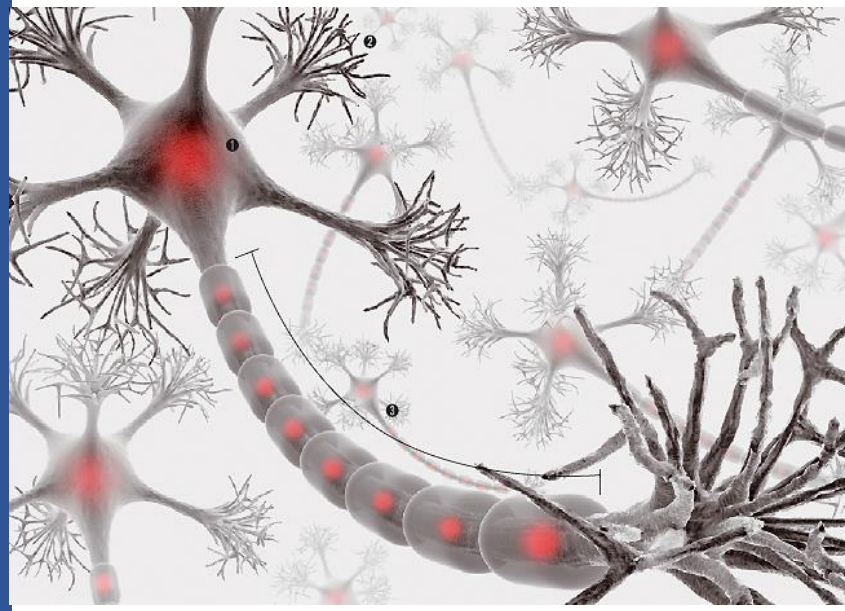
순방향 신경망 (Feedforward Network)

학습 목표

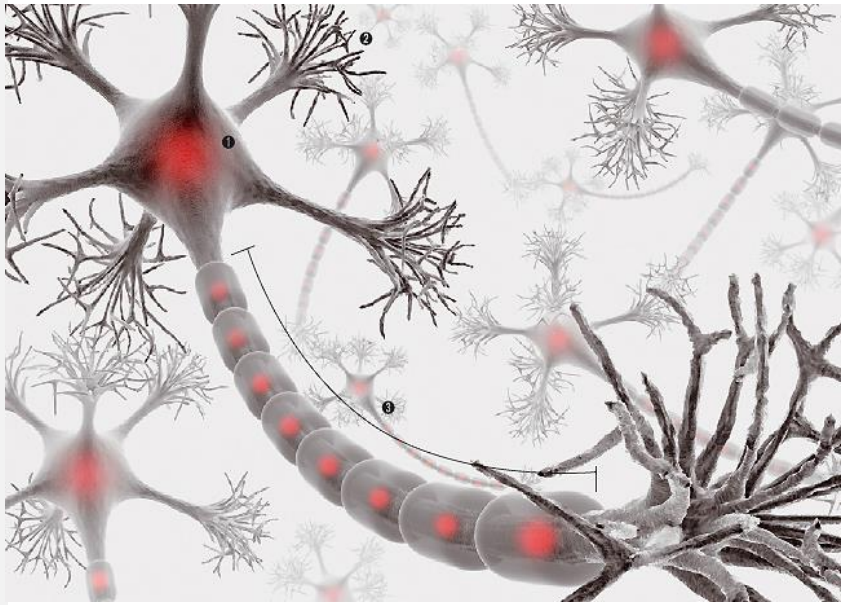
- 피드포워드 네트워크 구조를 이해한다.

주요 내용

1. 함수로서의 인공신경망
2. 피드포워드 네트워크 구조
3. 입력 계층 (Input Layer)
4. 은닉 계층 (Hidden Layer)
5. 출력 계층 (Output Layer)
6. 네트워크 크기



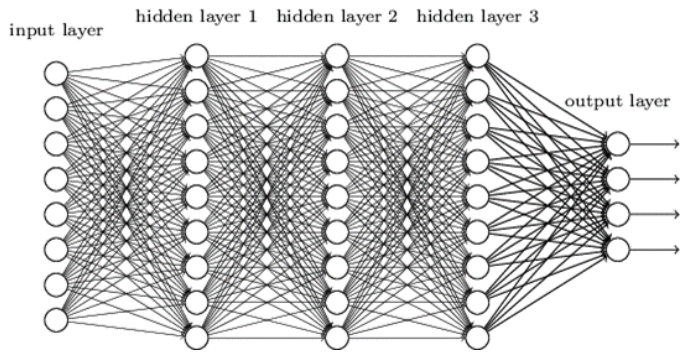
1 함수로서의 인공신경망



인공 신경망

$$y = f(x;\theta)$$

x ➡ Smooth and Continuous Geometric Transformation ➡ y



깊은 신경망은 아주 복잡한 맵핑 관계를 표현하는 **맵핑 함수**이다!

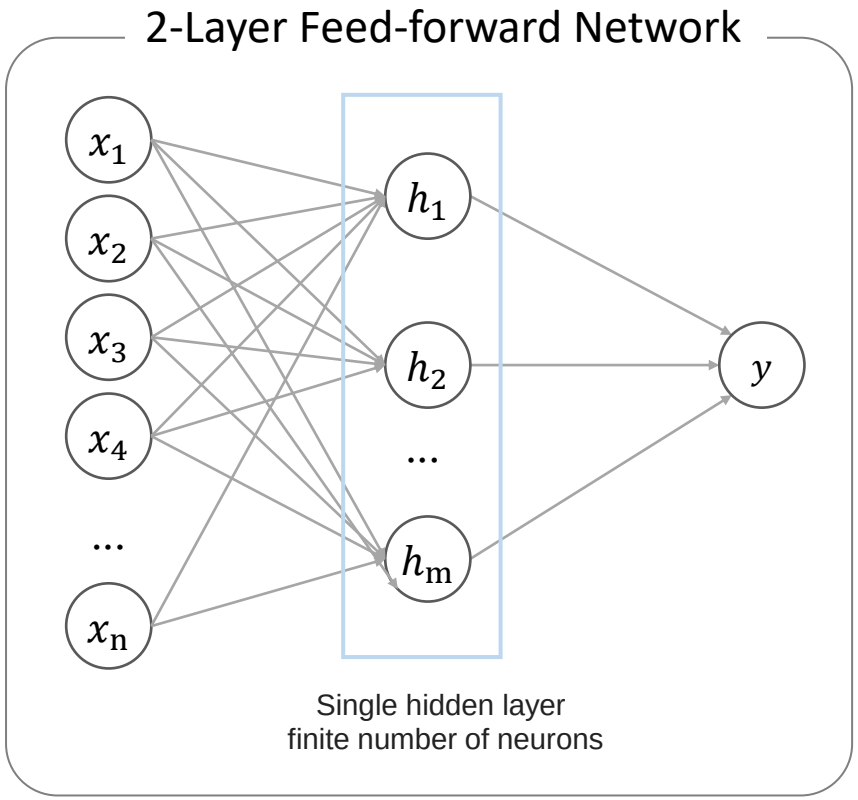
Universal Approximation Theorem

Universal Approximation Theorem

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbf{R}^n$, under mild assumptions on the activation function.

$$f(x) \in \mathrm{R}^n$$

Continuous function



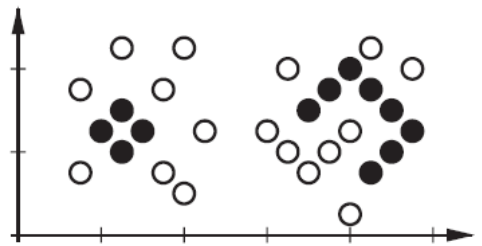
"더 깊은 신경망이 필요한가?"

- 신경망을 깊게 하면 적은 수의 뉴런으로 함수를 구현할 수 있음
- 신경망이 깊어질수록 함수를 정확하게 근사할 수 있음 (임계점이 적어지고 Local Minima가 모여서 최적점을 잘 찾음)

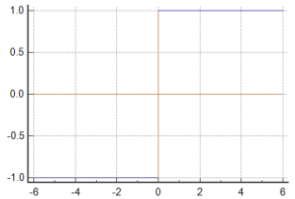
("Geometry of energy landscapes and the optimizability of deep neural networks", Cambridge, 2018)

예제 - Classification

분류 문제



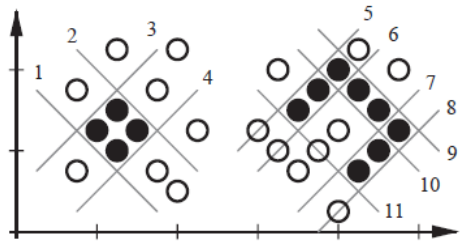
Activation function :



$$f(n) = \begin{cases} 1 & \text{if } n \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

3계층 네트워크 필요

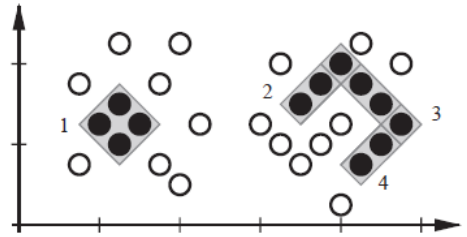
첫번째 계층



$$(\mathbf{W}^1)^T = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

$$(\mathbf{b}^1)^T = \begin{bmatrix} -2 & 3 & 0.5 & 0.5 & -1.75 & 2.25 & -3.25 & 3.75 & 6.25 & -5.75 & -4.75 \end{bmatrix}$$

두번째 계층



$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}, \mathbf{b}^T = \begin{bmatrix} -3 \\ -3 \\ -3 \\ -3 \end{bmatrix}$$

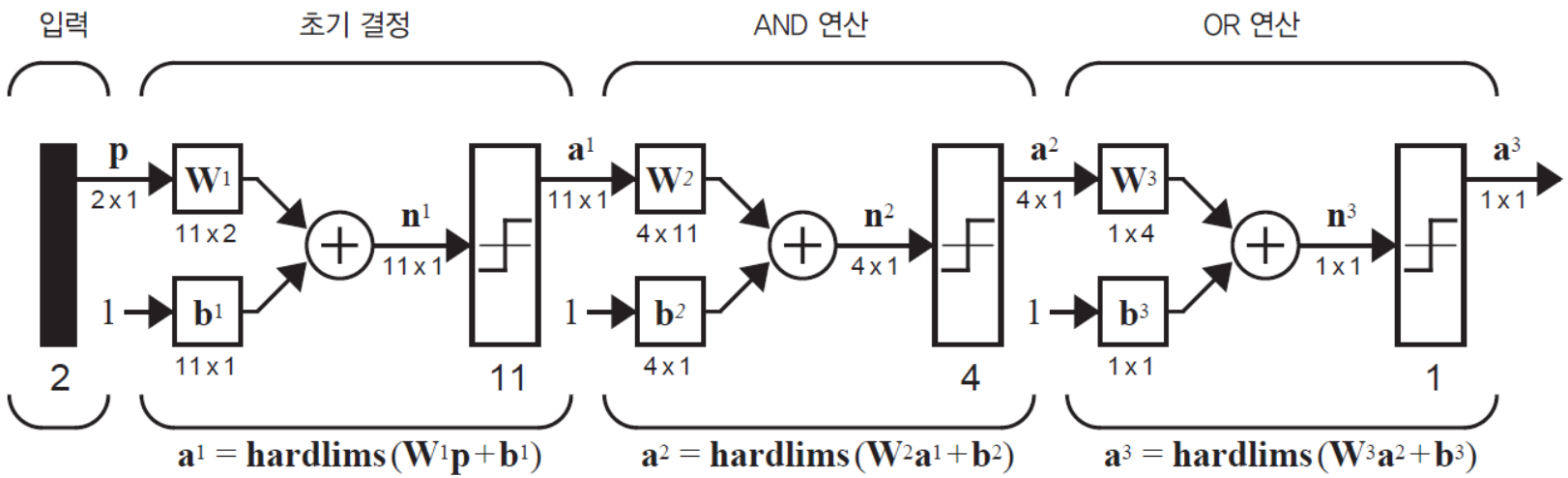
11개 뉴런의 출력을 두 번째 계층의 AND 뉴런을 이용해 그룹으로 결합

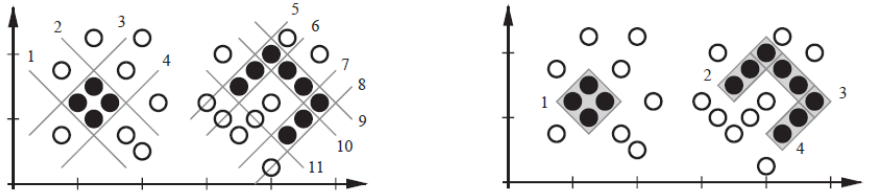
세번째 계층

세 번째 계층에서는 OR 연산을 사용해 두 번째 계층의 네 결정 영역을 하나의 영역으로 결합

$$\mathbf{W}^3 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}, \mathbf{b}^3 = \begin{bmatrix} 3 \end{bmatrix}$$

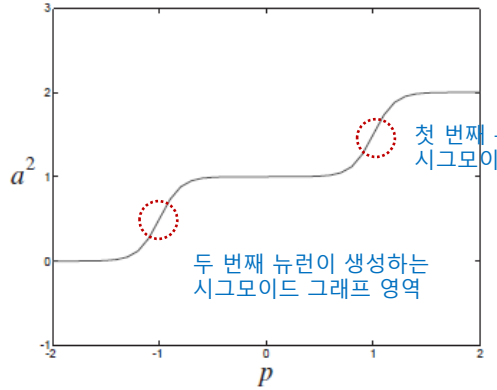
예제 - Classification



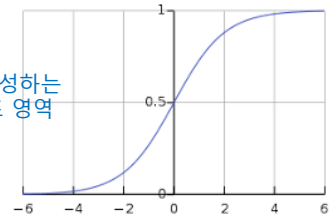


예제 - Function Approximation

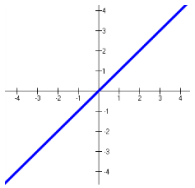
함수 근사 문제



Activation function :

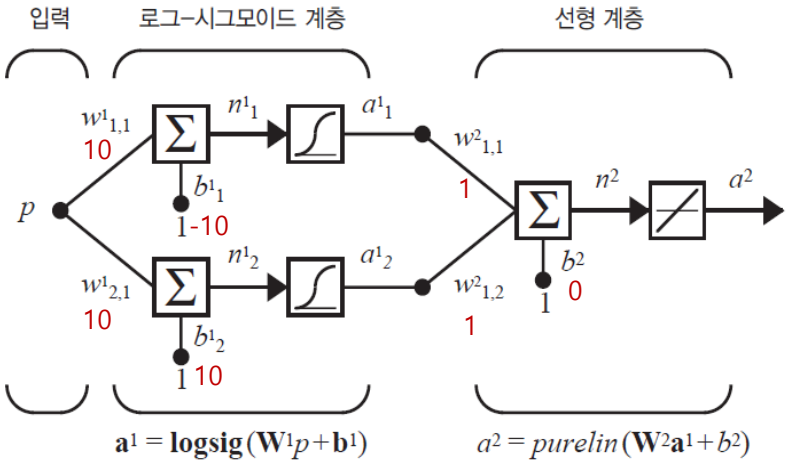


$$f^1(n) = \frac{1}{1+e^{-n}}$$



$$f^2(n) = n$$

함수 근사를 위한 2계층 네트워크



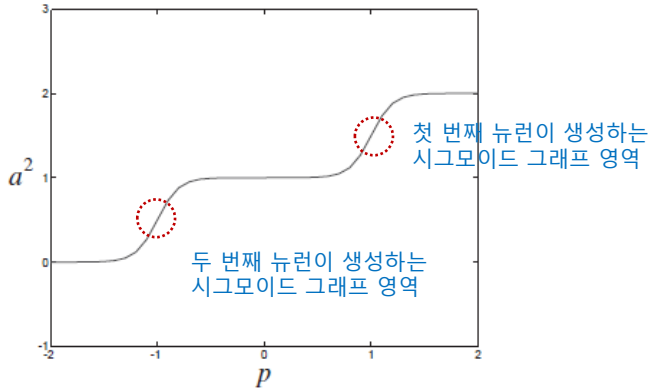
$$n_1^1 = w_{1,1}^1 p + b_1^1 = 0 \quad \Rightarrow \quad p = -\frac{b_1^1}{w_{1,1}^1} = -\frac{-10}{10} = 1$$

$$n_2^1 = w_{2,1}^1 p + b_2^1 = 0 \quad \Rightarrow \quad p = -\frac{b_2^1}{w_{2,1}^1} = -\frac{10}{10} = -1$$

계단의 중심은 첫 번째 계층 뉴런의 네트 입력이 0인 지점에 있다.

예제 - Function Approximation

네트워크 반응



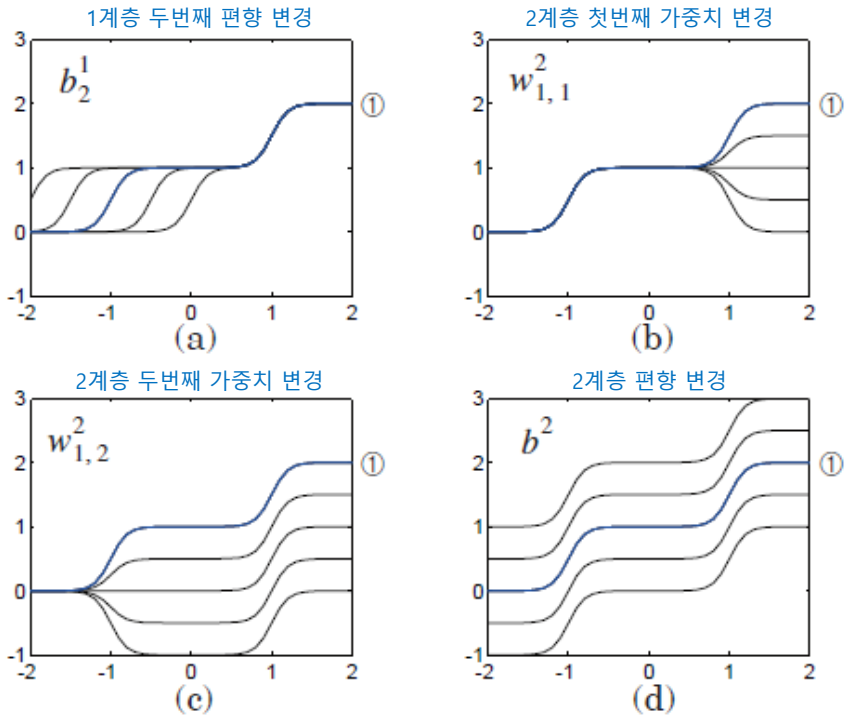
$$n_1^1 = w_{1,1}^1 p + b_1^1 = 0 \quad \Rightarrow \quad p = -\frac{b_1^1}{w_{1,1}^1} = -\frac{-10}{10} = 1$$

$$n_2^1 = w_{2,1}^1 p + b_2^1 = 0 \quad \Rightarrow \quad p = -\frac{b_2^1}{w_{2,1}^1} = -\frac{10}{10} = -1$$

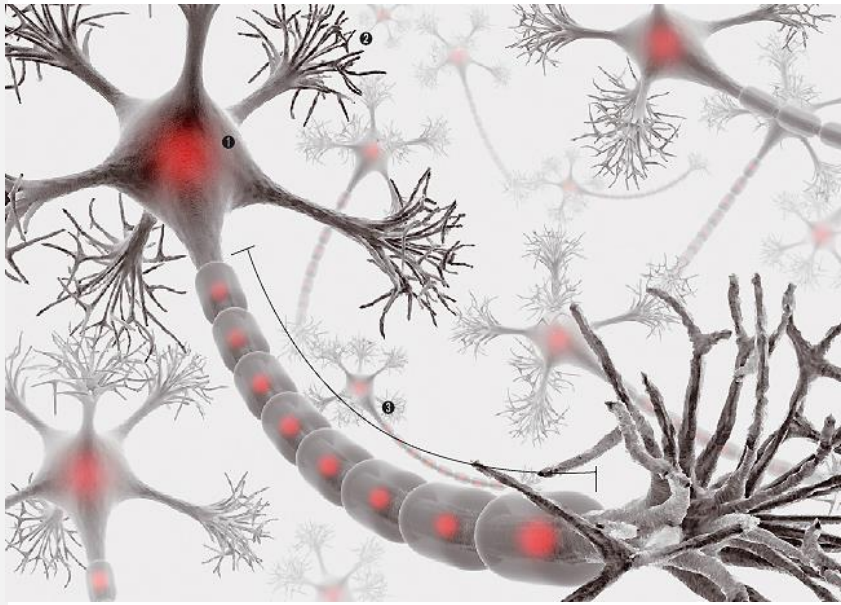
계단의 중심은 첫 번째 계층 뉴런의 네트 입력이 0인 지점에 있다.

파라미터 변경

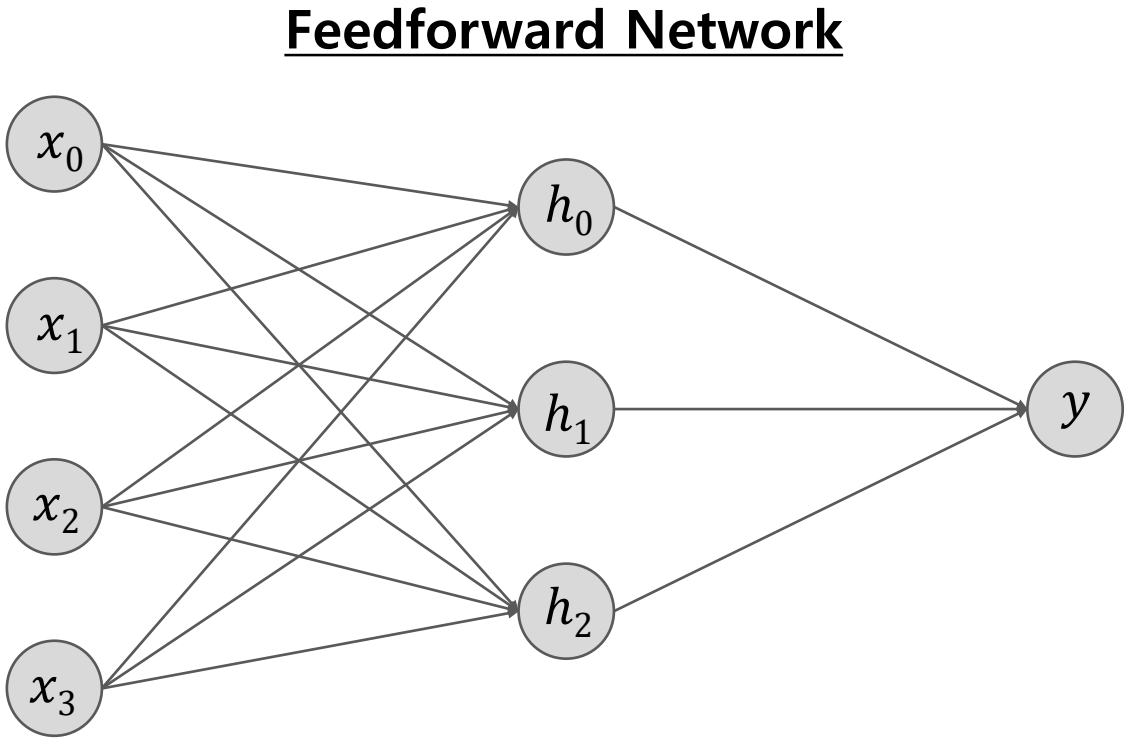
네트워크 반응에 미치는 영향



2 피드포워드 네트워크 구조



피드포워드 네트워크



- 모든 연결이 입력에서 출력 방향으로만 되어 있음
- 다층 퍼셉트론(Multi-Layered Perceptron)과 동일
- 입력 데이터를 1차원 벡터 형태로 받아서 처리

네트워크 설계

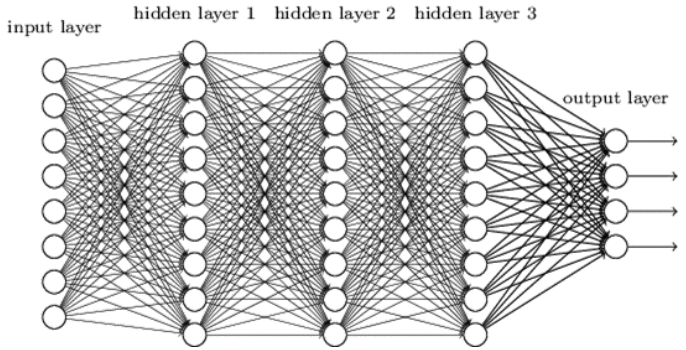
$$y = f(x;\theta)$$

1 Input

- 입력 형태

x

Smooth and Continuous
Geometric Transformation



y

2 Output

- 출력 형태
- Activation Function

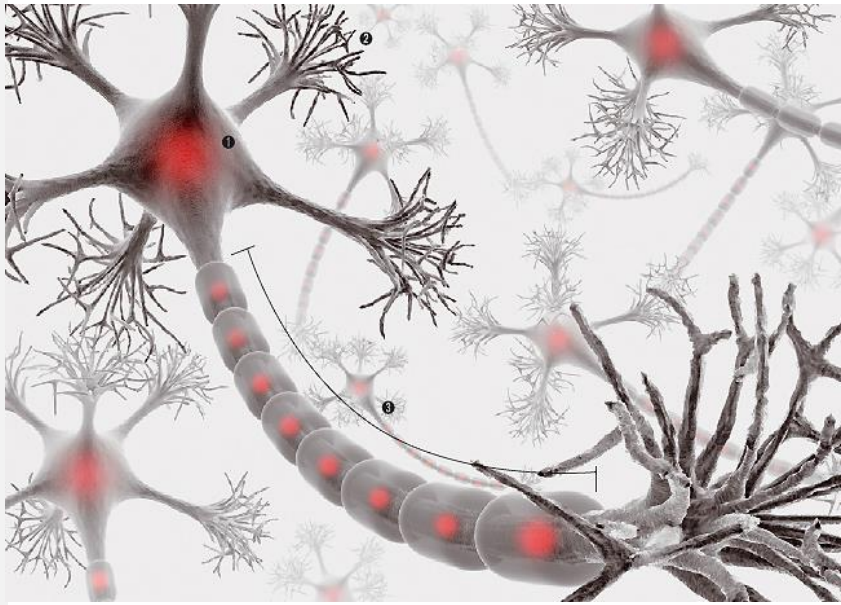
3 Hidden

- Activation Function

4 Network Size

- 네트워크 깊이 (depth) : 레이어 수
- 네트워크 폭 (width) : 레이어 별 뉴런 수
- 연결 방식

3 입력 계층 (Input Layer)

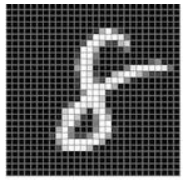


Input 입력 형태

Input :

$$x = (x_1, x_2, x_3, \dots, x_n)$$

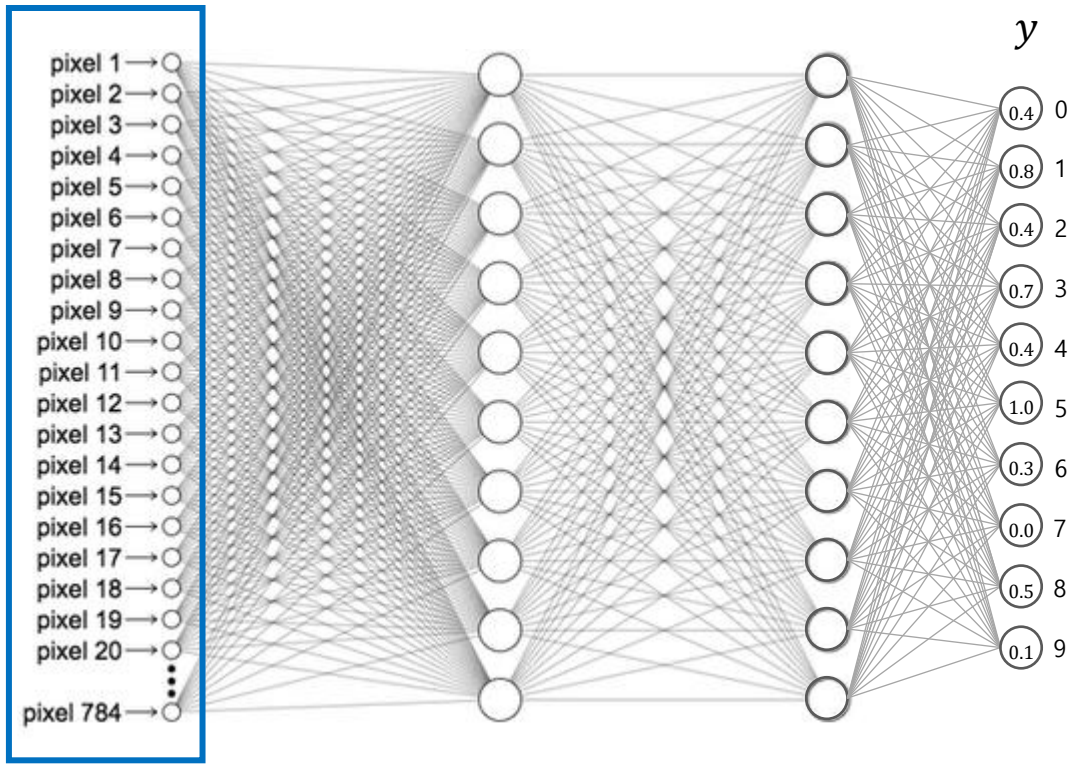
- 숫자로 이뤄진 n차원 벡터



28x28 이미지

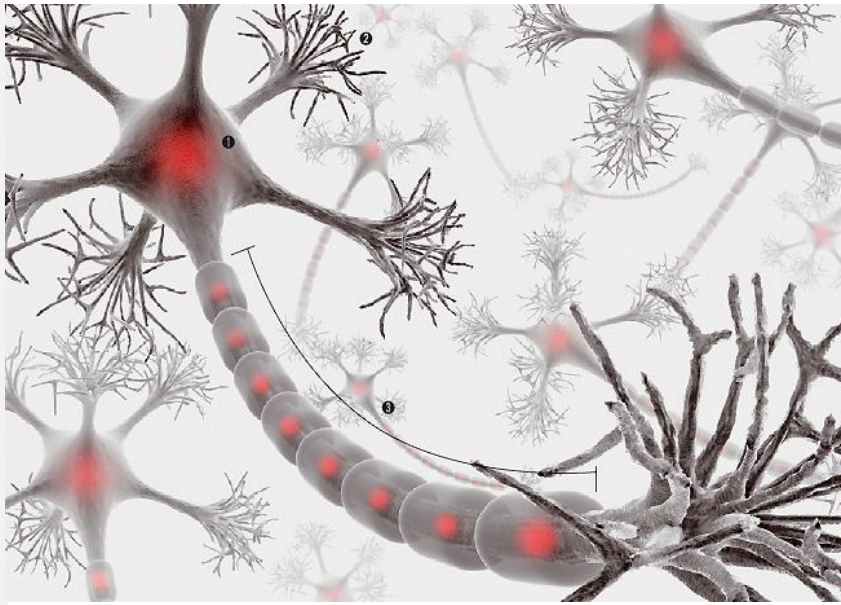
MNIST 예제

784 차원 벡터



- 28x28 픽셀을 1차원 벡터로 변환해서 입력
- 784차원의 입력 벡터

4 은닉 계층 (Hidden Layer)

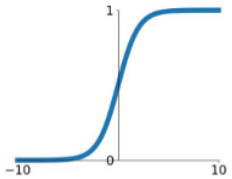


Hidden Activation Function 종류

Activation Function

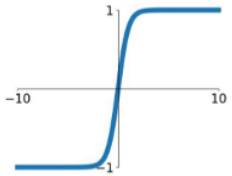
Sigmoid

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



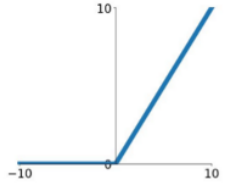
tanh

$$\tanh(x)$$



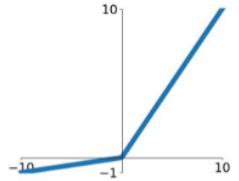
ReLU

$$\max(0, x)$$



Leaky ReLU

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

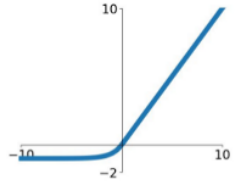


Maxout

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

ELU

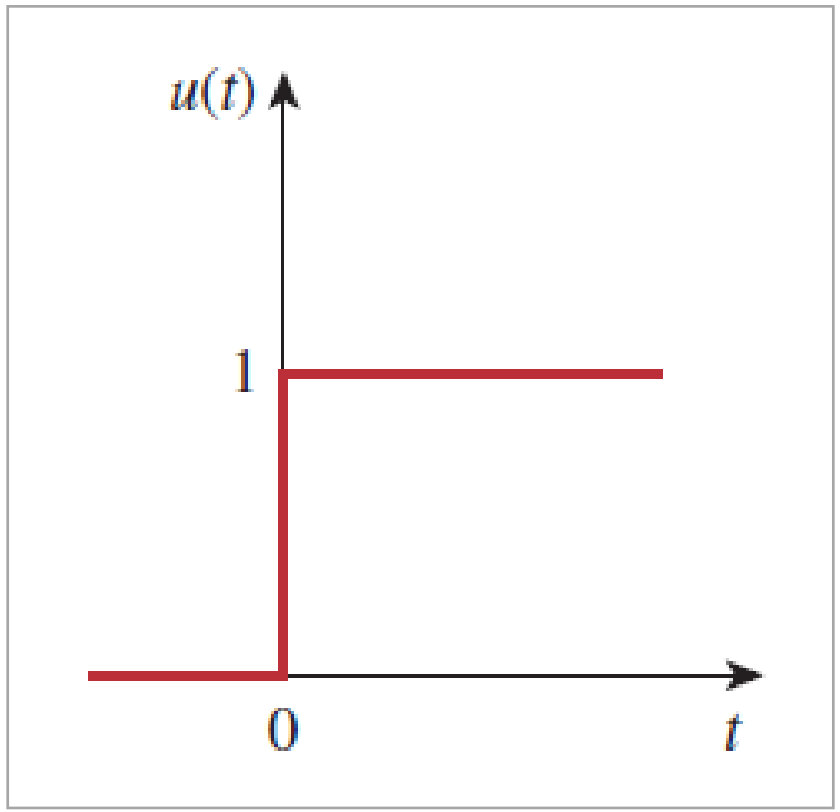
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- Hidden Unit 설계는 주요 연구 분야로 명확한 가이드라인이 많지 않음
- ReLU 계열이 좋은 성능을 보이고 있는 상황

Hidden Step Function

계단 함수 (Step function)

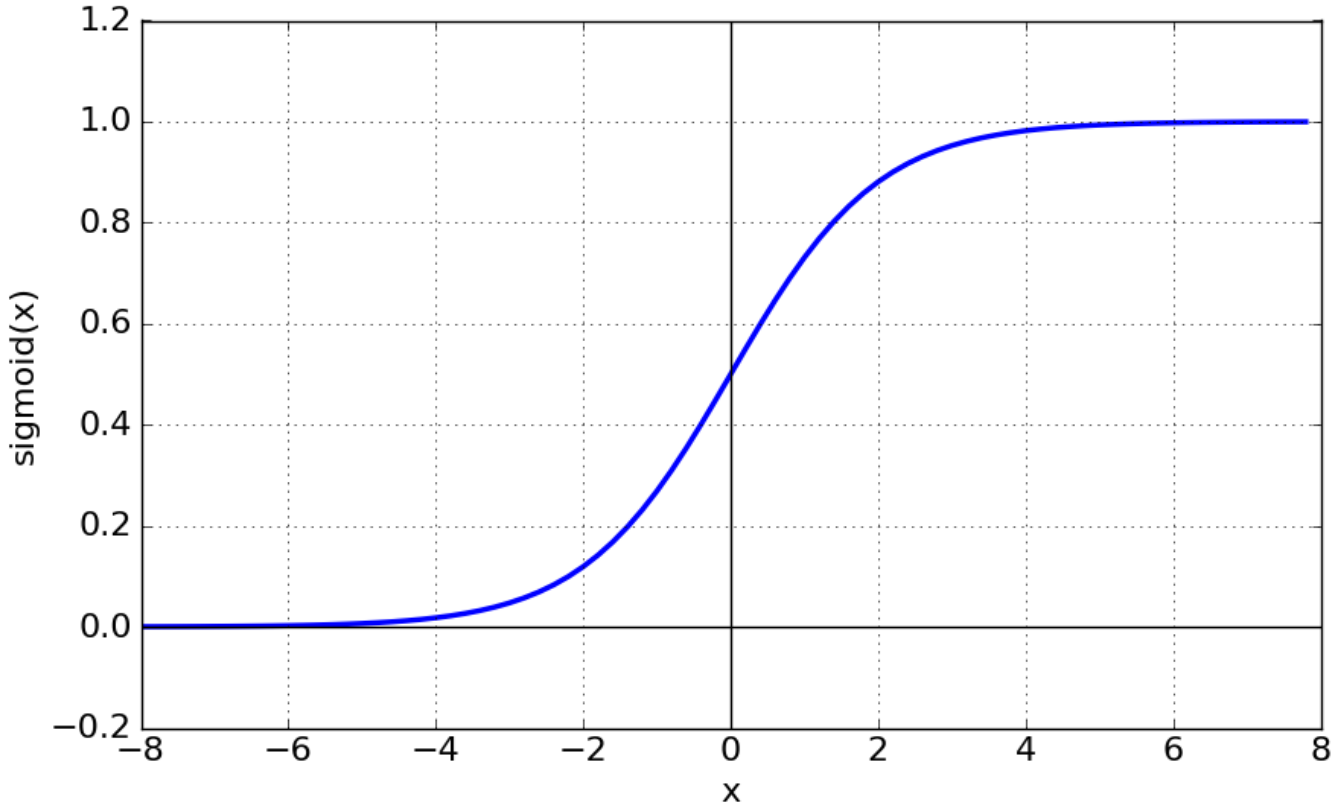


$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- 생체 뉴런의 발화 방식과 유사하며 퍼셉트론에서 사용
- 값이 0과 1
- 미분 값이 0이어서 학습 불가능.
- 따라서, Gradient 방식의 최적화를 사용하는 신경망에서 사용할 수 없음

Hidden Sigmoid

시그모이드 함수 (Sigmoid Function)

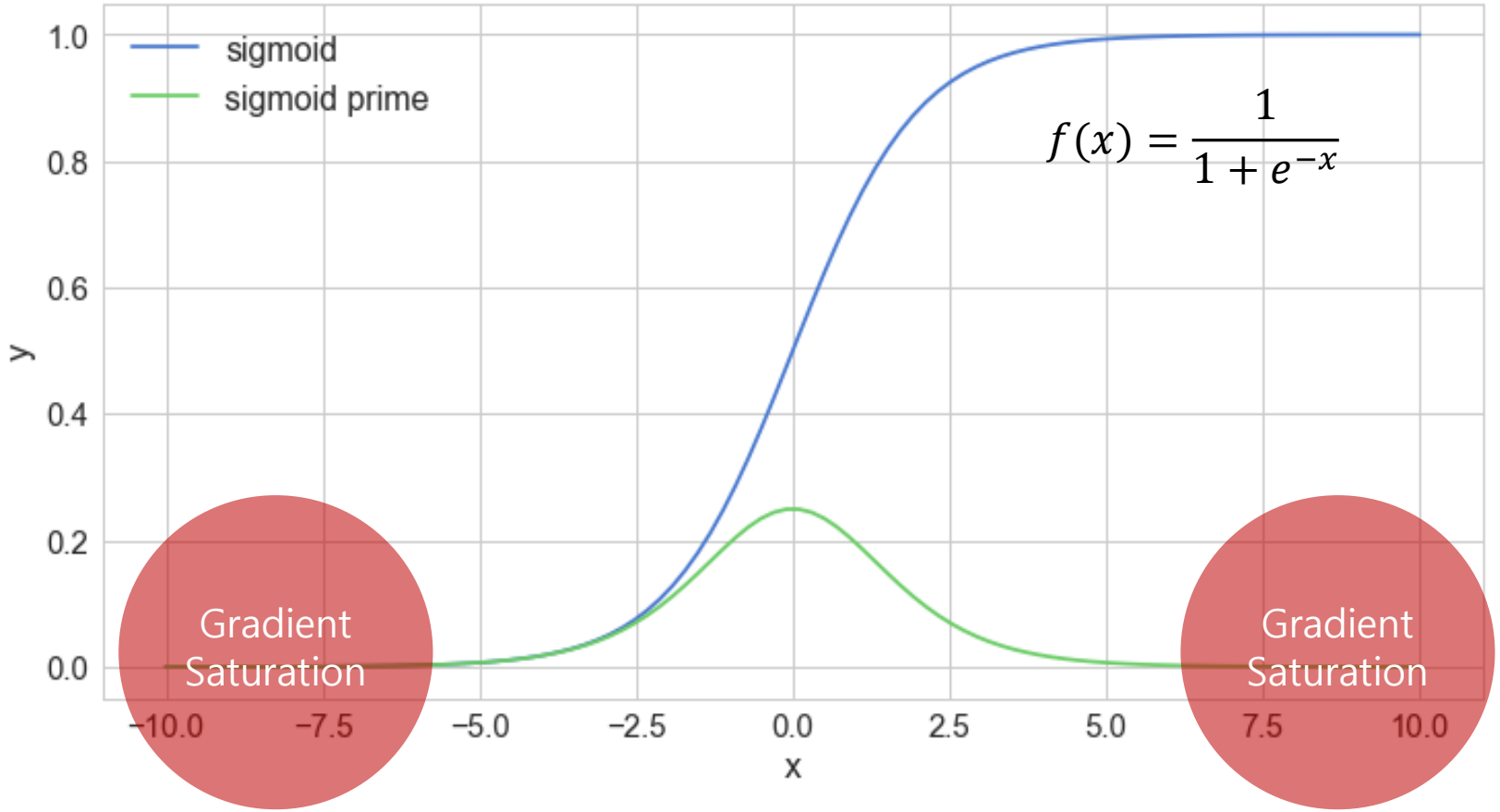


$$f(x) = \frac{1}{1 + e^{-x}}$$

- Step Function의 Smooth한 버전
- 미분 가능
- 0과 1사이의 값을 가짐
- exp 계산 비용이 비쌈
- Zero-centered가 아니어서 최적화에 비효율적

Hidden Sigmoid

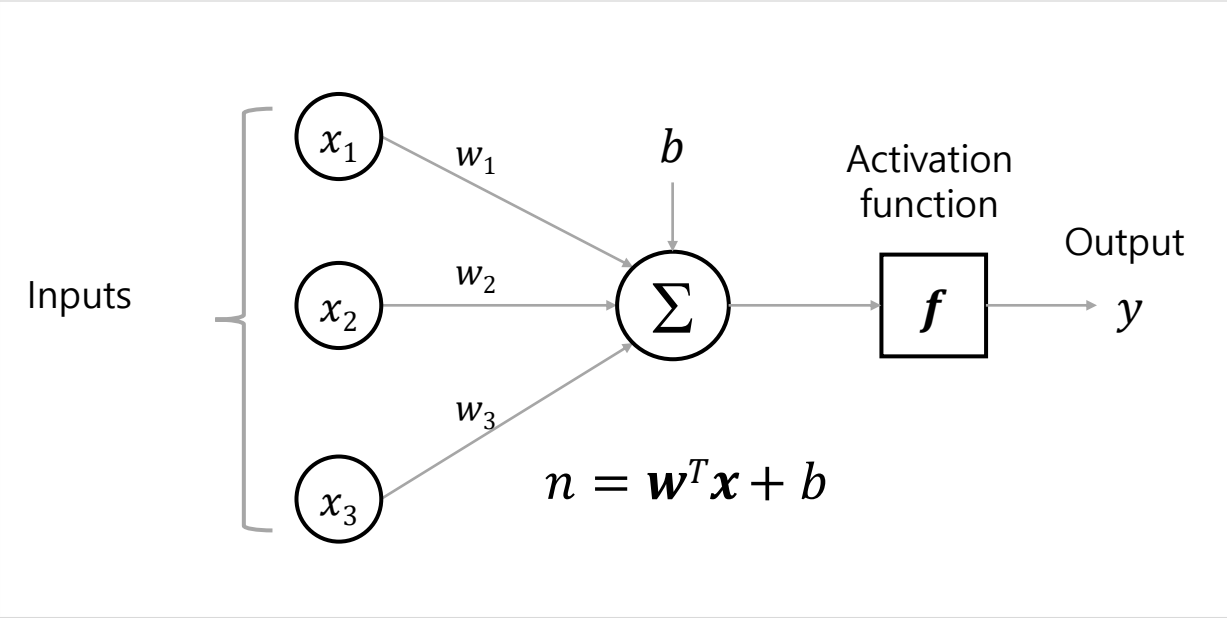
시그모이드 함수의 미분



Gradient Saturation 문제가 발생!

Hidden If Not Zero-Centered

뉴런의 입력이 항상 양수라면?



Local Gradient $\frac{\partial n}{\partial \boldsymbol{w}}$?

$\boldsymbol{x}$이므로 항상 양수

Hidden If Not Zero-Centered

Gradient Descent

$$w^{+} = w - \alpha \frac{\partial L}{\partial w}$$

L : Loss Function

Step Size

Gradient

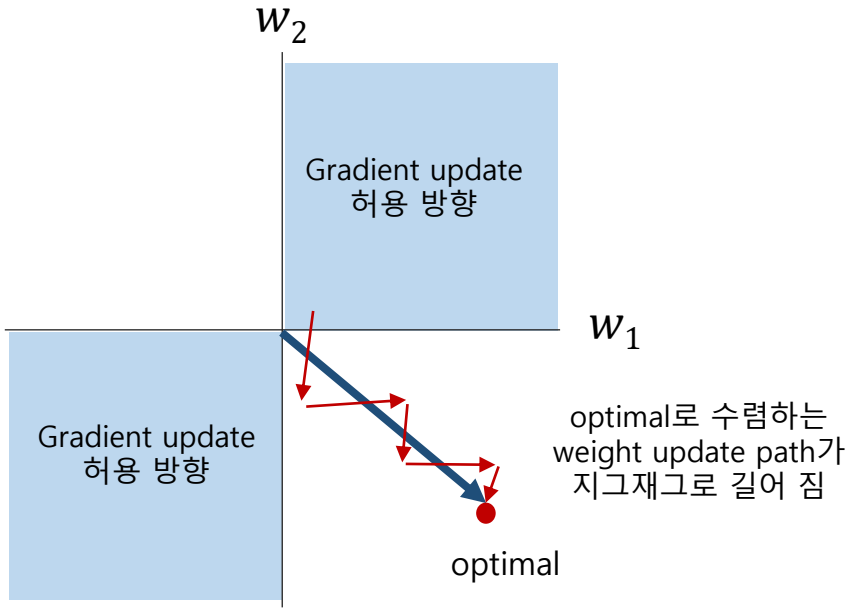
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial n} \cdot \frac{\partial n}{\partial w}$$

$\frac{\partial n}{\partial w} = x$ 는 항상 양수이므로

$\frac{\partial L}{\partial w}$는 $\frac{\partial L}{\partial n}$의 부호를 따르게 됨.

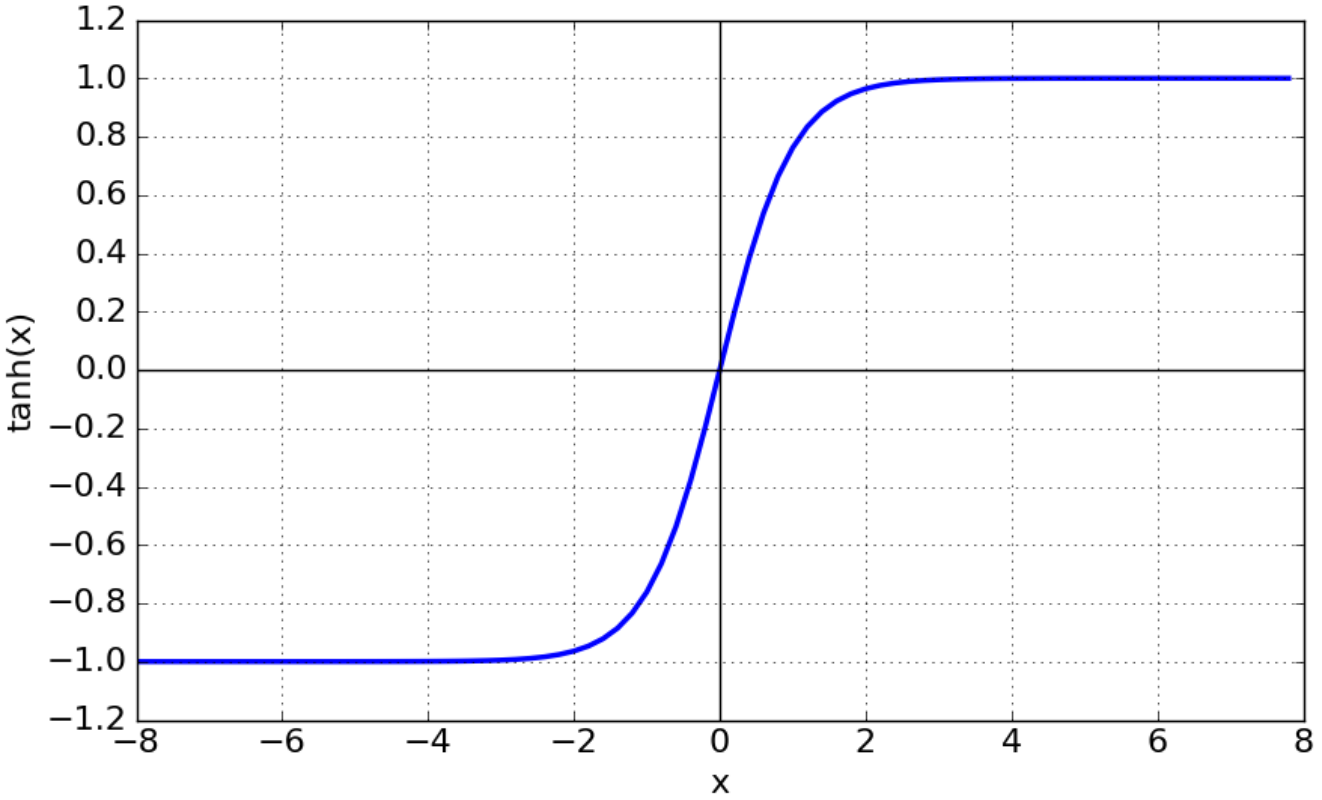
따라서 모두 양수 혹은 음수가 됨

최적화 경로가 지그재그로 길어짐



Hidden Hyperbolic Tangent

하이퍼볼릭 탄젠트 함수 (Tanh Function)

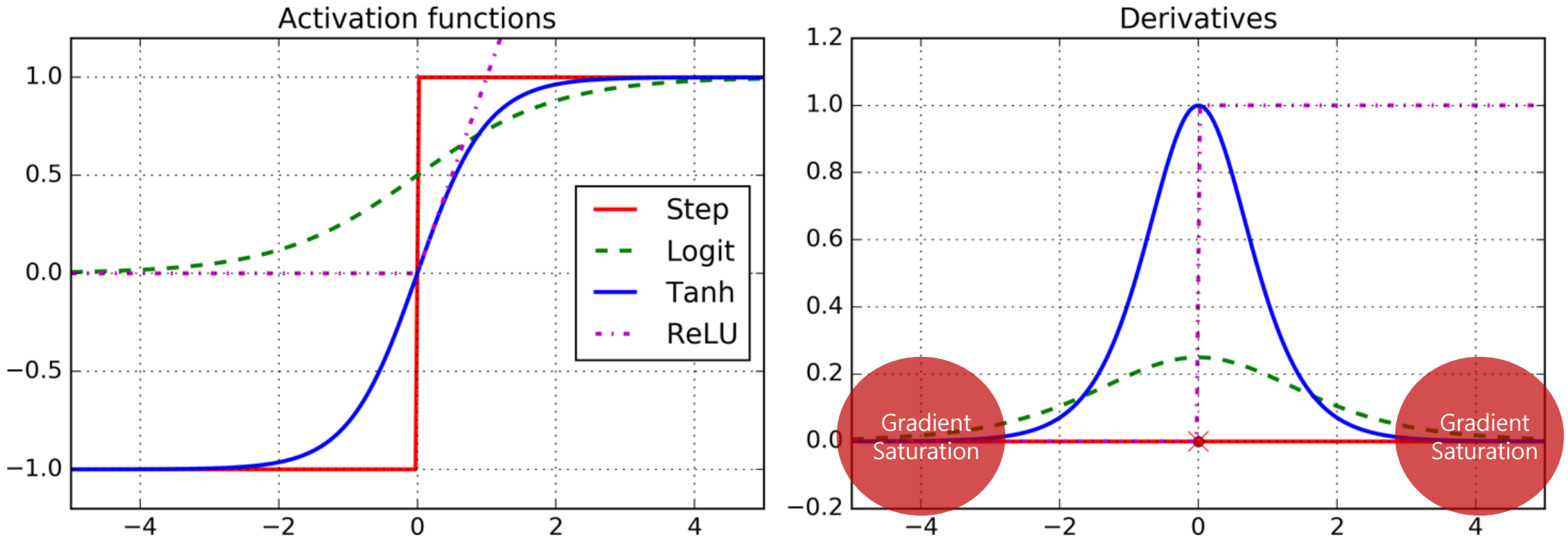


$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- -1과 1사이의 값을 가짐
- Zero-centered
- Identity Function과 유사
- exp 계산 비용이 비쌈

Hidden Hyperbolic Tangent

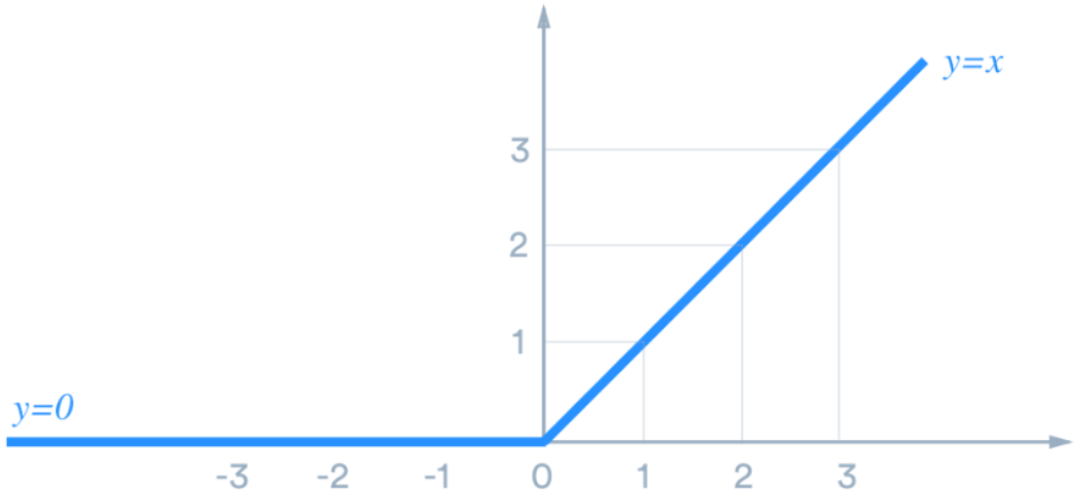
Sigmoidal Function의 미분



Gradient Saturation 문제가 발생!

Hidden Rectified Linear Unit

ReLU (Rectified Linear Unit)



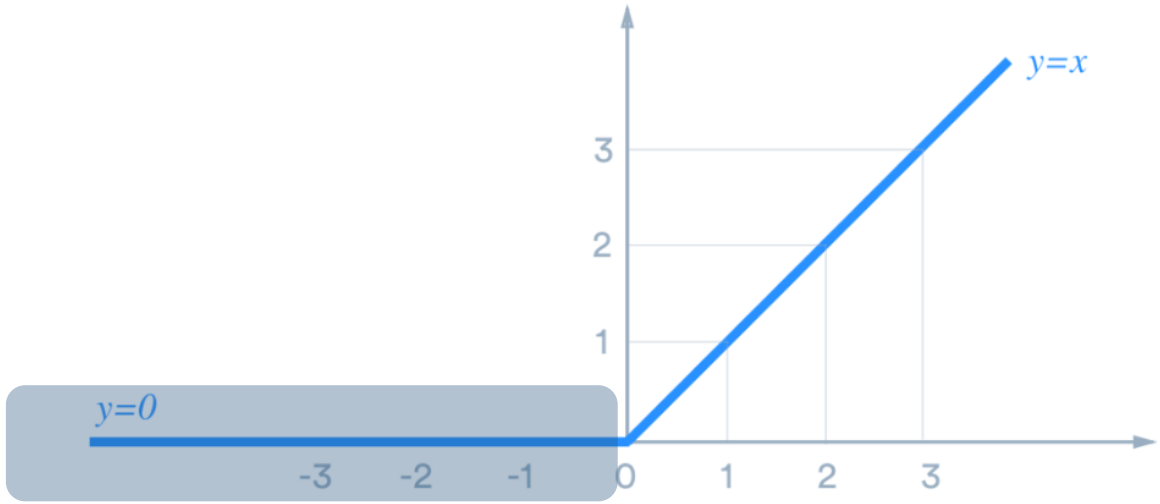
- 2012년 AlexNet에서 처음 사용
- 생체 뉴런의 발화 방식과 Sigmoid보다 더 유사

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Linear Unit과 비슷하기 때문에 최적화가 매우 잘됨
- Sigmoid, Tanh보다 6배 정도 학습이 빠름
- Active 상태에서 Gradient가 크기 때문에 Gradient Vanishing 문제가 생기지 않음
- 단, Dead ReLU와 Not Zero-centered 문제가 있음

Hidden Dead ReLU 문제

Dead ReLU



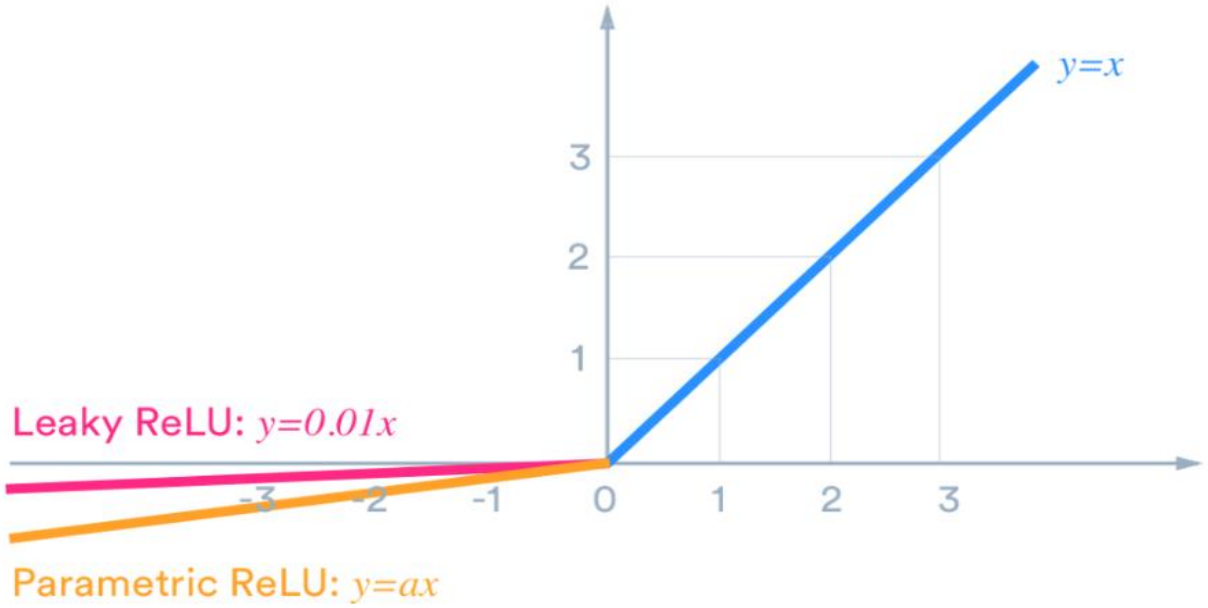
**음수 구간에도
아주 작은 Activation을
만들어 학습이 되게 하자**

"음수에서 Gradient 계산 되지 않음"

- $x < 0$ 구간에서 Activation이 0가 되면 Gradient도 0이 되어 학습이 진행되지 않음
- 초기화를 잘못하거나 Learning Rate가 매우 클 때 발생
- 테스트 데이터의 10~20%가 Dead ReLU면 문제가 될 수 있음

Hidden Leaky ReLU

Leaky ReLU와 PReLU

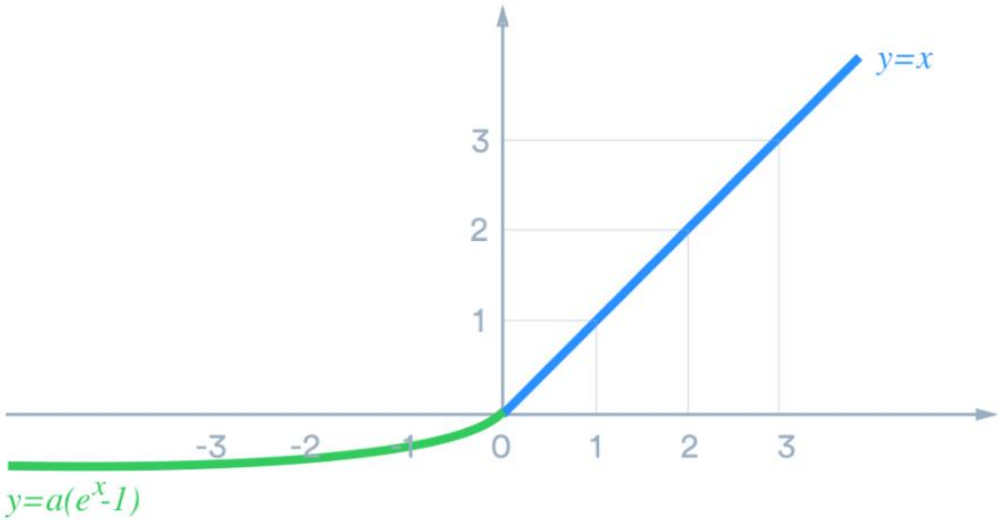


$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

- 음수 구간에 gradient가 생김으로써 훈련 속도가 빨라짐

Hidden ELU (Exponential Linear)

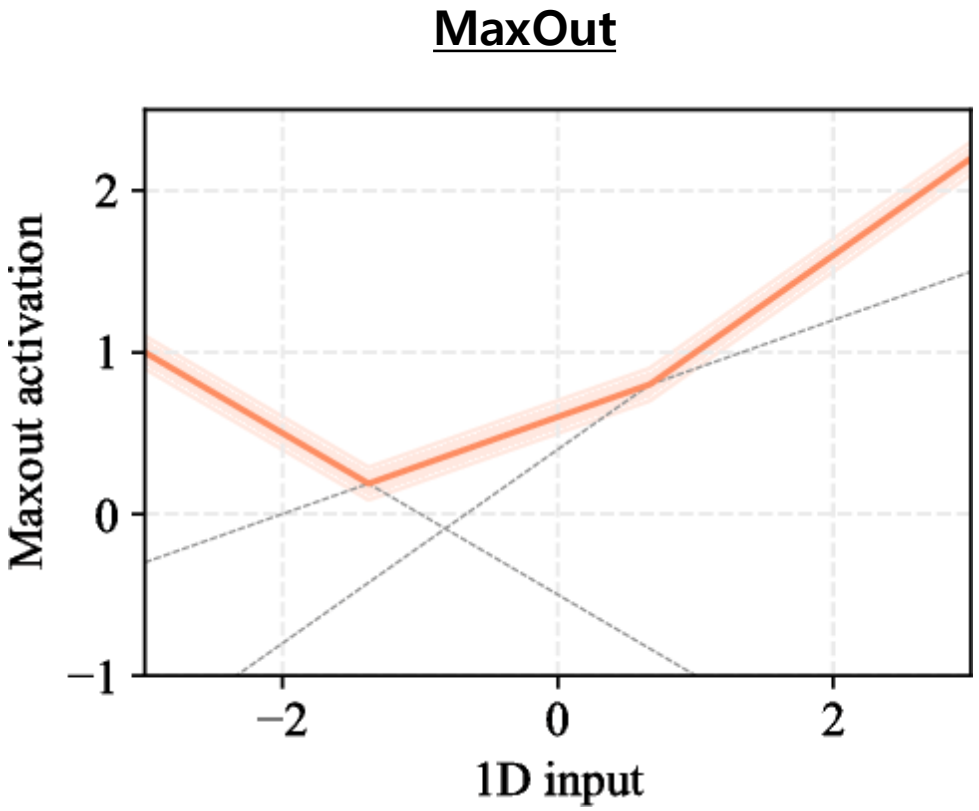
ELU (Exponential Linear)



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases}$$

- α는 학습을 통해 정해짐
- Zero mean output
- 음수 구간에서 saturation이 되어 noise robustness가 좋아짐

Hidden MaxOut

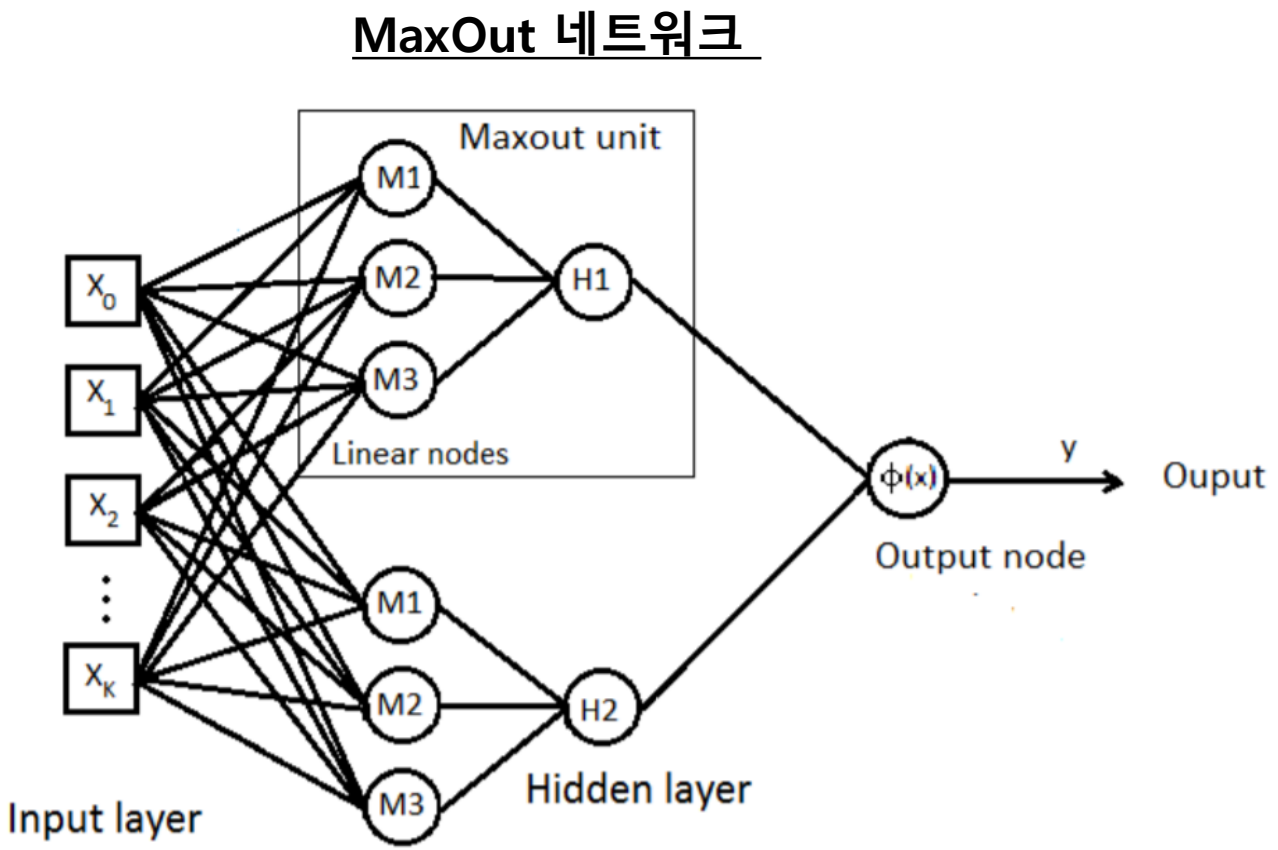


"Activation function을 학습해보자"

$$f(x) = \max\ (\boldsymbol{W}_i^T x + b_i)$$

- Rectified Linear Unit을 일반화한 Piecewise Linear Function 학습
- 매우 뛰어난 성능
- 여러 가중치와 편향에 대해 값을 구해서 최대 값을 취하는 방식으로 구간 별 최대 Linear Activation을 선택하는 효과

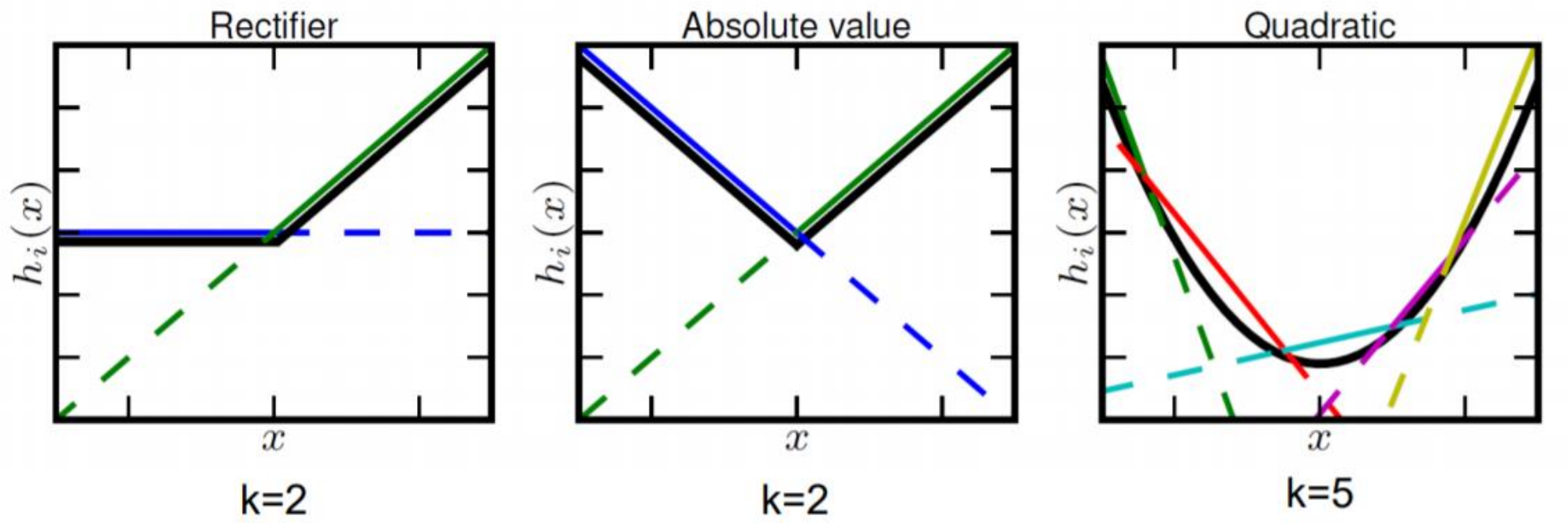
Hidden MaxOut



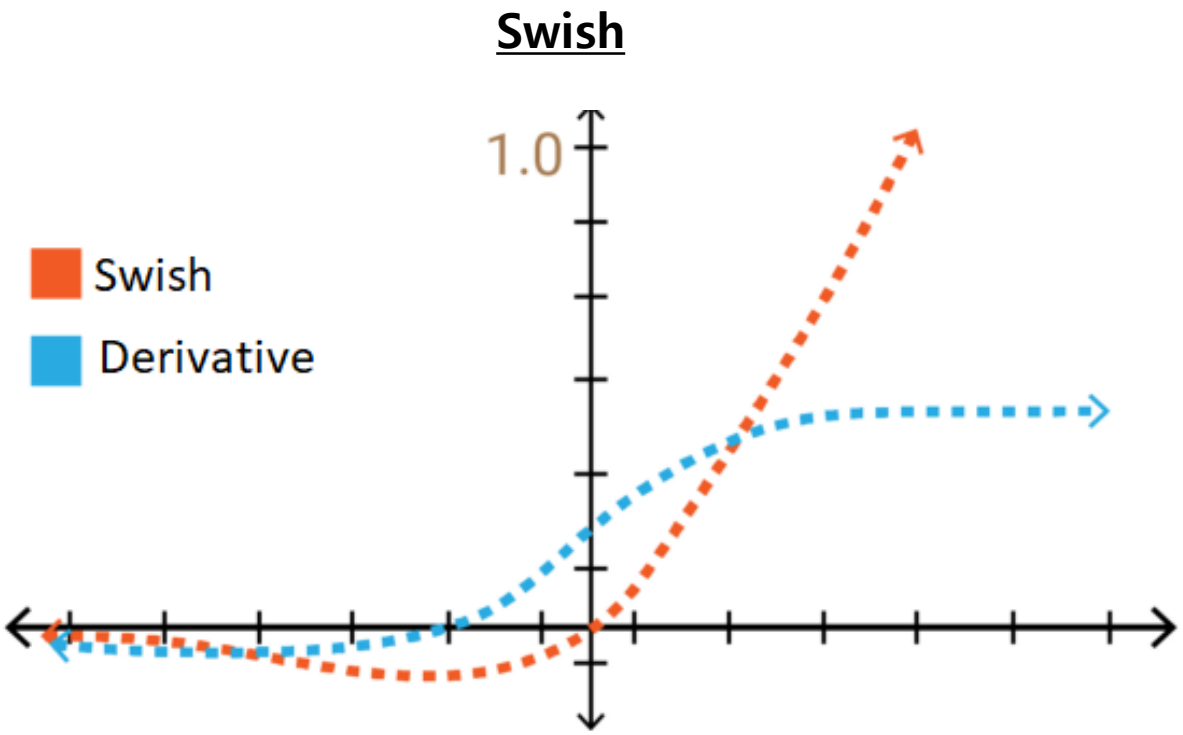
두 개의 MaxOut 유닛과 각각에 세개의 Linear Node로 이뤄진 MaxOut 네트워크

Hidden MaxOut

임의의 convex 함수를 근사



Hidden Swish

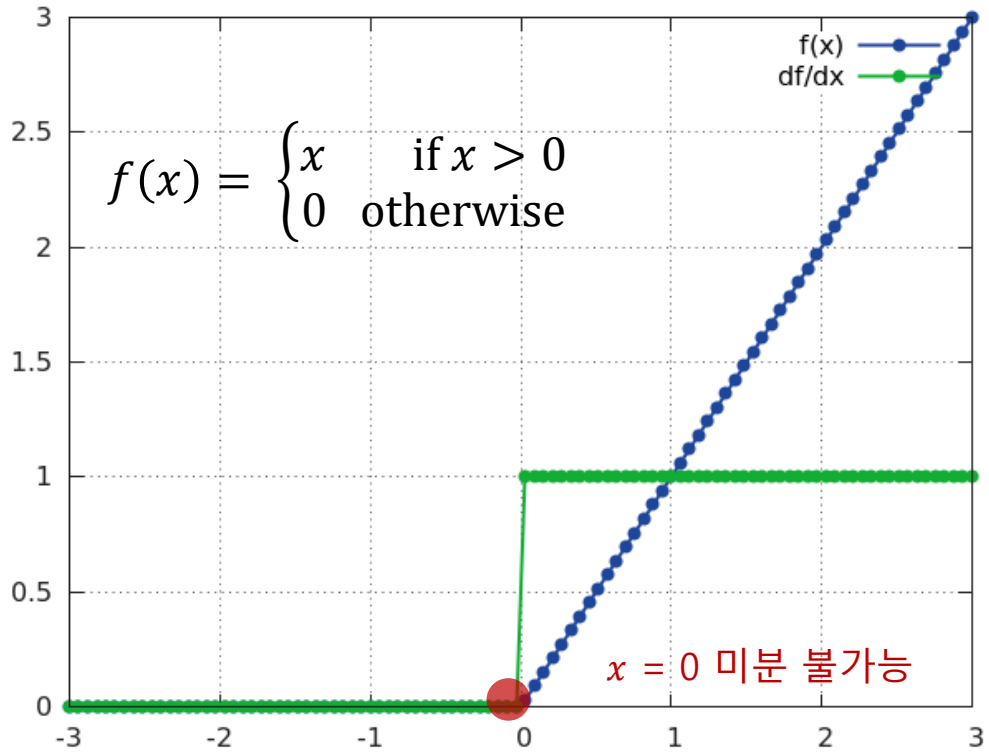


$$f(x) = x \cdot sigmoid(x)$$

- Google Brain의 AutoML로 찾은 Optimal Activation Function
- SELU(Scaled Exponential Linear Units)와 GELU(Gaussian Error Linear Units)등과 유사

Hidden Activation Function 미분

ReLU



Activation Function이 미분 가능하지 않으면 Gradient Descent는 사용할 수 없나?

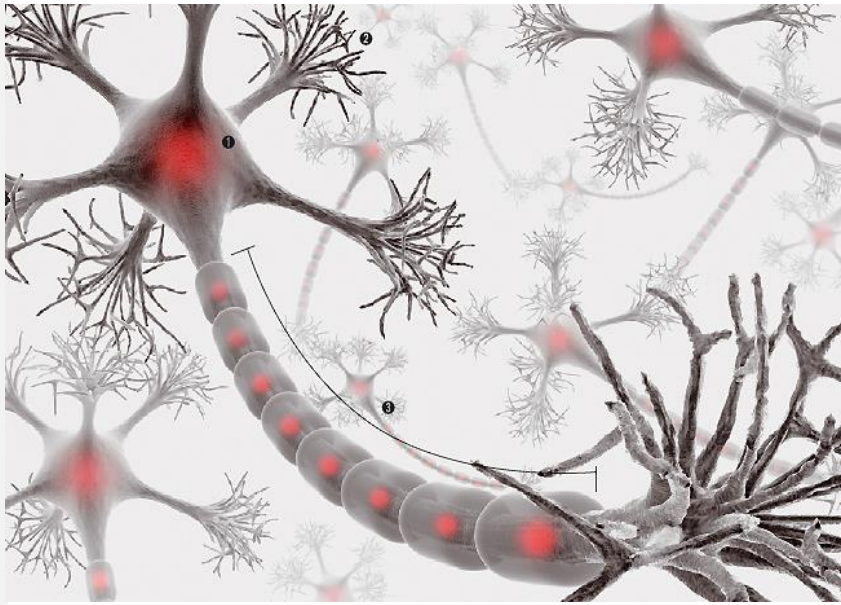
- $x = 0$ 근처에서 $x > 0$이면 미분을 1로 사용하고 $x < 0$이면 0으로 사용
- 이 정도의 미분 오차는 신경망에서 허용가능

구간 별로 미분해서 Gradient Descent를 적용!

Hidden Summary

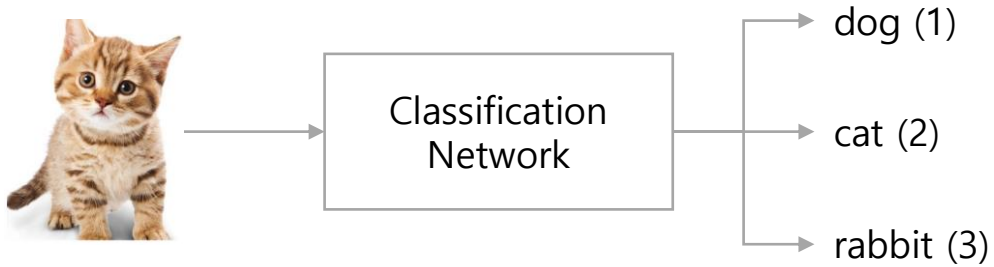
- ReLU를 사용하되 Learning Rate를 잘 선택하라.
- Leaky ReLU / Maxout / ELU / Swish 등의 ReLU Family는 사용하면 좋다.
- Tanh는 사용하도 되지만 그다지 유용하지 않다.
- Sigmoid 는 절대 사용하지 말 것!

4 출력 계층 (Output Layer)



Output 출력 형태

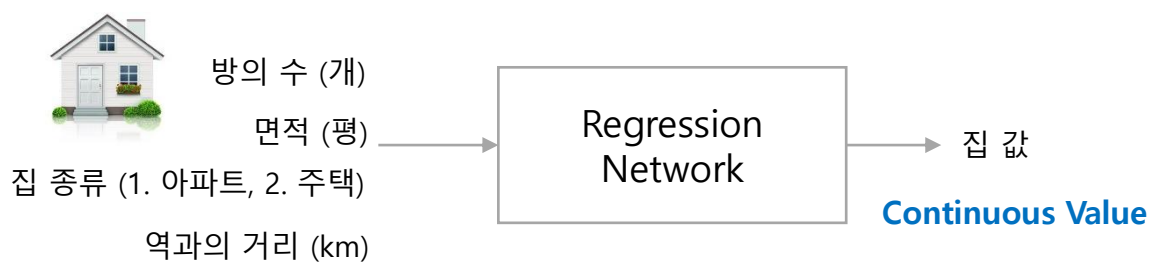
분류 문제 (Classification)



Discrete Value

- **입력 데이터에 대한 클래스를 또는 카테고리를 예측하는 문제**
- 출력은 입력 데이터가 속할 클래스
- **확률 모델** : 입력 데이터가 각 Class에 속할 확률 분포를 예측
  ex) Bernoulli, Categorical Distribution

회귀 문제 (Regression)



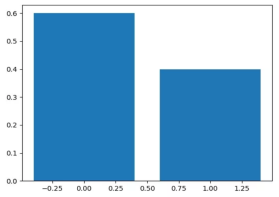
Continuous Value

- **여러 독립 변수와 종속 변수의 관계를 함수 형태로 분석하는 문제**
- 출력은 입력 데이터에 대한 함수 값
- **확률 모델** : 관측 값에 대한 확률 분포를 예측
  ex) Gaussian Distribution

Output Activation Function

분류 문제 (Classification)

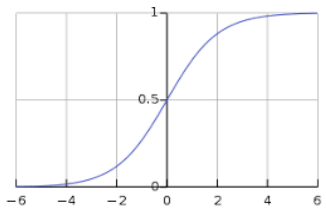
Bernoulli Distribution



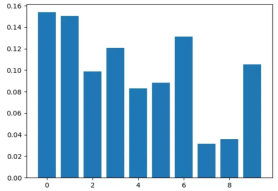
- 2개 카테고리로 분류된 이산 데이터

Sigmoid

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



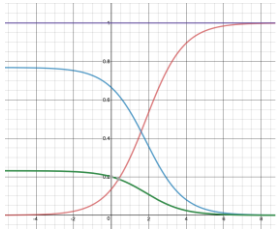
Categorical Distribution



- n개 카테고리로 분류된 이산 데이터

Softmax

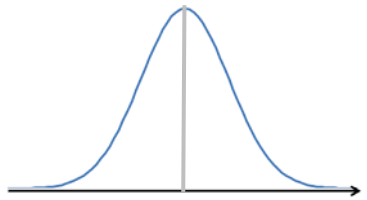
$$f(yi) = \frac{e^{y_i}}{\sum_{j=0}^{N} e^{y_j}}$$



Discrete Case

회귀 문제 (Regression)

Gaussian Distribution

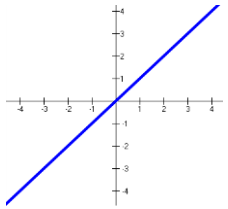


μ

- 연속 데이터는 대부분 가우시안으로 가정

Identity

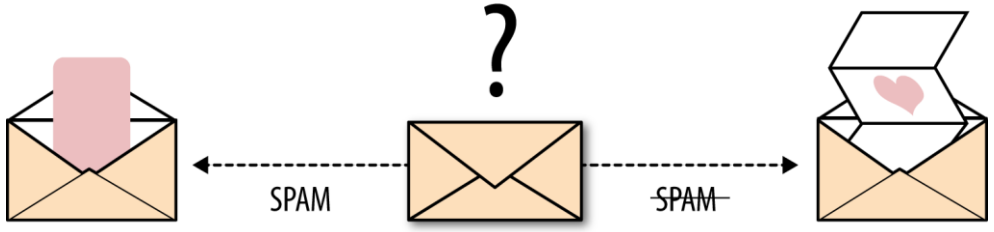
$$f(n) = n$$



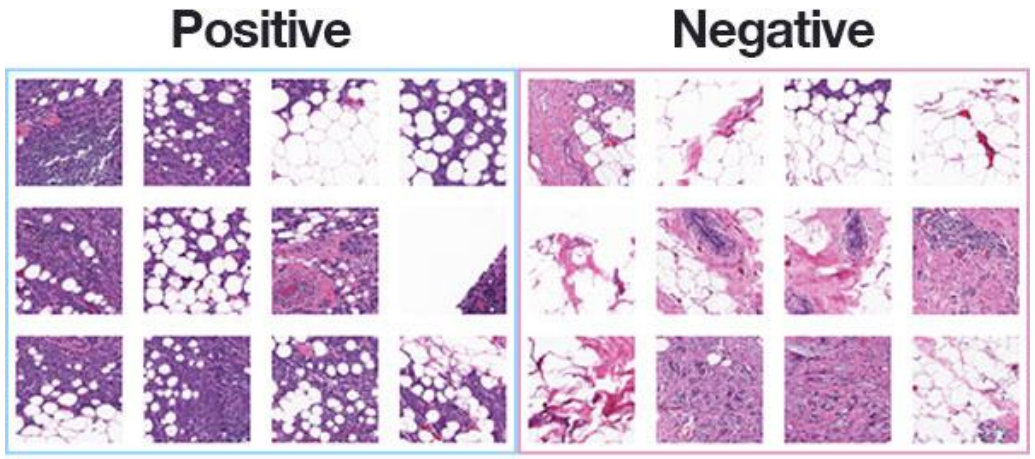
Continuous Case

Output 이진 분류 (Binary Classification)

메일이 스팸일 확률은?



세포 조직이 유방암 양성일 확률은?



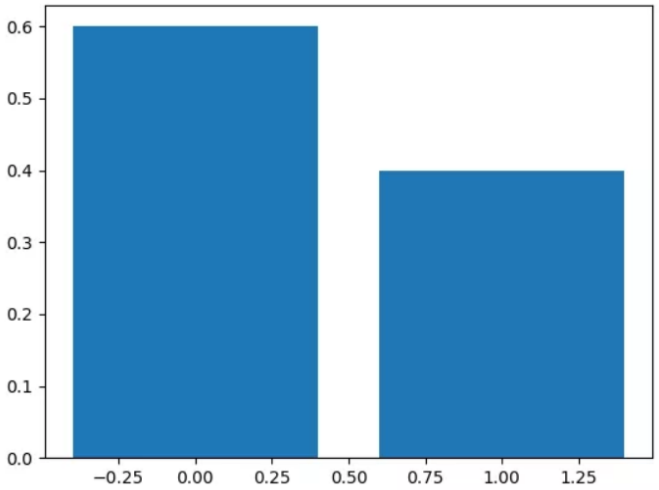
얼굴이 가짜 얼굴일 확률은?

Output 이진 분류 (Binary Classification)

이진 분류는
베르누이 분포를 추정하는 문제!

Bernoulli Distribution

$$p_{model}(y \mid x;\theta) = \mathcal{B}ernoulli(f(x;\theta))$$



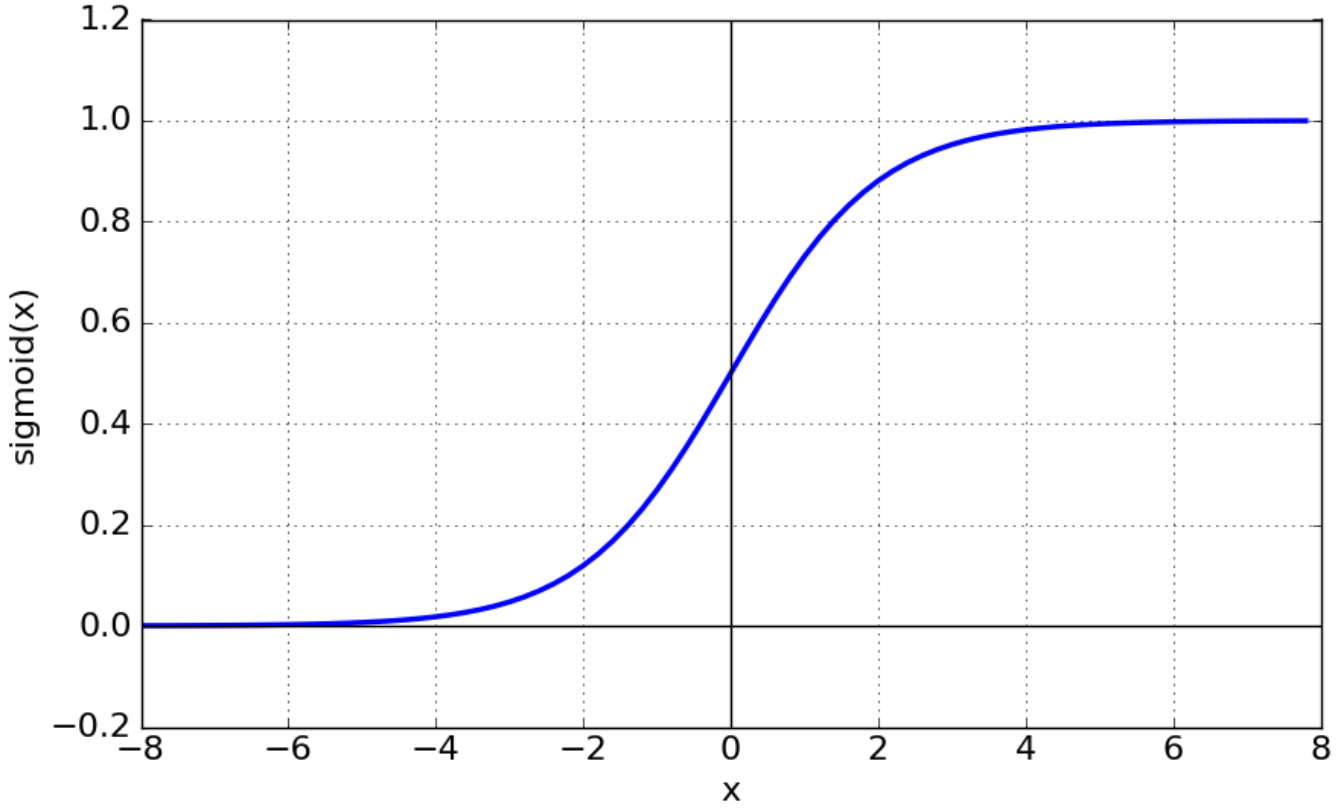
$$p(x\,;\mu) = \mu^x(1-\mu)^{1-x}$$

$$x = 0, 1$$

Output 이진 분류 (Binary Classification)

베르누이 분포를 갖는 출력은 Sigmoid 함수로 만들 수 있다!

시그모이드 함수 (Sigmoid Function)



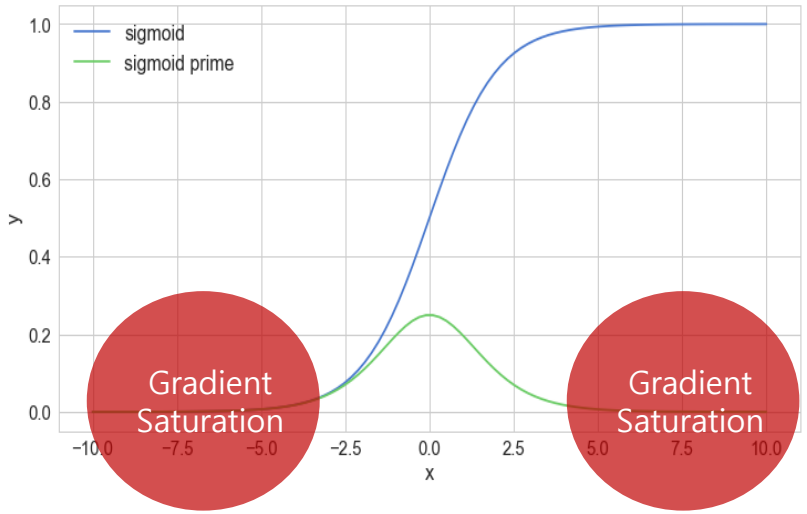
$$f(x) = \frac{1}{1 + e^{-x}}$$

- 입력 값을 확률 값으로 변환해 줌
- 값이[0, 1] 사이에 존재
- Cross Entropy Loss와 함께 사용

Output 이진 분류 (Binary Classification)

Gradient Saturation 문제 방지하려면 Cross-Entropy Loss와 함께 사용하라!

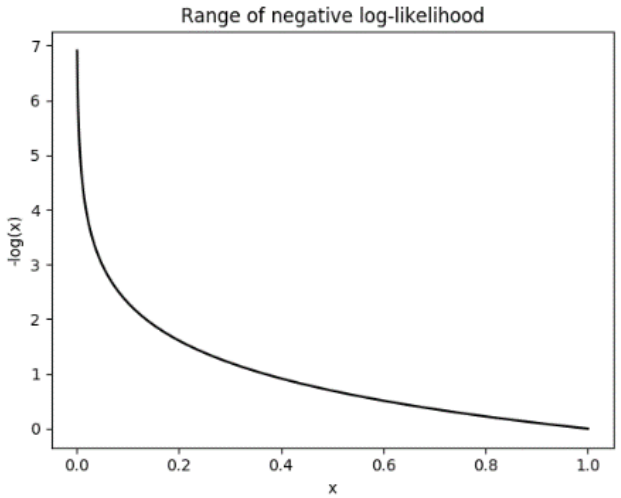
Sigmoid



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

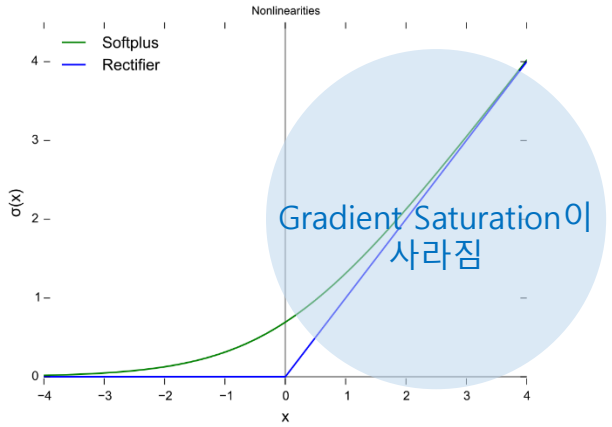
+

Cross-Entropy



$$f(x) = -\log(x)$$

Softplus



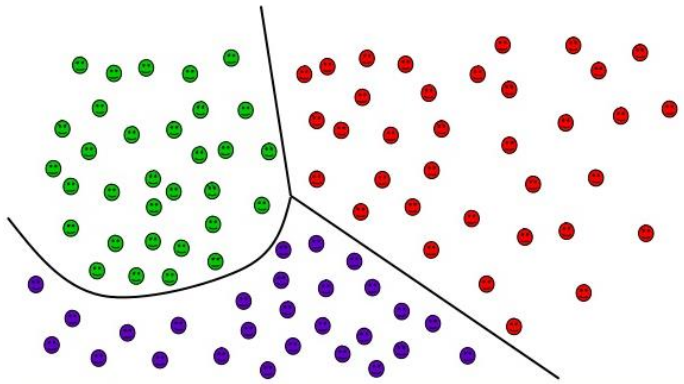
$$f(x) = -\log\big(\sigma(x)\big)$$
$$= \log(1+e^{-x})$$
$$= \text{softplus}(-x)$$

$$\text{softplus}(x) = \log(1+e^{x})$$

- Cross Entropy를 사용하면 log 함수가 exp를 상쇄

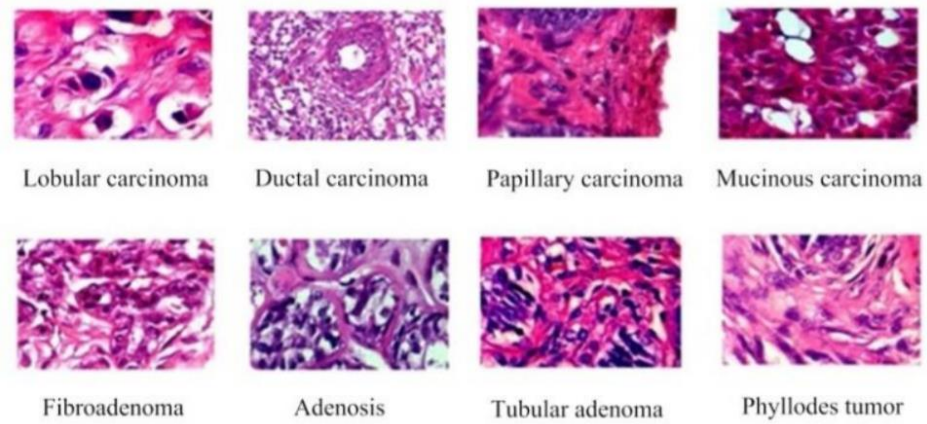
Output 다중 분류 (Multiclass Classification)

데이터가 각 Class에 속할 확률은?

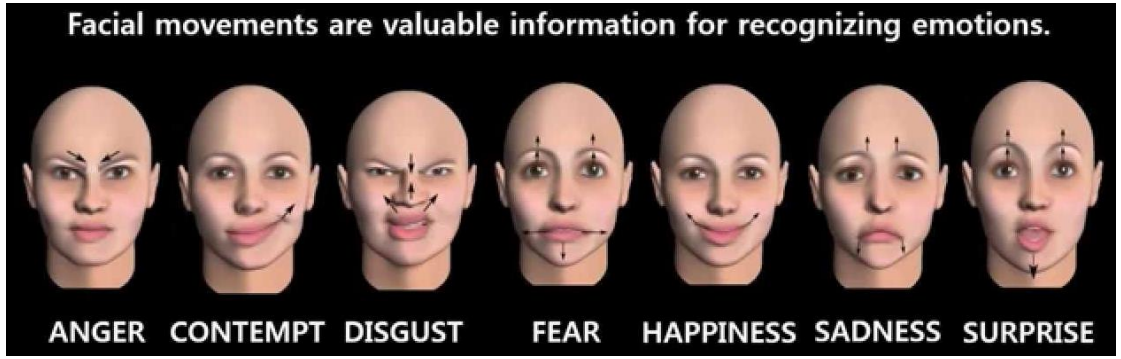


이미지 인식

유방암 종류 인식



감정 인식



https://www.kaggle.com/ashishpatel26/tutorial-facial-expression-classification-keras

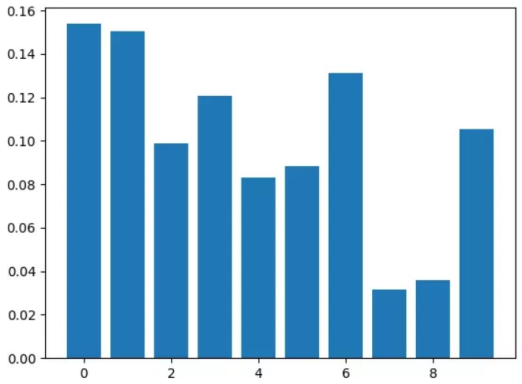
Output 다중 분류 (Multiclass Classification)

다중 분류는
카테고리 분포를 추정하는 문제!

베르누이 분포를 2개 결과에서
m개 결과로 일반화한 분포

Categorical Distribution

$$p_{model}(y \mid x;\theta) = Categorical(f(x;\theta))$$



$$p(x \mid \mu) = \prod_{k=1}^{K} {\mu_k}^{x_k}$$

$$x = (x_1, x_2, \ldots, x_K)^T \qquad \mu = (\mu_1, \mu_2, \ldots, \mu_K)^T$$

$$x_k = \begin{cases} 1, k = i \\ 0, k \neq i \end{cases} \quad i \in \{1, 2, \ldots, K\} \qquad \sum_{k=1}^{K} \mu_k = 1$$

i번째 Category에 속할 경우

Output 다중 분류 (Multiclass Classification)

Categorical Distribution을 갖는 출력은 Softmax 함수로 만들 수 있다!

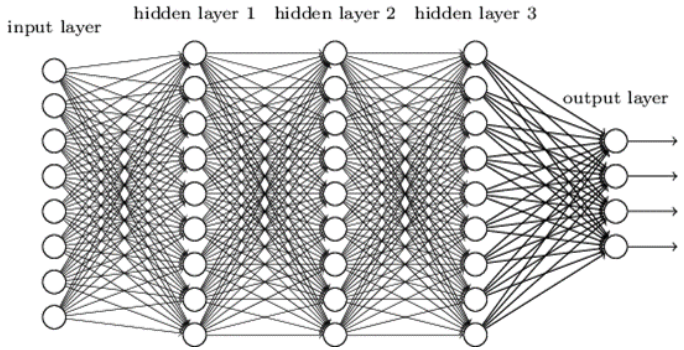
Score 값을 확률로 변환

Softmax

$$f(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{N} e^{y_j}}$$

Input

Network



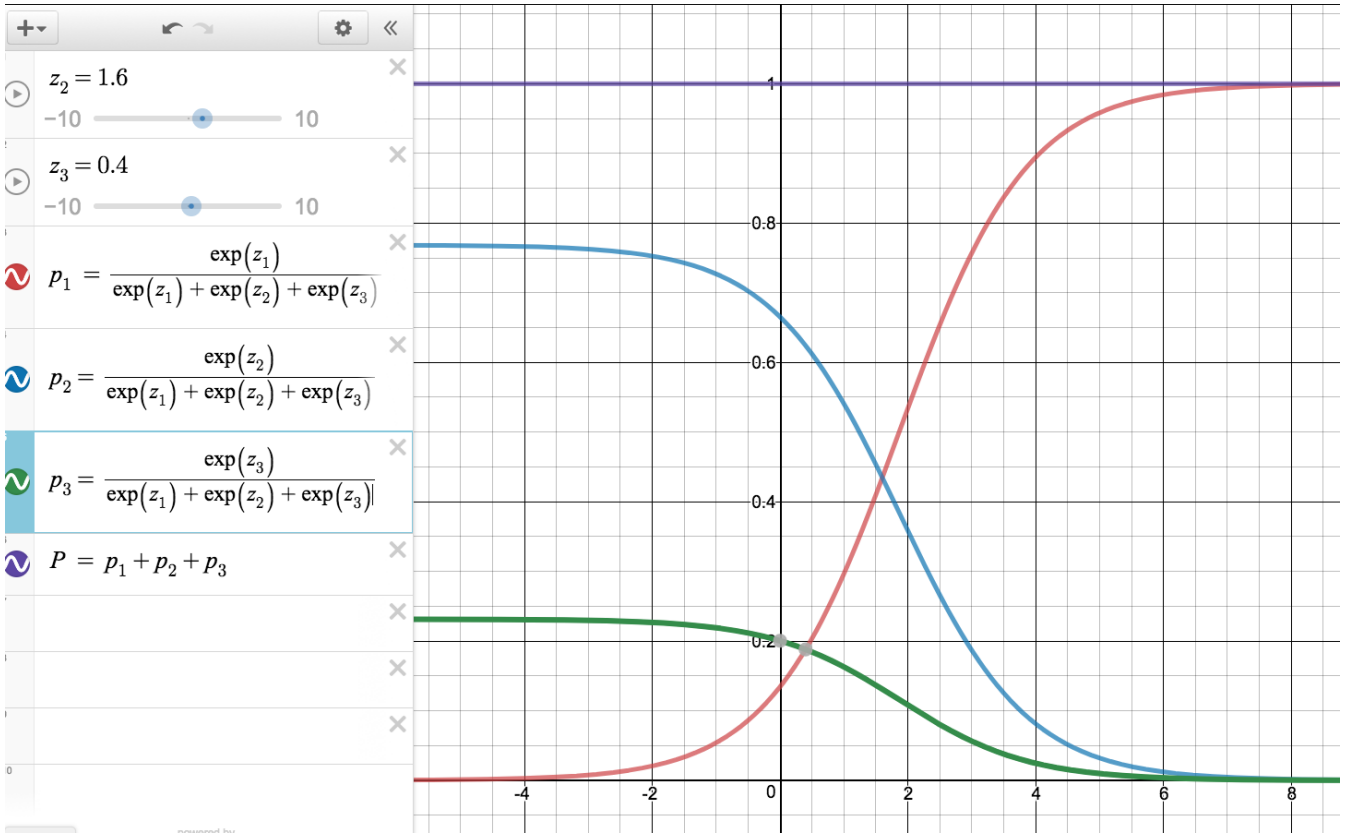
Output

3.7	다람쥐	0.1	다람쥐
12	고양이	0.7	고양이
4.5	강아지	0.2	강아지
...	...	...	...

scores/logits **probability**

Output 다중 분류 (Multiclass Classification)

Softmax Function

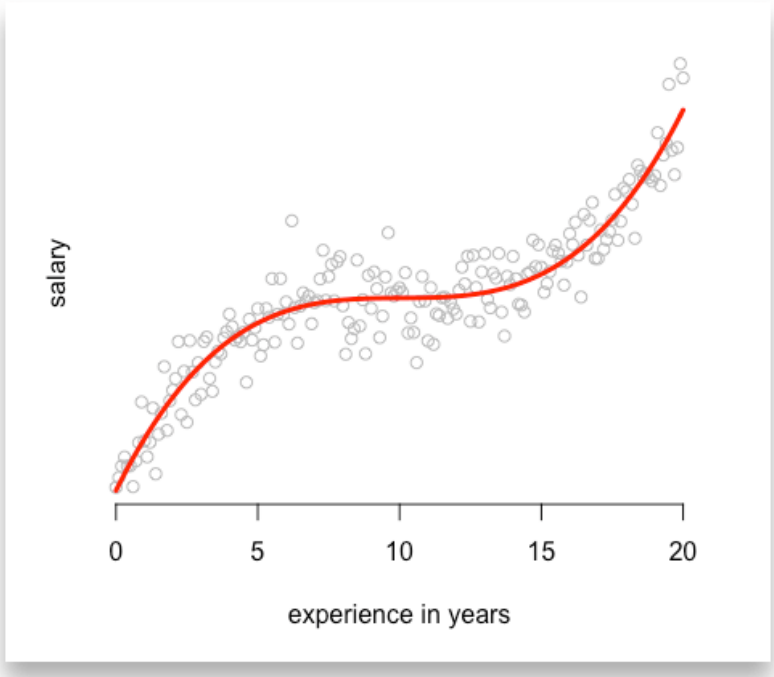


$$f(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{N} e^{y_j}}$$

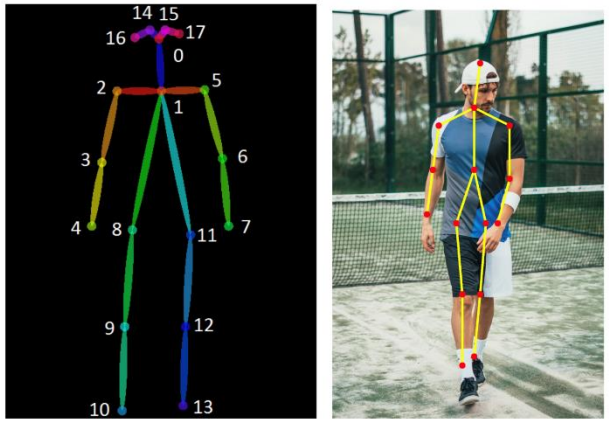
- 값이[0, 1] 사이에 존재
- 입력 값을 확률 값으로 변환해 줌
- n개 카테고리 분류에 사용
- Cross Entropy Loss와 함께 사용

Output 연속 함수 값 예측 (Regression)

입력 값에 대한 연속 함수의 함수 값을 예측

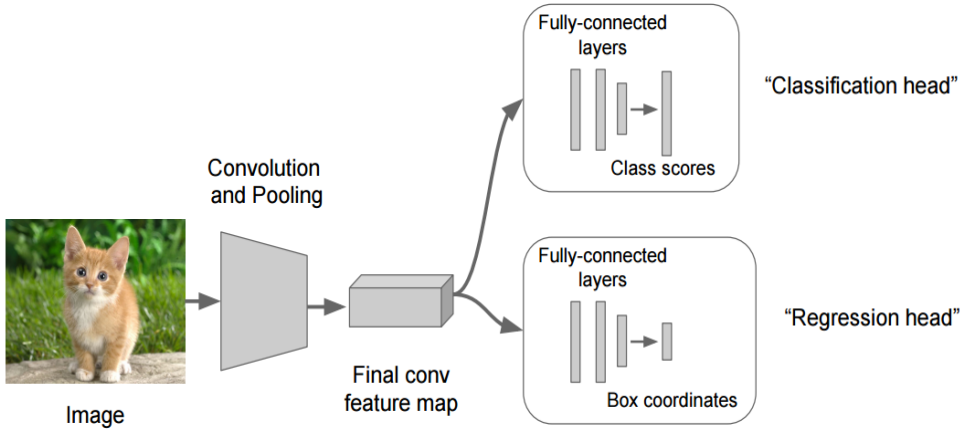


Human Pose Estimation



https://blog.nanonets.com/human-pose-estimation-2d-guide/

Object Detection에서 Bounding Box 예측



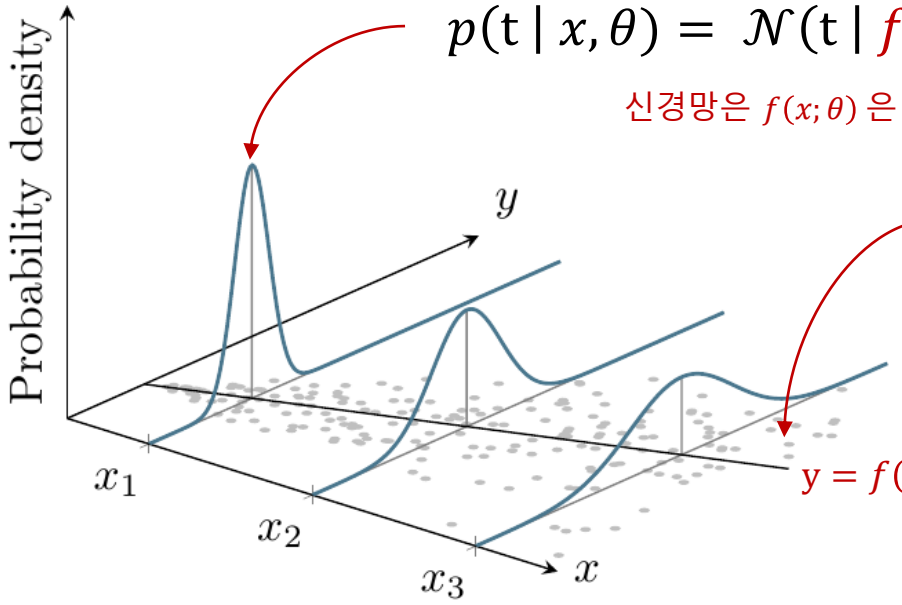
Output 연속 함수 값 예측 (Regression)

연속 함수 값을 예측하는 것은
타깃의 분포를 예측하는 것!

타깃 t의 확률 분포:

$$p(\mathrm{t} \mid x, \theta) = \mathcal{N}(\mathrm{t} \mid f(x;\theta), \beta^{-1})$$

신경망은 $f(x;\theta)$ 은 가우시안 분포의 평균을 예측



회색 점들 : 타깃 t

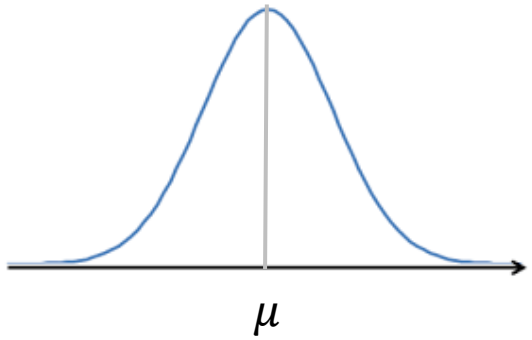
$$\mathrm{t} = f(\boldsymbol{x};\theta) + \epsilon$$

관측 오차

$$\epsilon \sim \mathcal{N}(\epsilon \mid 0, \beta^{-1})$$

$\mathrm{y} = f(x;\theta)$ 신경망 예측 값

Gaussian Distribution



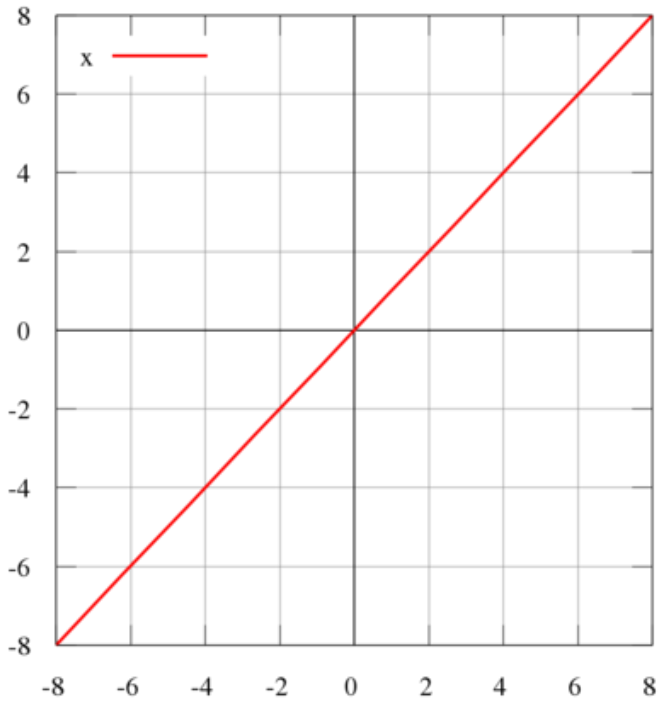
$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- 관측 값은 실제 함수 값에 노이즈를 더해진 값
- 노이즈가 정규분포를 따르면 타깃도 노이즈의 분포를 따르게 됨

Output 연속 함수 값 예측 (Regression)

연속 함수의 출력은 값을 바꾸지 않는 Identity Function을 사용한다!

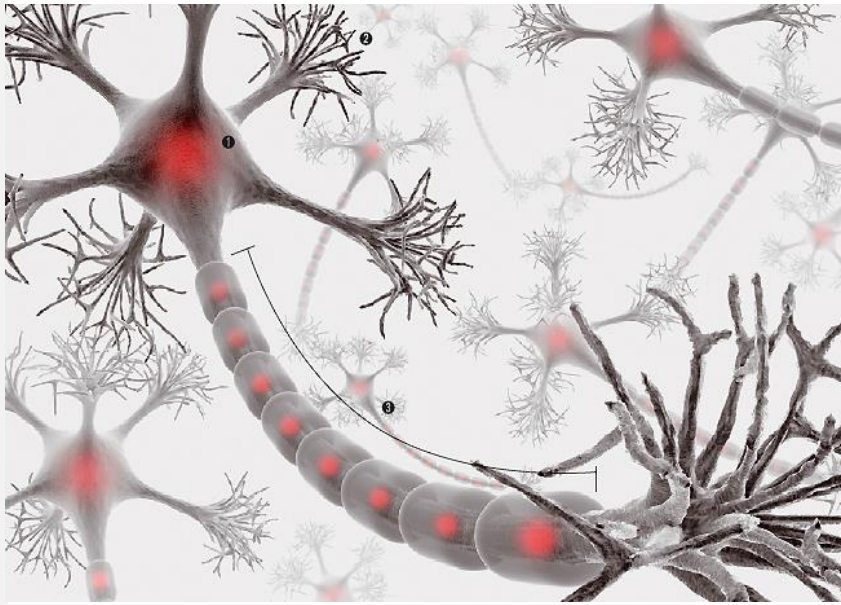
Identity Function



$$f(n) = n$$

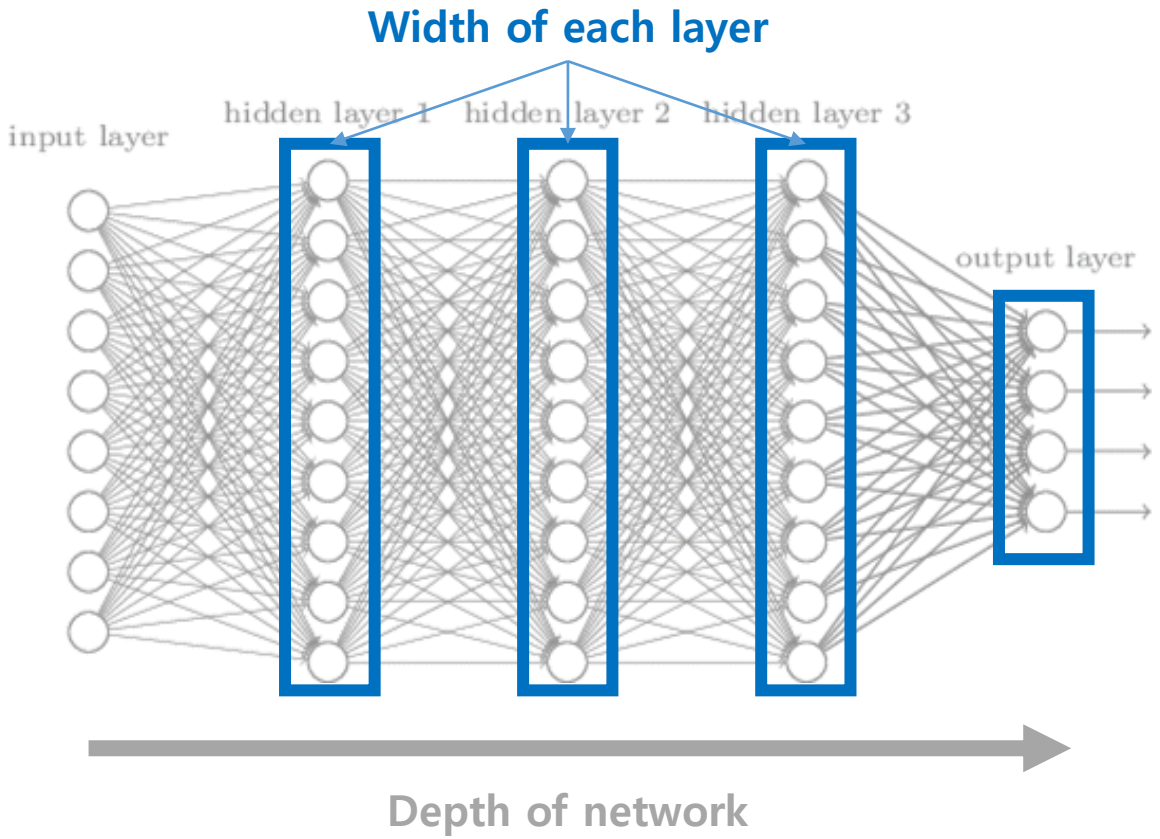
- 값을 그대로 통과시킬 때 사용함
- 최적화가 매우 쉽고 광범위하게 사용될 수 있음
- 실제 구현할 때는 Activation Function을 적용하지 않음

4 네트워크 크기



Network Size 네트워크 크기 설계

Network의 Depth와 Width를 어떻게 정할 것인가?



네트워크 깊이가 깊을 수록

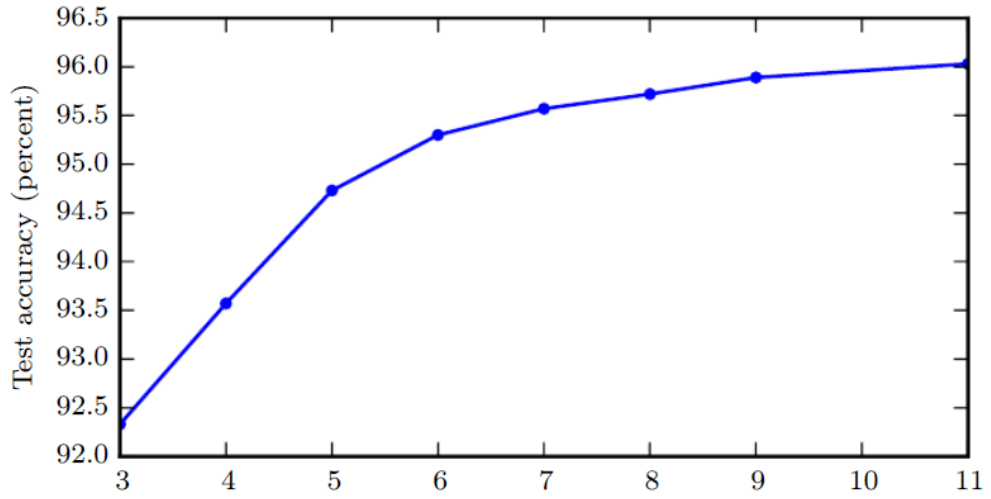
- 계층 별 뉴런을 적게 사용
- 적은 파라미터를 사용
- 일반화를 잘 함

하지만, 최적화가 어렵다.

문제에 따라 최적의 네트워크 구조는
Trial & Error로 찾아내야 한다!

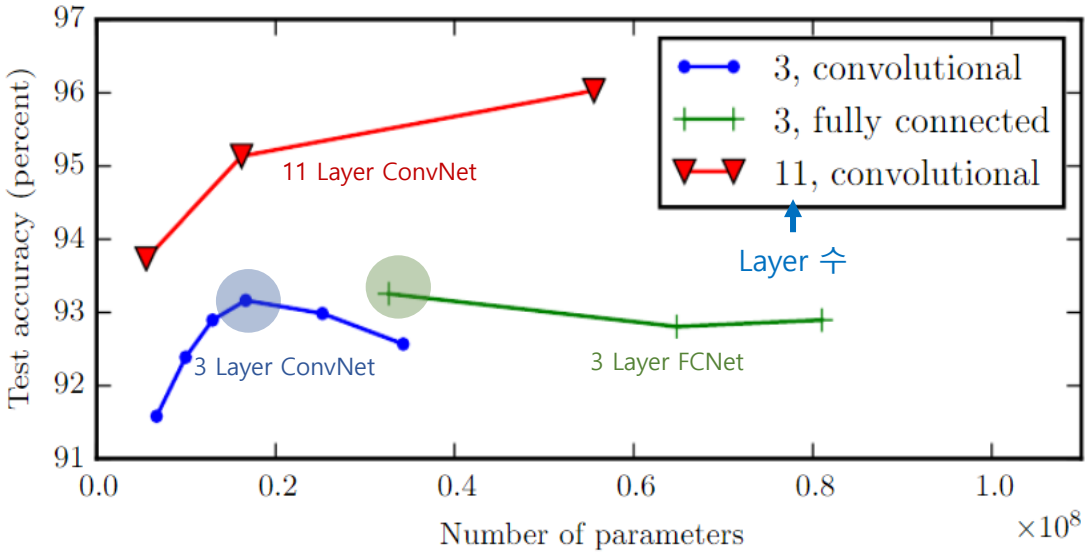
Network Size 네트워크 깊이와 모델 크기

Depth vs. Accuracy



- 네트워크 깊이가 깊을 수록 일반화를 잘 함

Width vs. Accuracy

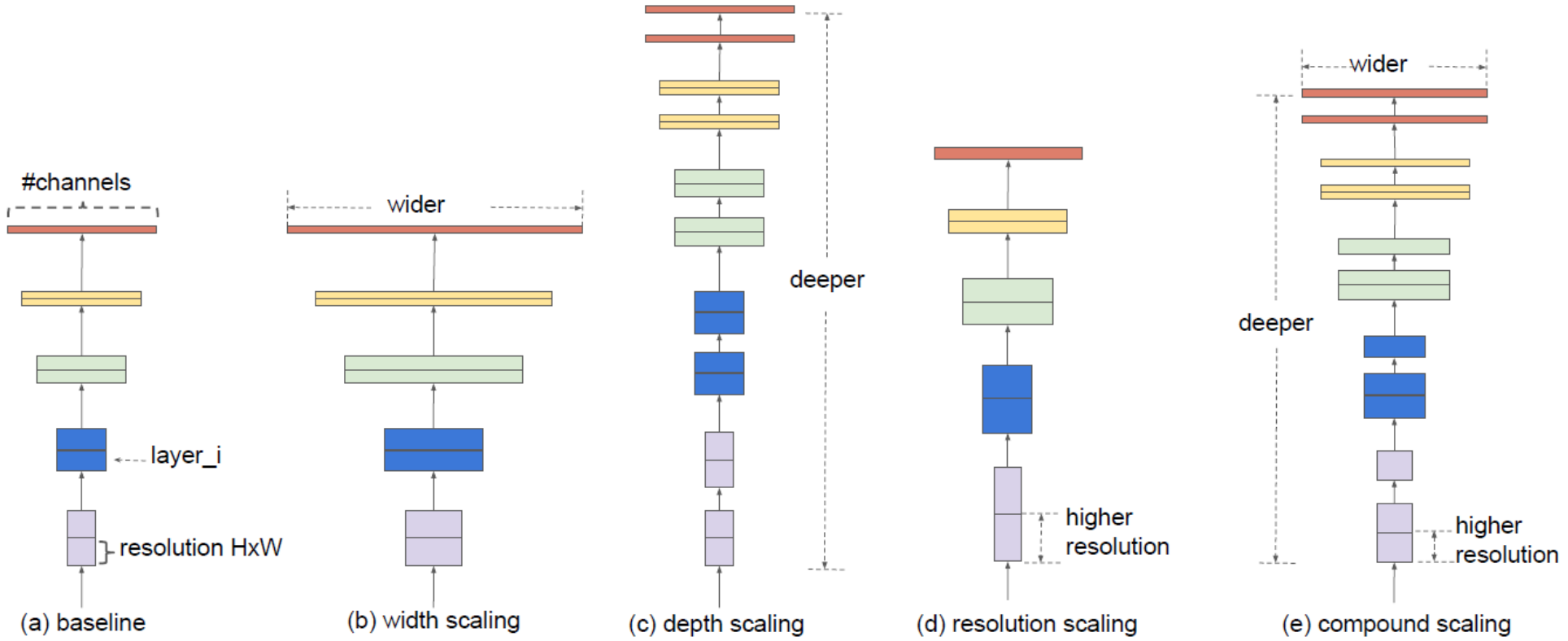


- 네트워크 깊이를 높이지 않고 파라미터 수만 늘리면 성능이 오히려 떨어지게 됨

Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks
Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet (2013)

Network Size 네트워크 깊이와 모델 크기

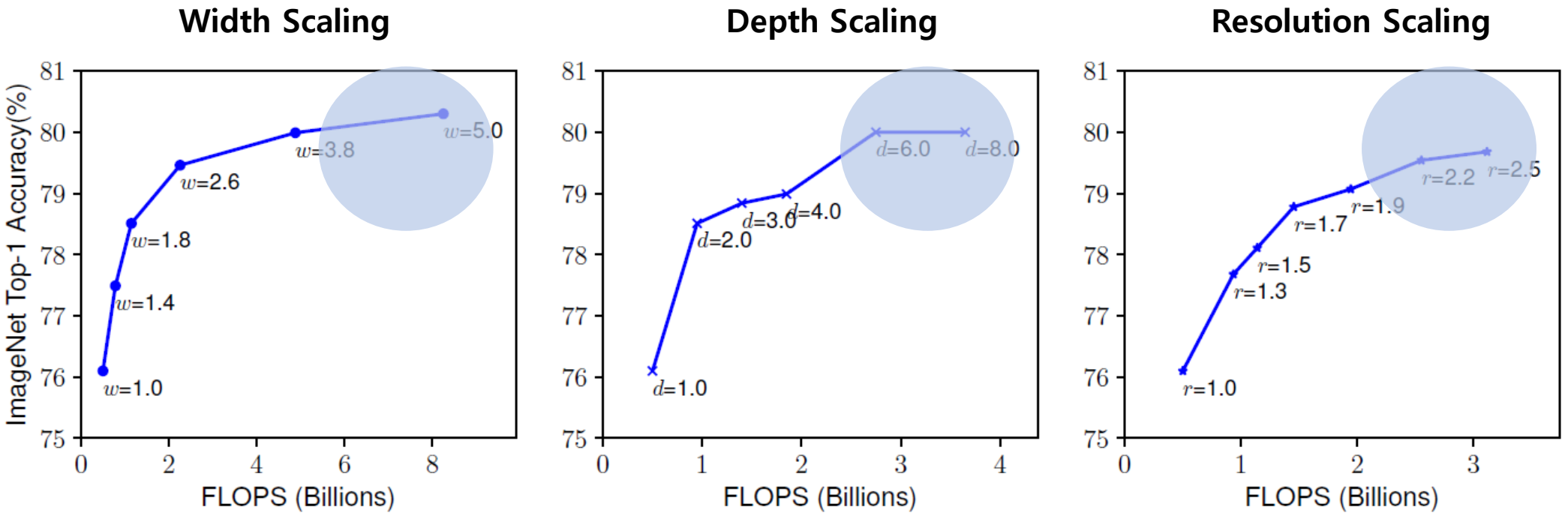
Model Scaling (Width, Depth, Resolution, Compounding)



EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Mingxing Tan Quoc V. Le (2019)

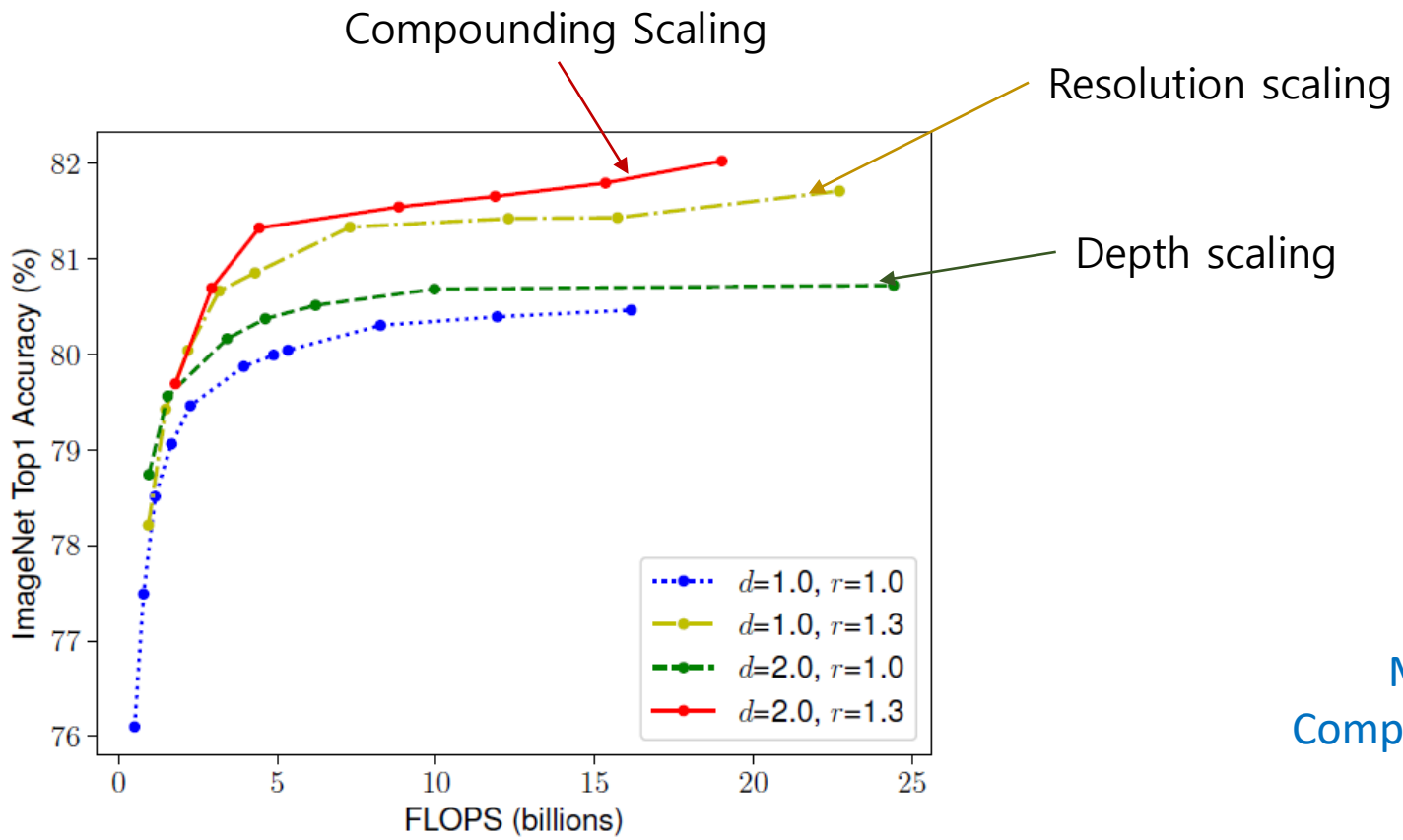
Network Size 네트워크 깊이와 모델 크기

모델이 커질수록 accuracy는 올라가지만 쉽게 saturation됨



EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Mingxing Tan Quoc V. Le (2019)

Network Size 네트워크 깊이와 모델 크기



Model Scaling을 할 경우엔
Compounding Scaling이 가장 효과적

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Mingxing Tan Quoc V. Le (2019)

Thank you!