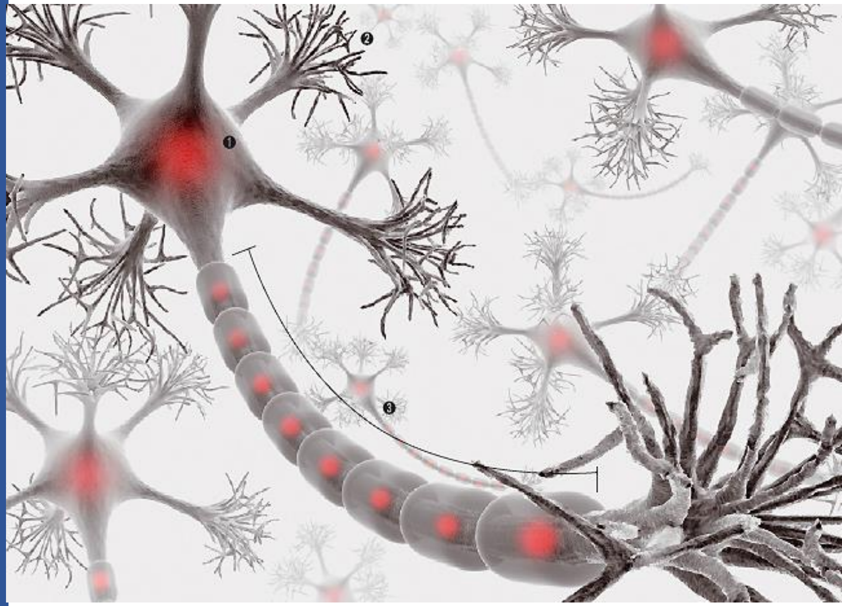
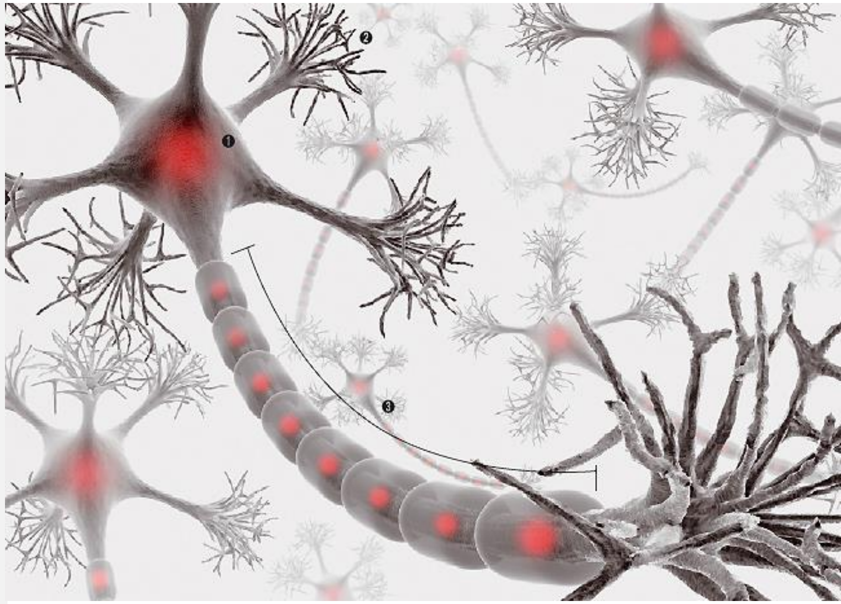
[복습] 순방향 신경망, 신경망 학습, TensorFlow



학습 목표

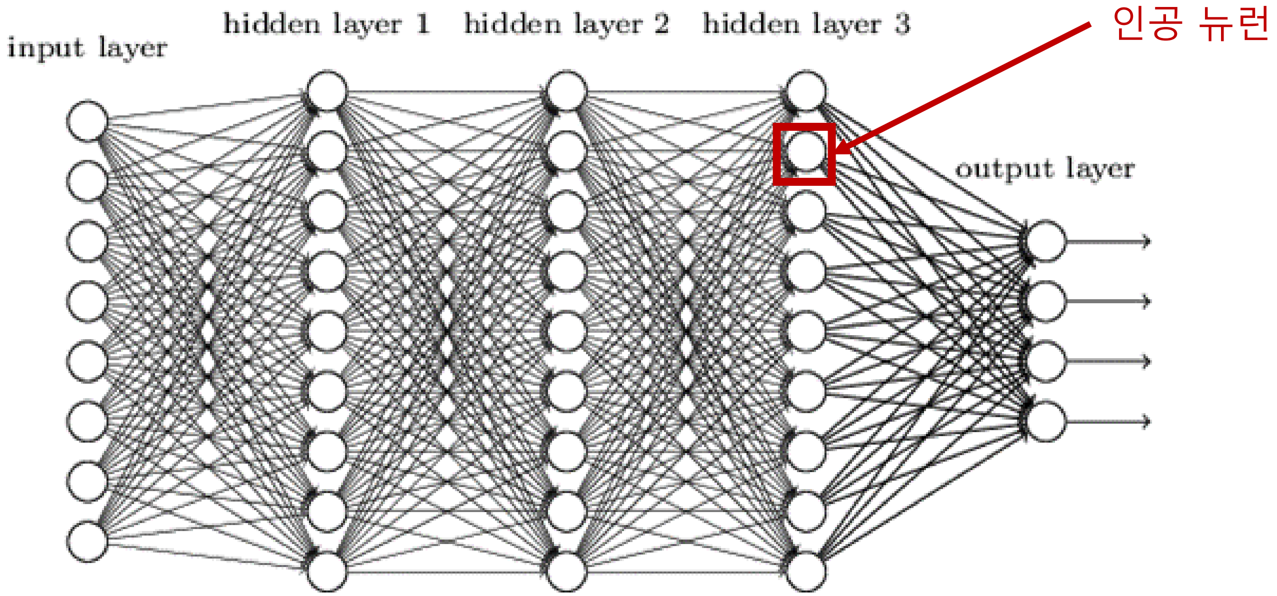
- Day1, 2에 배웠던 내용들을 복습해본다.

1 함수로서의 신경망



인공 신경망

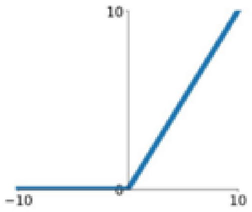
<u>인공 신경망 (Artificial Neural Network)</u>



- 생체 신경망의 작동 원리를 모방해서 만든 인공 신경망
- 인공 뉴런들이 서로 복잡하게 연결되어 있는 네트워크
- 뉴런은 신호를 받아서 임계치 이상이 되면 신호를 발화

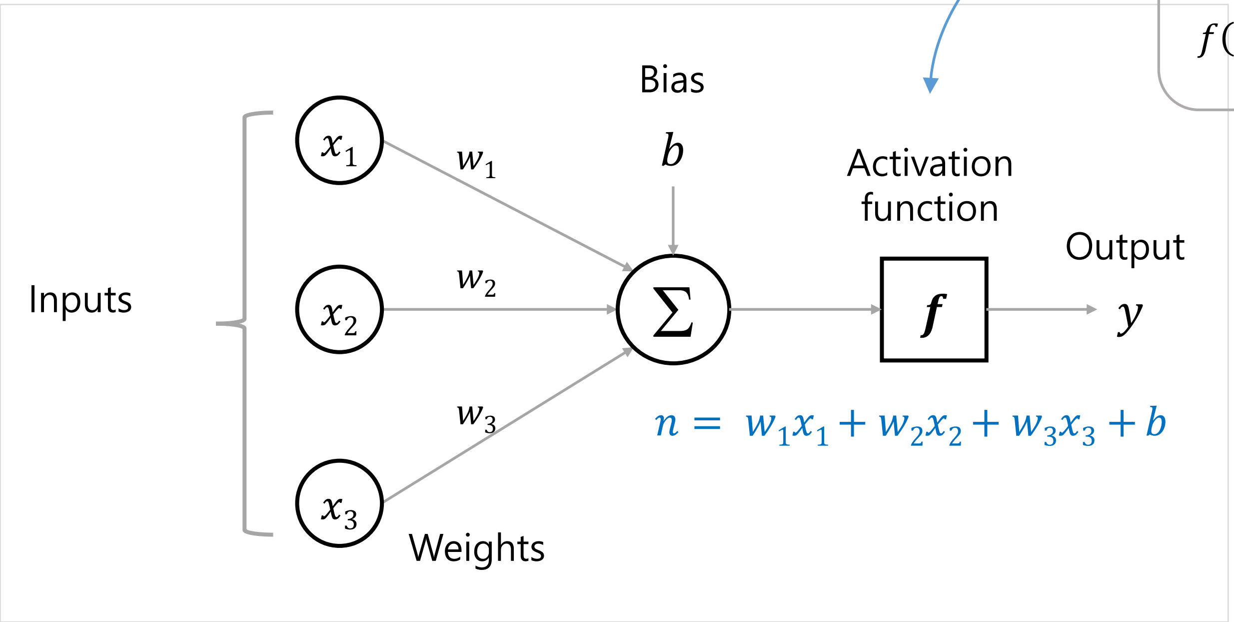
인공 신경망

Activation function : ReLU



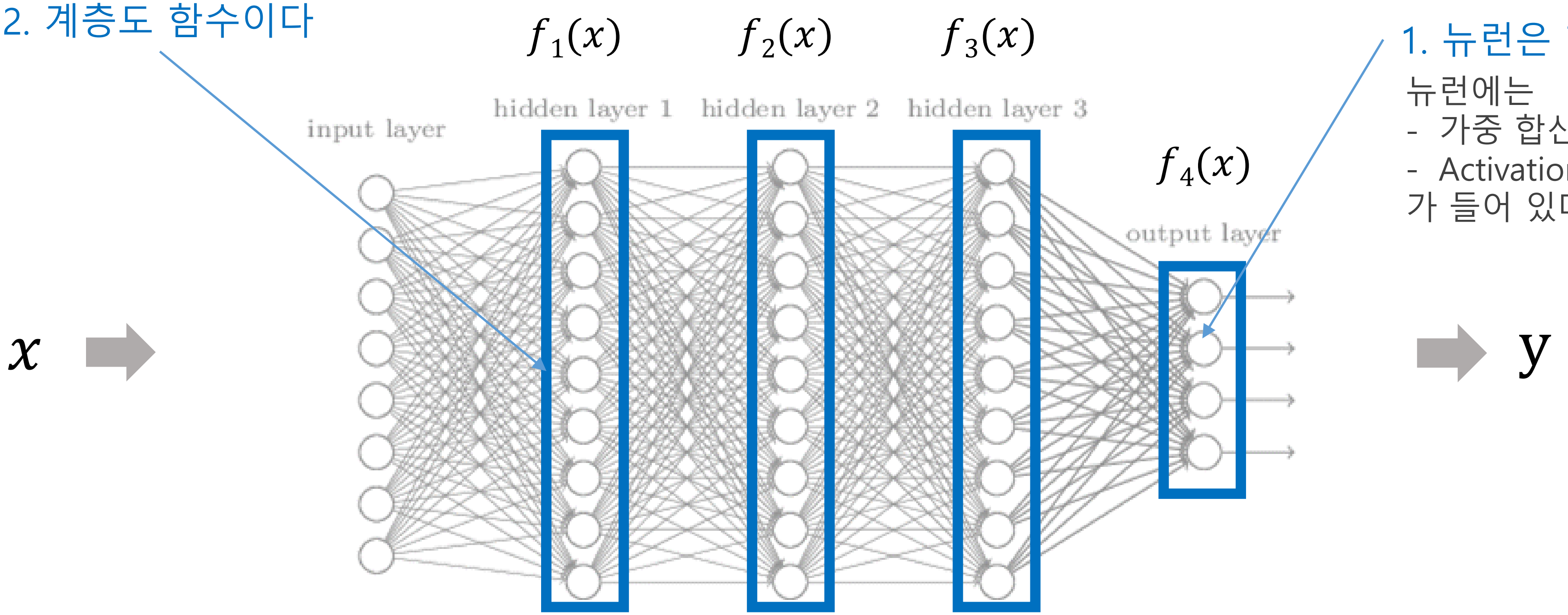
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

인공 뉴런



인공 신경망

2. 계층도 함수이다



1. 뉴런은 함수이다

뉴런에는
- 가중 합산
- Activation 함수

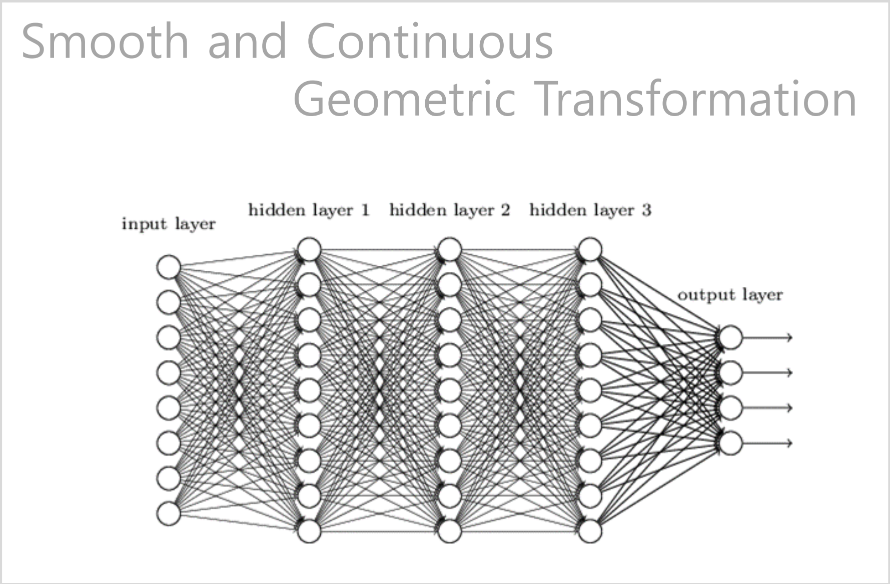
가 들어 있다.

$$y = f_4(f_3(f_2(f_1(x))))$$

3. 여러 함수들이 네트워크를 형성하고 있는 **합성 함수**

인공 신경망

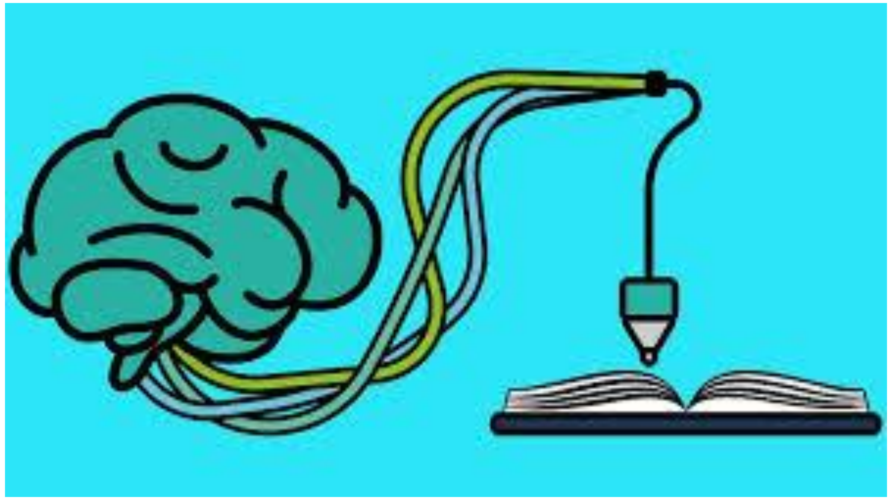
$$y = f(x;\theta)$$

x →  → y

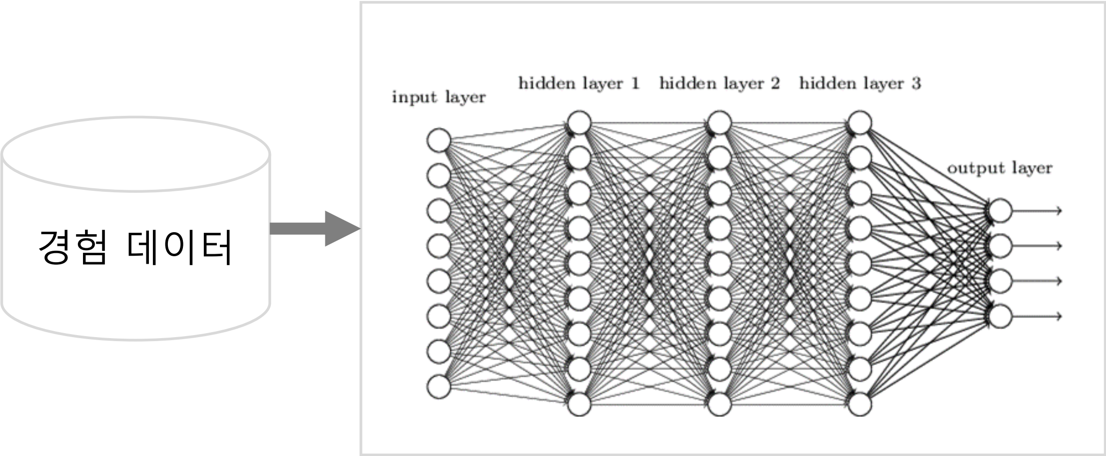
깊은 신경망은 아주 복잡한 맵핑 관계를 표현하는 **맵핑 함수**이다!

인공 신경망

사람의 뇌가 경험을 통해 학습하듯...



인공 신경망도 경험 데이터를 통해 학습



$y = f(x; \theta)$ 아주 복잡한 맵핑 함수를 학습을 통해 스스로 만들어 낸다!

Universal Approximation Theorem

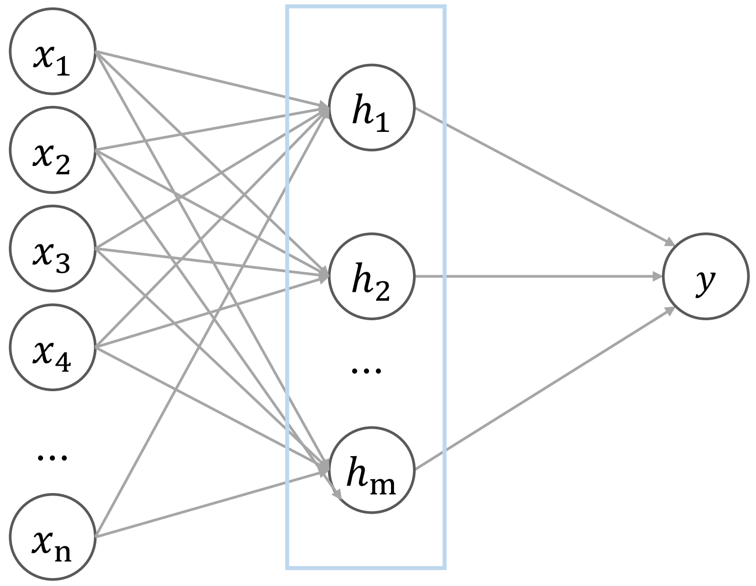
Universal Approximation Theorem

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

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

Continuous function

2-Layer Feed-forward Network



Single hidden layer
finite number of neurons

"더 깊은 신경망이 필요한가?"

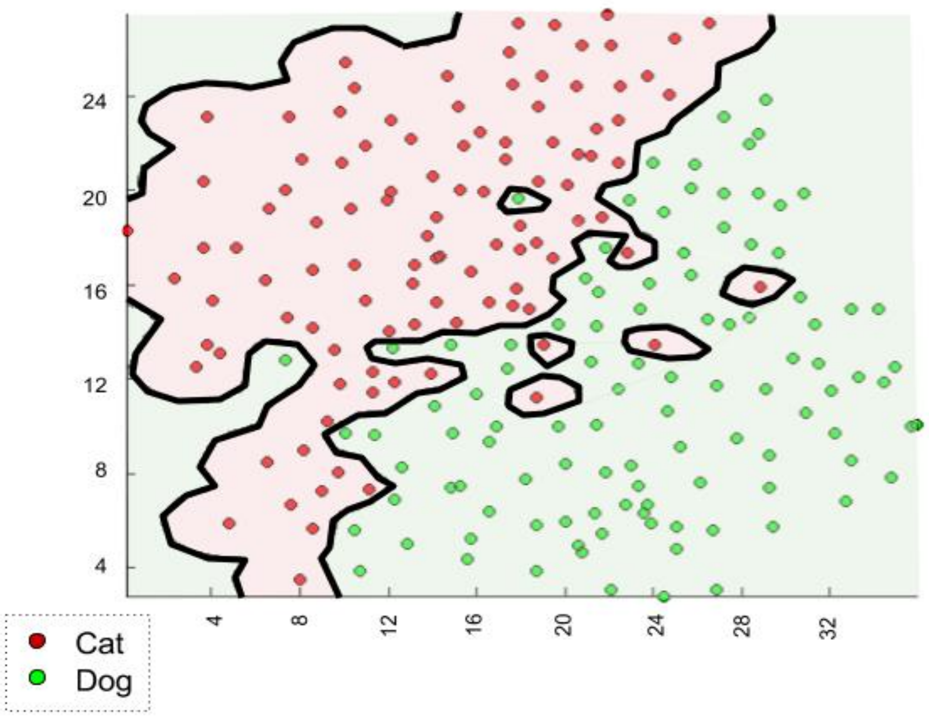
- 신경망을 깊게 하면 적은 수의 뉴런으로 함수를 구현할 수 있음
- 신경망이 깊어질수록 함수를 정확하게 근사할 수 있음 (임계점이 적어지고 Local Minima가 모여서 최적점을 잘 찾음)

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

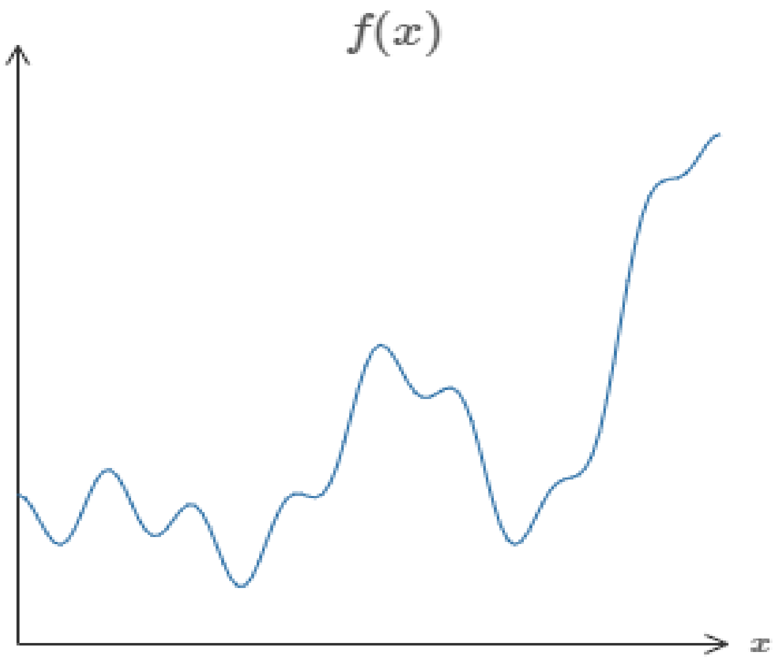
Universal Approximation Theorem

임의의 연속 함수를 근사한다는 의미는?

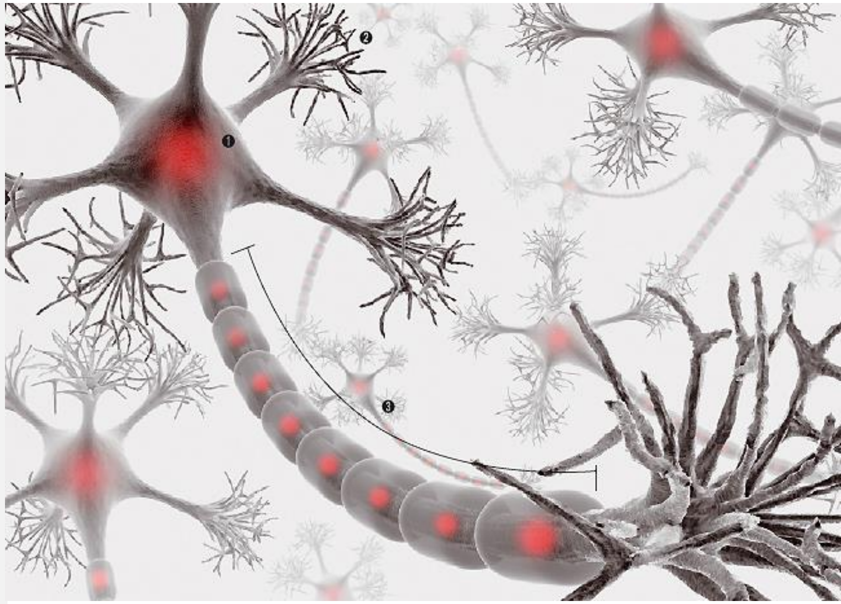
Classification



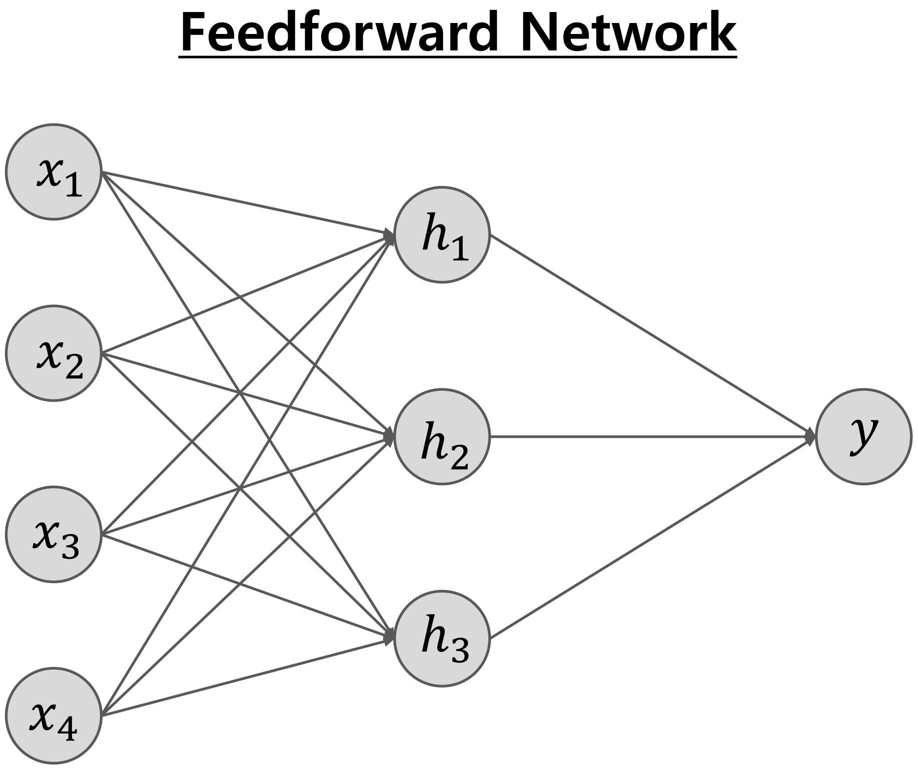
Regression



2 피드포워드 네트워크 구조



피드포워드 네트워크

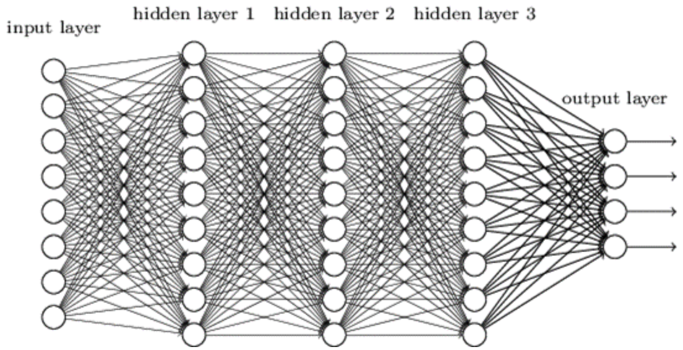


- 모든 연결이 입력에서 출력 방향으로만 되어 있음
- 다층 퍼셉트론(Multi-Layered Perceptron)과 동일
- 입력 데이터를 1차원 벡터 형태로 받아서 처리

네트워크 설계

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

Smooth and Continuous
Geometric Transformation



1 **Input**

- 입력 형태

x → … → 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

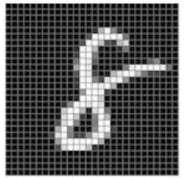
Input 입력 형태

MNIST 예제

Input :

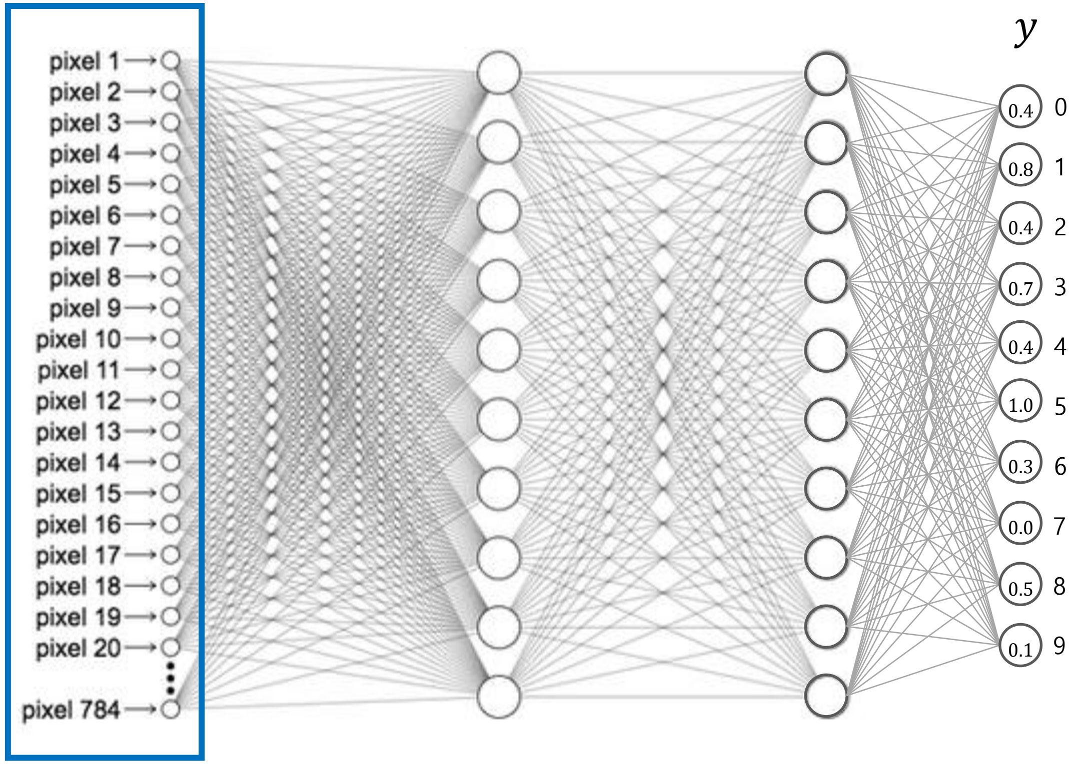
$$x = (x_1, x_2, x_3, \dots , x_n)$$

- 숫자로 이뤄진 n차원 벡터



28x28 이미지

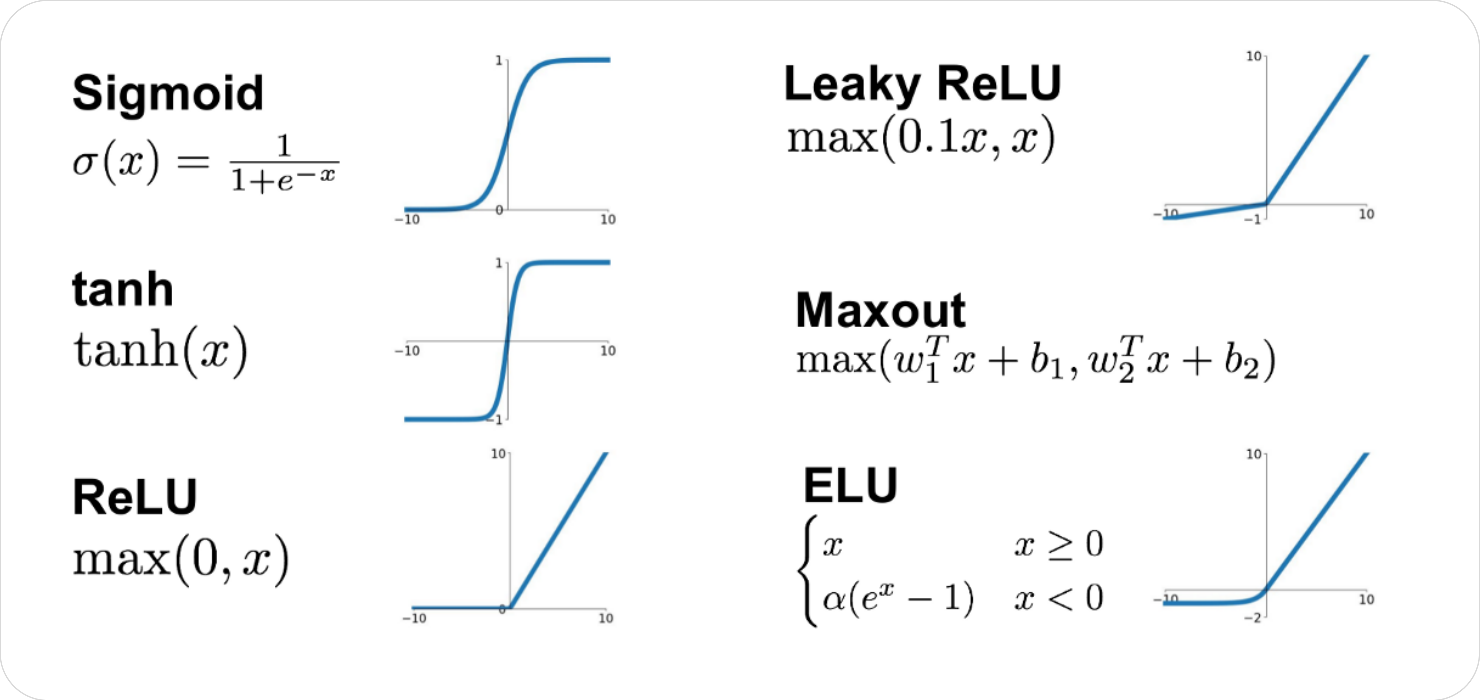
784 차원 벡터



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

Hidden Activation Function 종류

Activation Function



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

Output 출력 형태

분류 문제 (Classification)

Classification Network

dog (1)

cat (2)

rabbit (3)

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)

방의 수 (개)

면적 (평)

집 종류 (1. 아파트, 2. 주택)

역과의 거리 (km)

Regression Network

집 값

Continuous Value

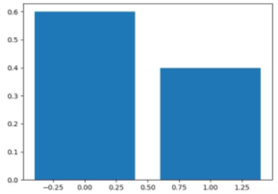
- **여러 독립 변수와 종속 변수의 관계를 함수 형태로 분석하는 문제**
- 출력은 입력 데이터에 대한 함수 값
- **확률 모델** : 관측 값에 대한 확률 분포를 예측

  ex) Gaussian Distribution

Output Activation Function

분류 문제 (Classification)

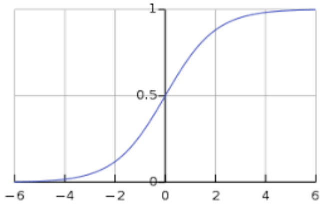
Bernoulli Distribution



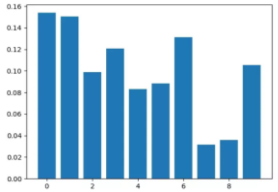
- 2개 카테고리로 분류된 이산 데이터

Sigmoid

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



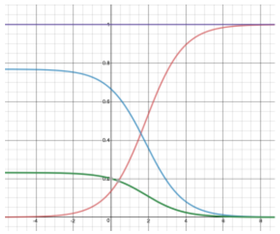
Categorical Distribution



- n개 카테고리로 분류된 이산 데이터

Softmax

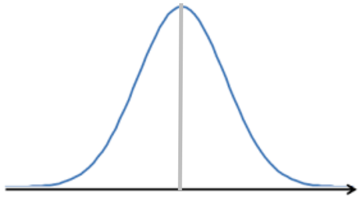
$$f(yi) = \frac{e^{y_i}}{\sum_{j=0}^{N} e^{y_j}}$$



Discrete Case

회귀 문제 (Regression)

Gaussian Distribution

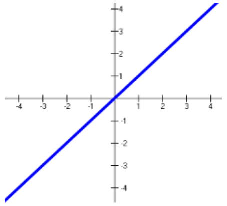


μ

- 연속 데이터는 대부분 가우시안으로 가정

Identity

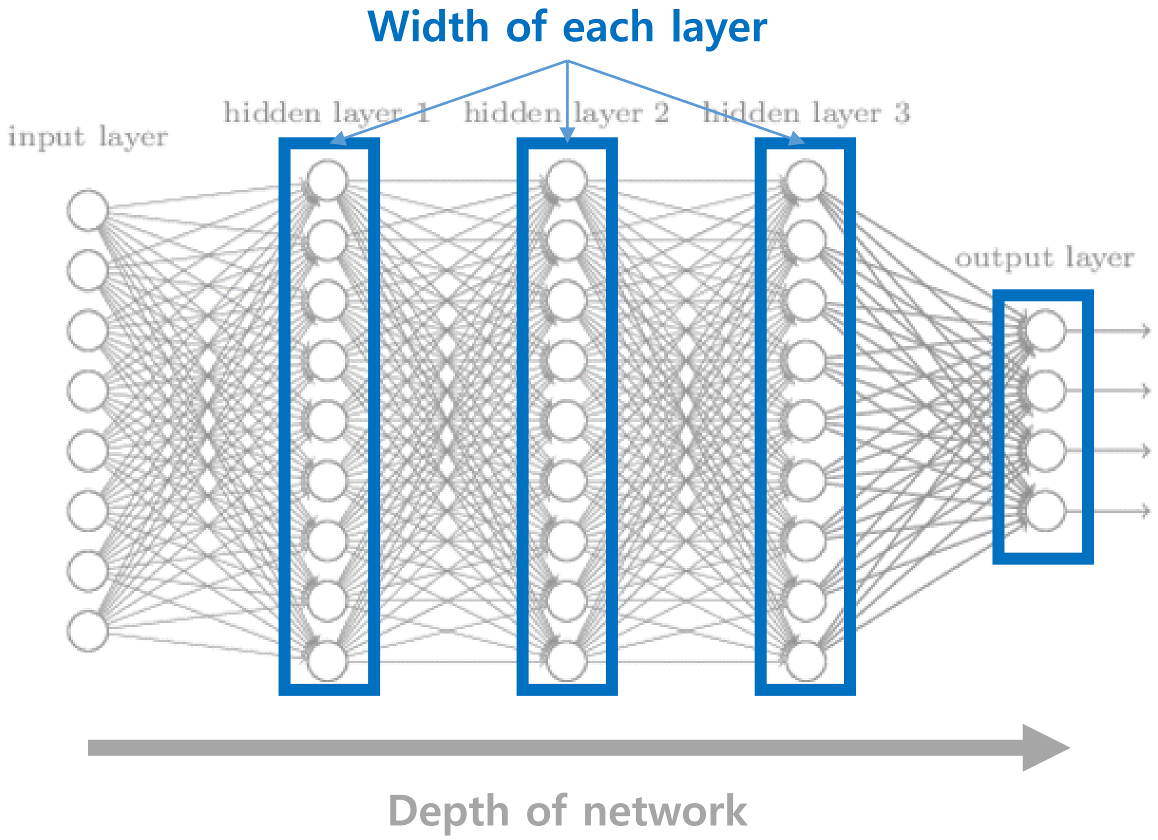
$$f(n) = n$$



Continuous Case

Network Size 네트워크 크기 설계

Network의 Depth와 Width를 어떻게 정할 것인가?



네트워크 깊이가 깊을 수록

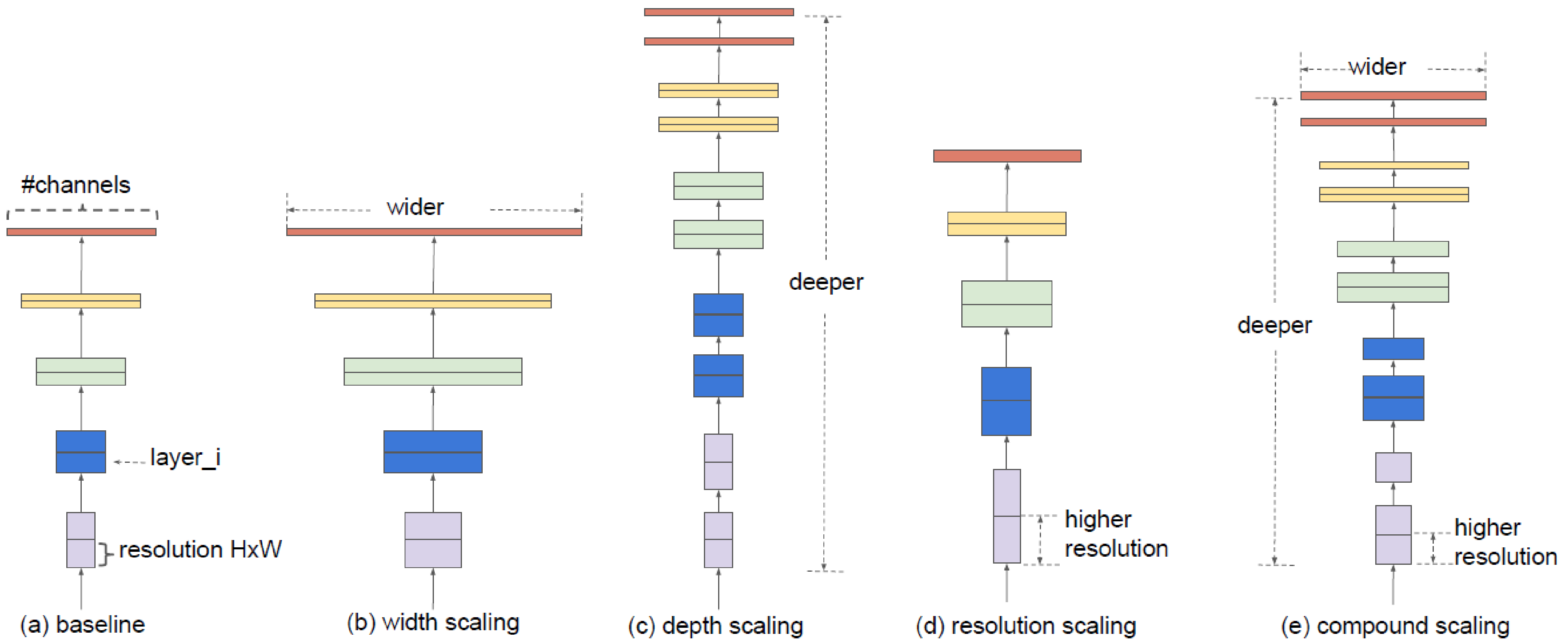
- 계층 별 뉴런을 적게 사용
- 적은 파라미터를 사용
- 일반화를 잘 함

하지만, 최적화가 어렵다.

문제에 따라 최적의 네트워크 구조는
Trial & Error로 찾아내야 한다!

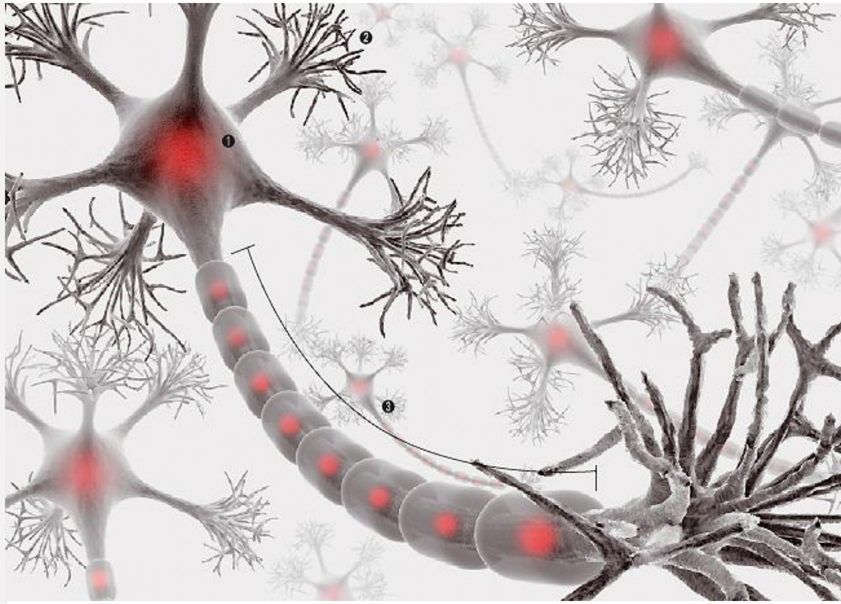
Network Size 네트워크 깊이와 모델 크기

Model Scaling (Width, Depth, Resolution, Compounding)



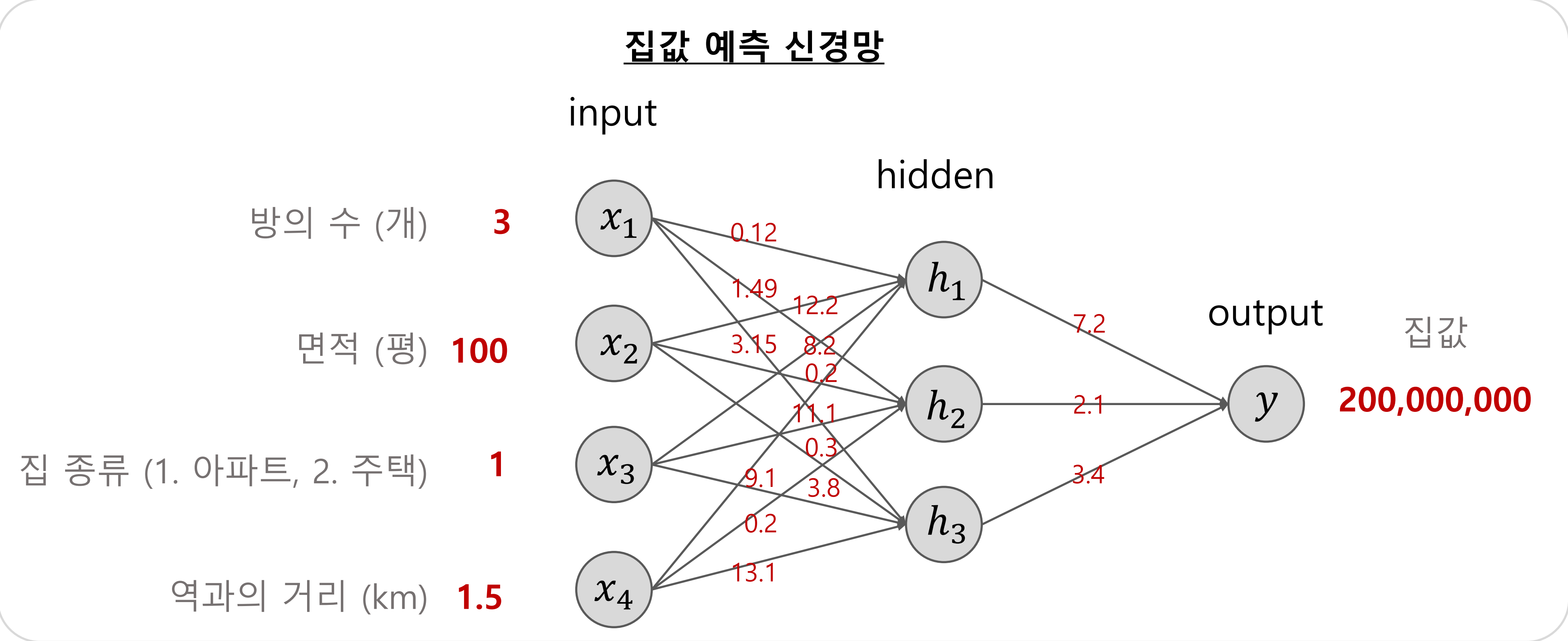
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Mingxing Tan Quoc V. Le (2019)

3 신경망 학습



인공 신경망의 학습

주어진 입력과 타깃 데이터를 이용해서 신경망 스스로 함수를 찾아내는 것



신경망 스스로 파라미터를 찾아 함수를 정의하는 것을 학습이라고 한다!

최적화 문제

<u>회귀 (Regression)</u>

타깃과 인공신경망이 예측한 값의 차이를 최소화하는 파라미터를 찾아라.

파라미터

$$\min_{\theta} \quad \frac{1}{n} \sum \| t - f(x;\theta) \|_2^2$$

타깃 (관측 레이블)

모델의 예측

평균 제곱 오차 (Mean Squared Error)

최적화 문제

분류 (Classification)

관측 분포와 인공신경망이 예측한 분포의 차이를 최소화하는 파라미터를 찾아라.

파라미터 → $$\min_{\theta} \quad -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} t_k \cdot \log f(x;\theta)_k$$

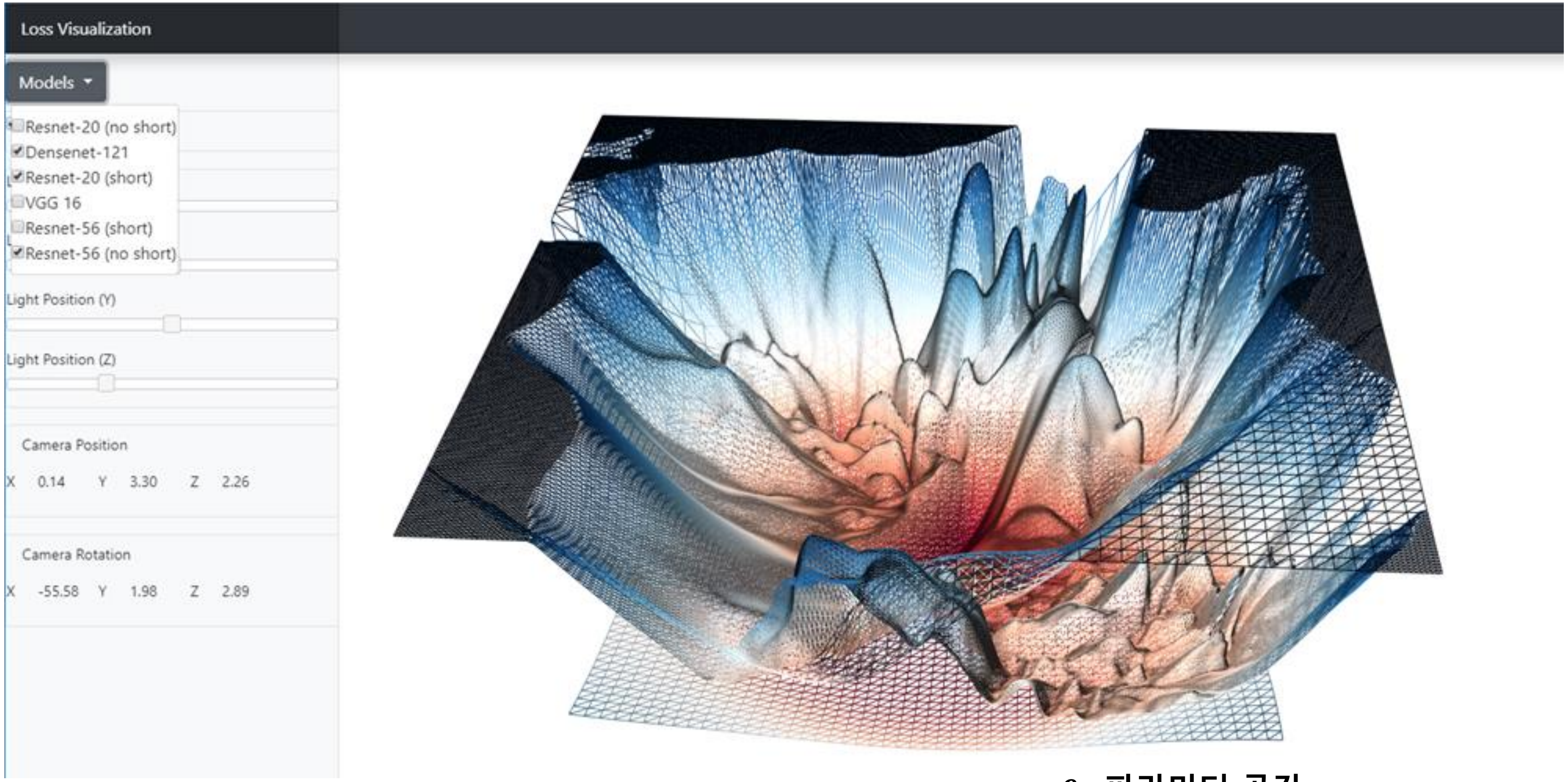
K : Class 개수

관측 분포 (t_k)

모델이 예측한 분포 ($f(x;\theta)_k$)

크로스 엔트로피 (Cross Entropy)

Loss Surface

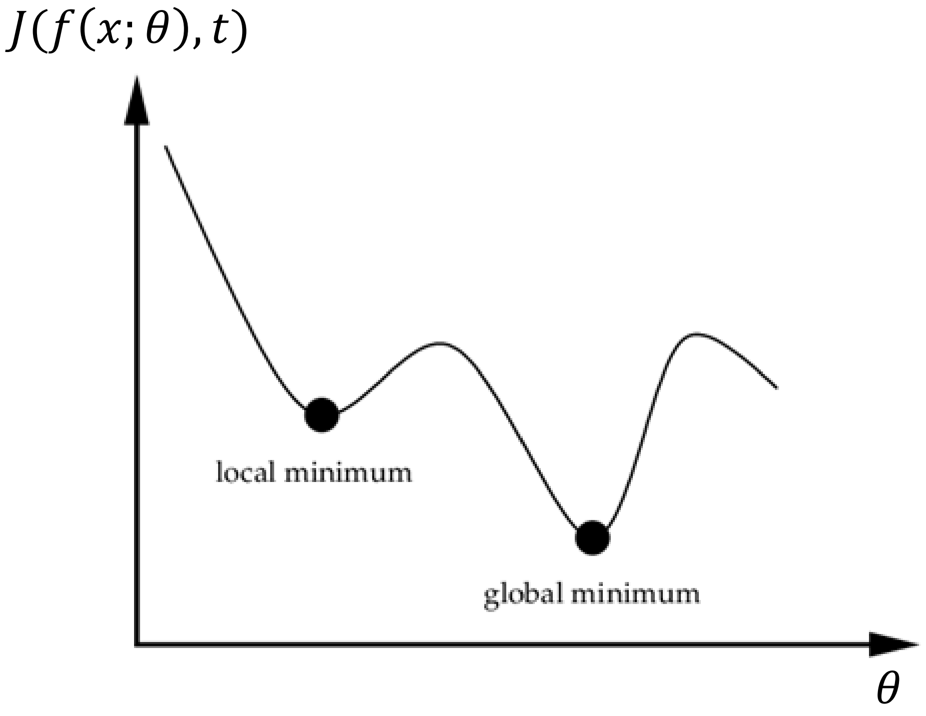


θ : 파라미터 공간

http://www.telesens.co/2019/01/16/neural-network-loss-visualization/

Loss를 최소화 하려면?

Loss Minimization



최적화 알고리즘

1차 미분

- Gradient Descent
- Variants of Gradient Descent :
  : SGD, Adagrad, Momentum, RMS prob, Adam

Deep Learning에서 주로 사용하는 방법

1.5차 미분

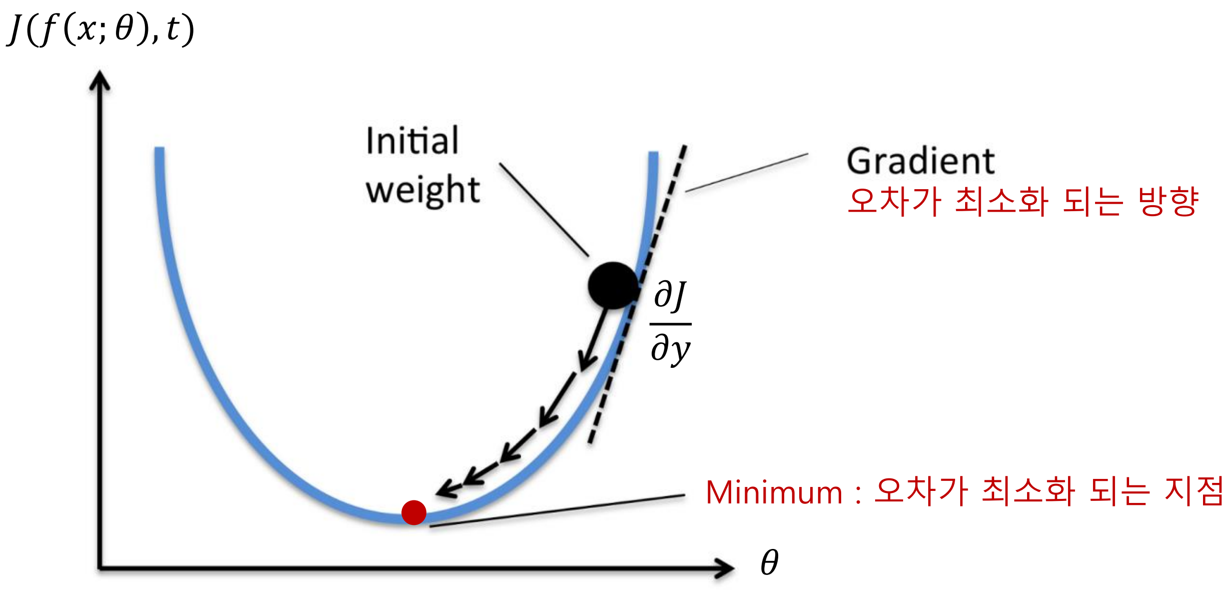
- Quasi-Newton Method
- Conjugate Gradient Descent
- Levenberg-Marquardt Method

2차 미분

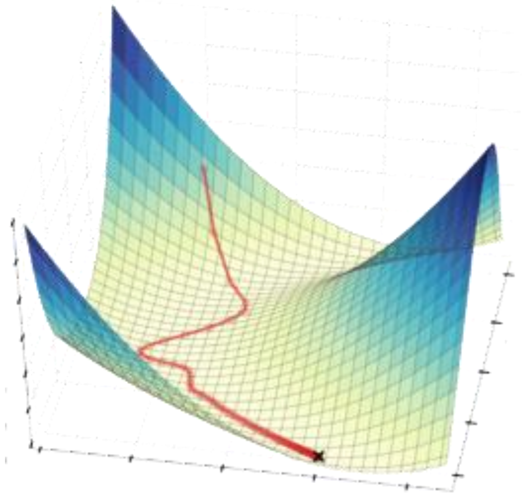
- Newton Method
- Interior Point Method

Gradient Descent

Gradient Descent



3D View



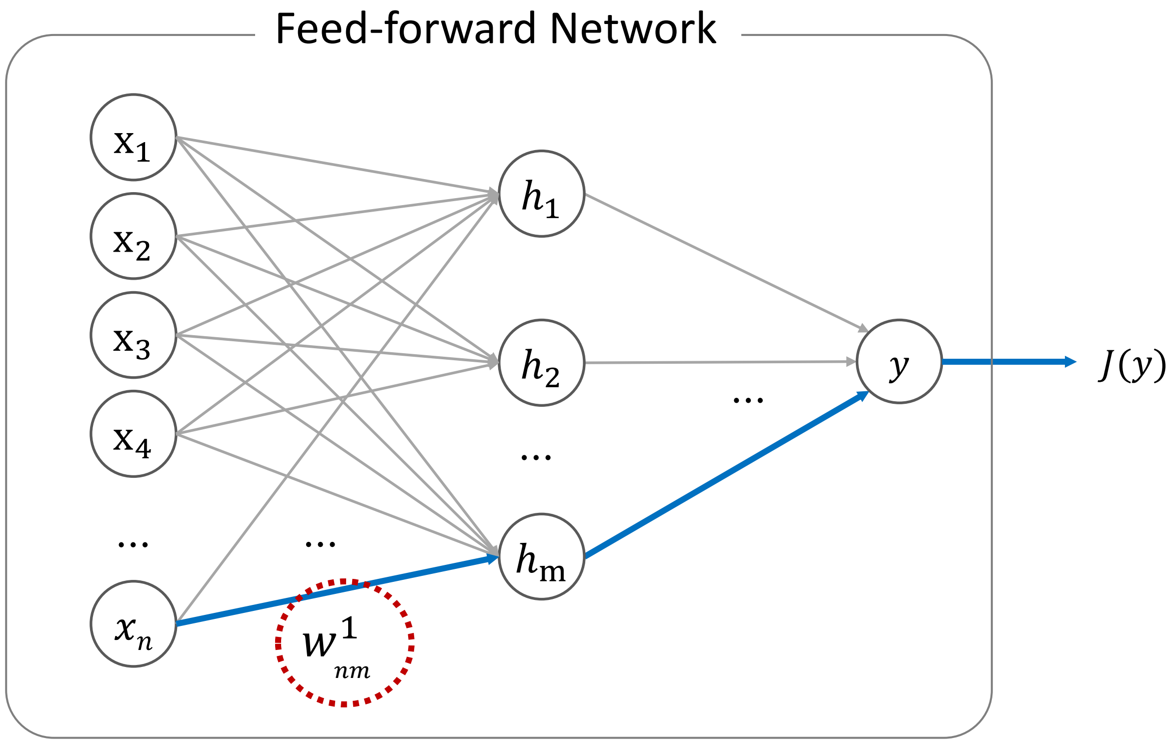
Parameter Update

$$\theta^{+} = \theta - \alpha \frac{\partial J}{\partial \theta}$$

Step Size

Gradient

Gradient Descent



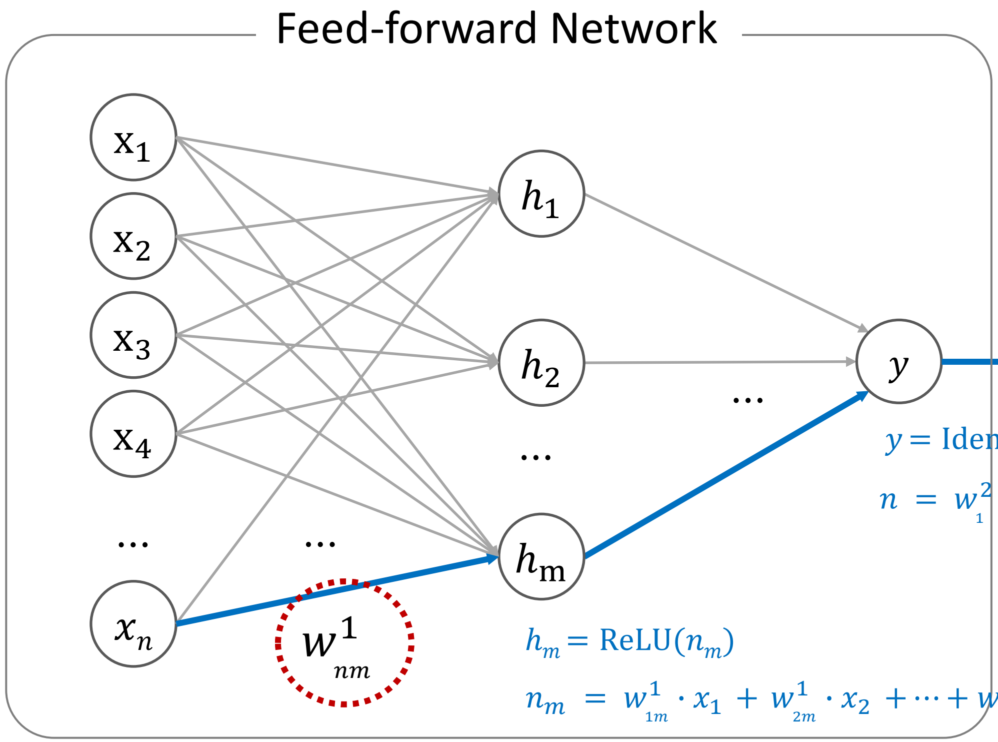
Parameter Update

$$w^{1\,+}_{nm} = w^1_{nm} - \alpha \frac{\partial J}{\partial w^1_{nm}}$$

Step Size

Gradient

Gradient Descent



$y = \mathrm{Identity}(n)$

$n = w^2_1 \cdot h_1 + w^2_2 \cdot h_2 + \cdots + w^2_m \cdot h_m$

$h_m = \mathrm{ReLU}(n_m)$

$n_m = w^1_{1m} \cdot x_1 + w^1_{2m} \cdot x_2 + \cdots + w^1_{nm} \cdot x_n$

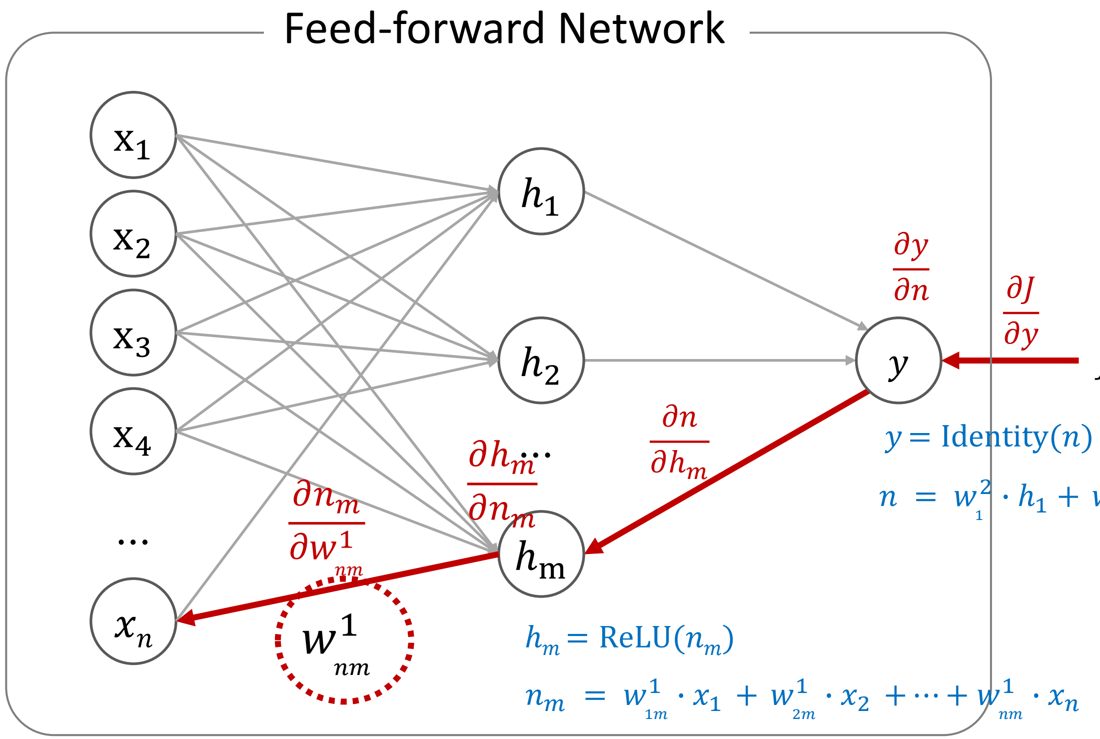
$J(y) = \frac{1}{N} \sum_{i=1}^{N} (y - t)^2$

Gradient of Parameter

$$w^{1\,+}_{nm} = w^1_{nm} - \underbrace{\alpha}_{\text{Step Size}} \underbrace{\frac{\partial J}{\partial w^1_{nm}}}_{\text{Gradient}}$$

“가중치는 Loss Function의 간접 파라미터이므로
직접 미분이 안됨”

Backpropagation



$J(y) = \frac{1}{N}\sum_{i=1}^{N} (y - t)^2$

Gradient of Parameter

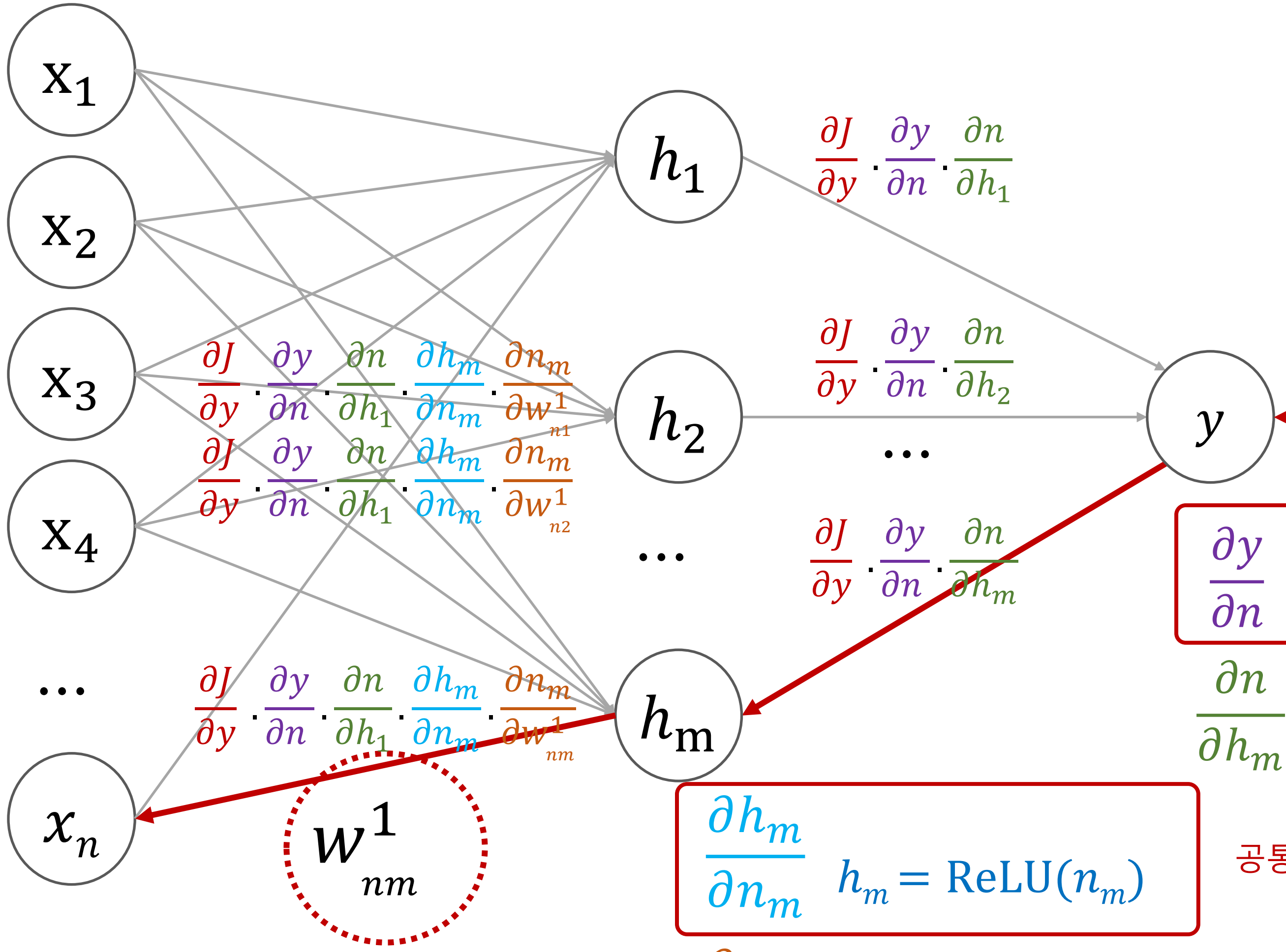
$$\frac{\partial J}{\partial w^1_{nm}} = \boxed{\frac{\partial J}{\partial y}\cdot\frac{\partial y}{\partial n}\cdot\frac{\partial n}{\partial h_m}\cdot\frac{\partial h_m}{\partial n_m}\cdot\frac{\partial n_m}{\partial w^1_{nm}}}$$

연쇄 법칙 (Chain Rule) 사용

$$= \frac{1}{N}\sum_{i=1}^{N} 2(y - t).\mathrm{Identity}'(n).w^2{}_m.\mathrm{ReLU}'(n_m).x_n$$

$$\begin{cases} \frac{\partial J}{\partial y} = \frac{1}{N}\sum_{i=1}^{N} 2(y - t) \\ \frac{\partial y}{\partial n} = \mathrm{Identity}'(n) \\ \frac{\partial n}{\partial h_m} = w^2{}_m \\ \frac{\partial h_m}{\partial n_m} = \mathrm{ReLU}'(n_m) \\ \frac{\partial nm}{\partial w^1_{nm}} = x_n \end{cases}$$

Backpropagation



$$\frac{\partial J}{\partial y}$$

$$J(y) = \frac{1}{N} \sum_{i=1}^{N} (y - t)^2$$

$$\boxed{\frac{\partial y}{\partial n} \quad y = \mathrm{Identity}(n)}$$

공통

$$\frac{\partial n}{\partial h_m} \quad n = w^2_1 \cdot h_1 + w^2_2 \cdot h_2 + \cdots + w^2_m \cdot h_m$$

가중치 별로 별도 계산

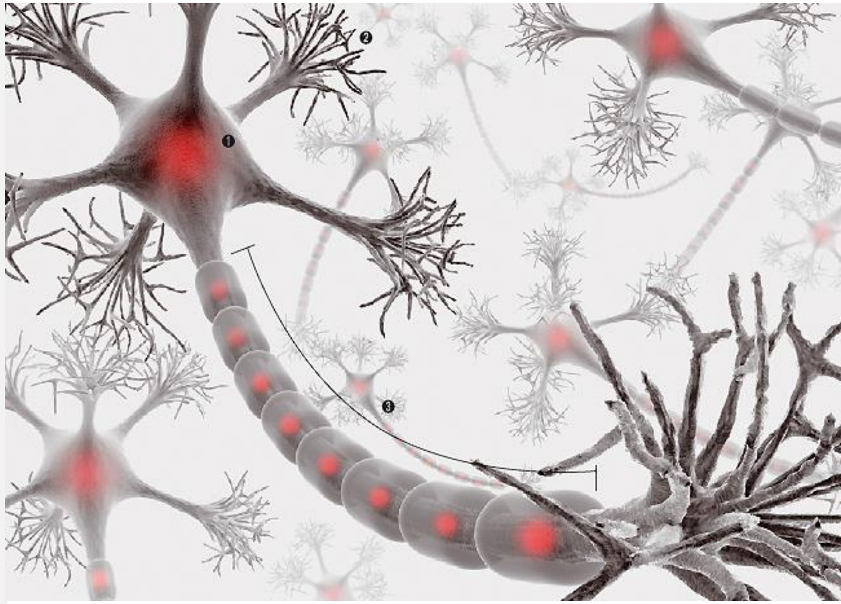
$$\boxed{\frac{\partial h_m}{\partial n_m} \quad h_m = \mathrm{ReLU}(n_m)}$$

공통

$$\frac{\partial n_m}{\partial w^1_{nm}} \quad n_m = w^1_{1m} \cdot x_1 + w^1_{2m} \cdot x_2 + \cdots + w^1_{nm} \cdot x_n$$

가중치 별로 별도 계산

4 TensorFlow



TensorFlow 2.0

Static Graph 방식

Define and Run

- 계산 그래프를 정의하고 난 후 명시적으로 그래프를 실행하는 방식

Dynamic Graph 방식

Define by Run

- 코드를 실행하면서 동시에 계산 그래프를 생성하는 방식
- PyTorch, Chainer 등에서 지원

Eager Execution

Define and Run에서 Define by Run 으로!

```python
import tensorflow as tf

a = tf.constant(5)
b = tf.constant(3)

c = a + b

with tf.session() as sess:
    print(sess.run(c))
```

symbolic

```python
import tensorflow as tf

a = tf.constant(5)
b = tf.constant(3)

c = a + b

print(c)
```

concrete

TensorFlow 1.x :	8	Tensor("add_2:0", shape=(), dtype=int32)
TensorFlow 2.x :	Error	tf.Tensor(8, shape=(), dtype=int32)

Eager Execution

z = w * x + b 구현

TensorFlow 1.x

```
import tensorflow as tf

## 그래프 정의
g = tf.Graph()
with g.as_default():
    x = tf.placeholder(dtype=tf.float32,
                       shape=(None), name='x')
    w = tf.Variable(2.0, name='weight')
    b = tf.Variable(0.7, name='bias')
    z = w * x + b
    init = tf.global_variables_initializer()

## 세션 생성 및 그래프 g 전달
with tf.Session(graph=g) as sess:
    ## w와 b 초기화
    sess.run(init)
    ## z 평가
    for t in [1.0, 0.6, -1.8]:
        print('x=%4.1f --> z=%4.1f'%(
              t, sess.run(z, feed_dict={x:t})))
```

TensorFlow 2.x

```
import tensorflow as tf

w = tf.Variable(2.0, name='weight')
b = tf.Variable(0.7, name='bias')

# ## z 평가
for x in [1.0, 0.6, -1.8]:
    z = w * x + b
    print('x=%4.1f --> z=%4.1f'%(x, z))
```

모델 정의 tf.Module

tf.Module : 신경망 모듈 클래스의 베이스 클래스

```
tf.Module(
    name=None
)
```

- **trainable_variables** : 훈련 변수들의 목록
- Variables : 모든 변수들의 목록
- Submodules : 멤버 Module의 목록

모듈에 포함된 tf.Variable, tf.Module, input에 적용되는 function들을 관리함

https://www.tensorflow.org/api_docs/python/tf/Module

모델 정의 tf.Module

tf.Module을 이용해서 계층 정의하기

```
class Dense(tf.Module):

  def __init__(self, in_features, out_features, name=None):

    super(Dense, self).__init__(name=name)

    # 가중치와 편향 정의
    self.w = tf.Variable(tf.random.normal([in_features, out_features]), name='w')
    self.b = tf.Variable(tf.zeros([out_features]), name='b')

  def __call__(self, x):

    y = tf.matmul(x, self.w) + self.b    # 가중 합산
    return tf.nn.relu(y)  # 활성 함수 실행
```

모델 정의 tf.Module

tf.Module을 이용해서 모델 정의하기

```
class MLP(tf.Module):

  def __init__(self, input_size, sizes, name=None): # sizes에는 각 계층의 뉴런 개수가 정의되어 있음
    super(MLP, self).__init__(name=name)

    self.layers = [] # 계층의 리스트
    with self.name_scope:
      for size in sizes:
        self.layers.append(Dense(input_size=input_size, output_size=size)) # 각 계층 정의
        input_size = size

  @tf.Module.with_name_scope
  def __call__(self, x):
    for layer in self.layers: # 각 계층을 순서대로 호출
      x = layer(x)
    return x
```

신경망 훈련 tf.GradientTape

```
@tf.function
def train_step(input, target):
 with tf.GradientTape() as tape:
    # forward Pass
    predictions = model(input)
    # compute the loss
    loss = tf.reduce_mean(
          tf.keras.losses.sparse_categorical_crossentropy(
          target, predictions, from_logits=True))
 # compute gradients
 grads = tape.gradient(loss, model.trainable_variables)
 # perform a gradient descent step
 optimizer.apply_gradients(zip(grads, model.trainable_variables))
 return loss
```

신경망 모델 실행 및 Loss 계산

Gradient 계산 (Backpropagation)

Parameter Update (최적화)

Thank you!