w_{ij}^2 \quad (\text{weight decay})$$

L1 regularization

$$R(\mathbf{w}) = \sum_i \sum_j |w_{ij}|$$

Elastic net (L1 + L2)

$$R(\mathbf{w}) = \sum_i \sum_j \beta w_{ij}^2 + |w_{ij}|$$

L2 regularization

전체 Loss

$$\mathcal{L} = \mathcal{L}_0 + \lambda \sum_w w^2$$

전체 Loss의 미분

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}_0}{\partial w} + 2\lambda w$$

가중치 업데이트

$$w \rightarrow w - \eta \frac{\partial \mathcal{L}_0}{\partial w} - 2\eta \lambda w$$

$$= \underline{(1 - 2\eta \lambda) w} - \eta \frac{\partial \mathcal{L}_0}{\partial w}$$

Weight Decay

L2 regularization

$$\mathbf{x} = [1, 1, 1, 1]$$

$$\mathbf{w}_1 = [1, 0, 0, 0] \rightarrow R(\mathbf{w}_1) = 1$$

$$\mathbf{w}_2 = [0.25, 0.25, 0.25, 0.25] \rightarrow R(\mathbf{w}_2) = 0.25$$

$$\mathbf{w}_1^\top \mathbf{x} = \mathbf{w}_2^\top \mathbf{x} = 1$$

⋮

L2 regularization

$$\mathbf{x} = [1, 1, 1, 1]$$

$$\mathbf{w}_1 = [1, 0, 0, 0] \rightarrow R(\mathbf{w}_1) = 1$$

$$\mathbf{w}_2 = [0.25, 0.25, 0.25, 0.25] \rightarrow R(\mathbf{w}_2) = 0.25$$

$$\mathbf{w}_1^\top \mathbf{x} = \mathbf{w}_2^\top \mathbf{x} = 1$$

⋮

$$\mathbf{x}' = [10, 1, 1, 1]$$

$$\mathbf{w}_1 = [1, 0, 0, 0]$$

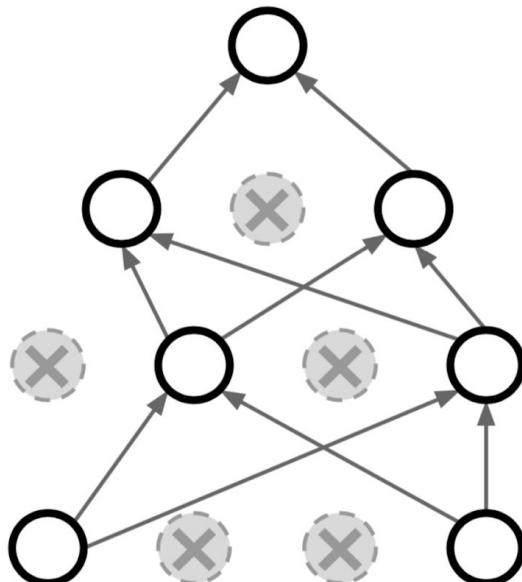
$$\mathbf{w}_2 = [0.25, 0.25, 0.25, 0.25]$$

⋮

$$\mathbf{w}_1^\top \mathbf{x}' = 10$$

$$\mathbf{w}_2^\top \mathbf{x}' = 3.25$$

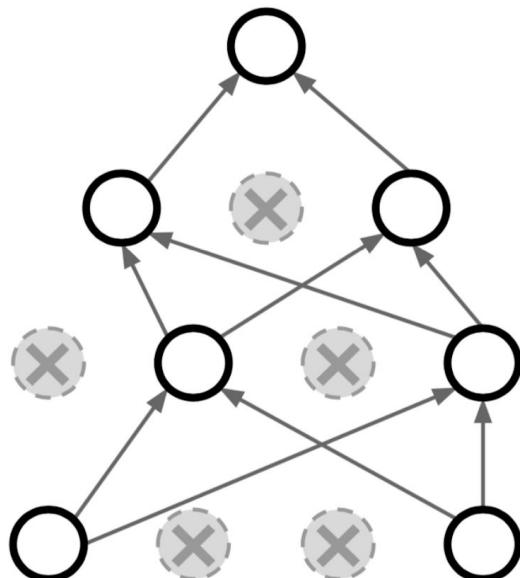
Dropout



Forces the network to have a redundant representation;
Prevents co-adaptation of features



Dropout



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Batch Normalization

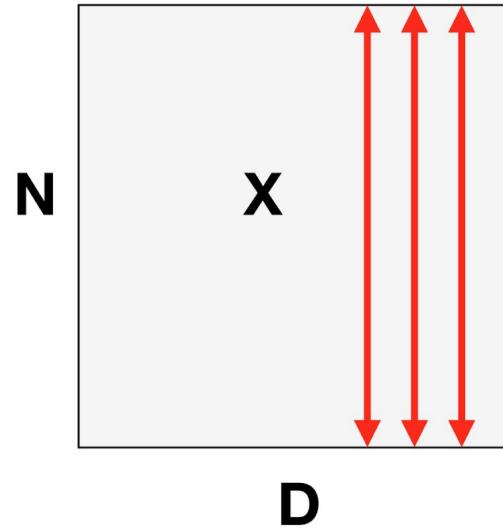
- 장점
 - 학습을 빨리 진행할 수 있다
 - 초기값에 크게 의존하지 않는다
 - Overfitting을 억제한다
 - Dropout의 필요성이 감소한다
- 각 층에서 활성화 값이 정규분포가 되도록 조정한다



Batch Normalization

- 각 레이어의 input을 정규분포로 만들어보자

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$



Batch Normalization

Vanilla Normalization

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

$\gamma^{(k)}, \beta^{(k)}$ 는 학습되는 파라미터

$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathbb{E}[x^{(k)}]$$

Scaling and Transformation

$$\hat{y}^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$



Batch Normalization

- BatchNorm layer at test time
 - 테스트 배치에 대해서 mean/std를 계산하지 않는다
 - 대신 트레이닝 하는 동안 single fixed empirical mean/std를 사용 한다
 - 실제로는 트레이닝 시 mean/std의 moving average를 계산하여 미니 배치 통계량으로 사용한다

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

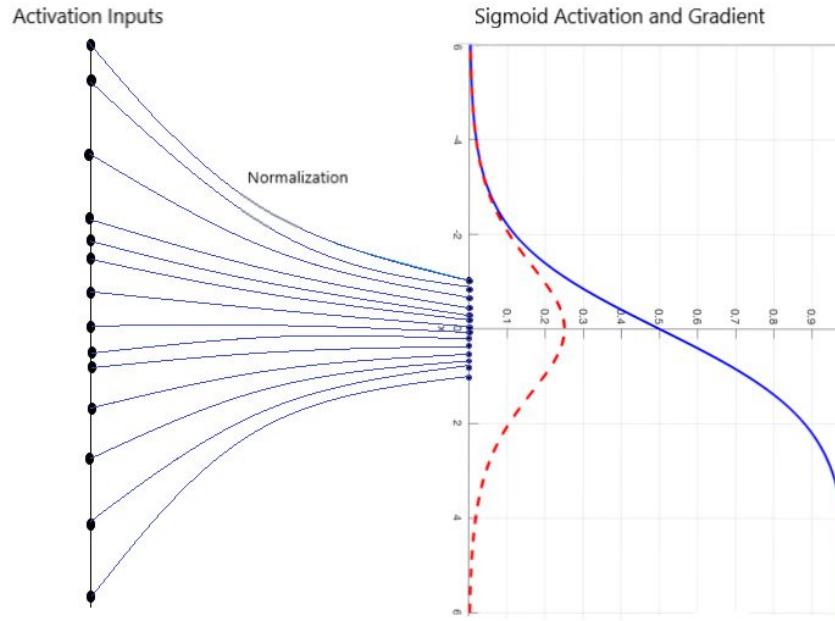
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Batch Normalization



참고자료

- 밑바닥부터 시작하는 딥러닝 1, 2
<http://www.yes24.com/Product/Goods/34970929?Acode=101>
<http://www.yes24.com/Product/Goods/72173703>
- 모두를 위한 딥러닝 시즌2
<https://www.edwith.org/boostcourse-dl-tensorflow/joinLectures/22150>
- 모두의 연구소 이일구님 강의 자료
<https://github.com/ilguyi>
- KAIST MAC Lab. 남주한 교수님 강의 자료
<http://mac.kaist.ac.kr/~juhan/gct634/>