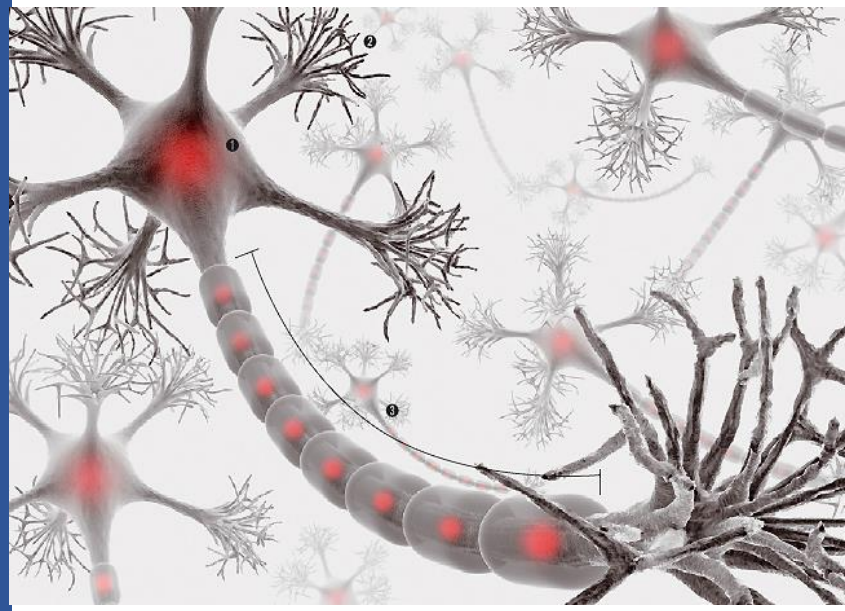
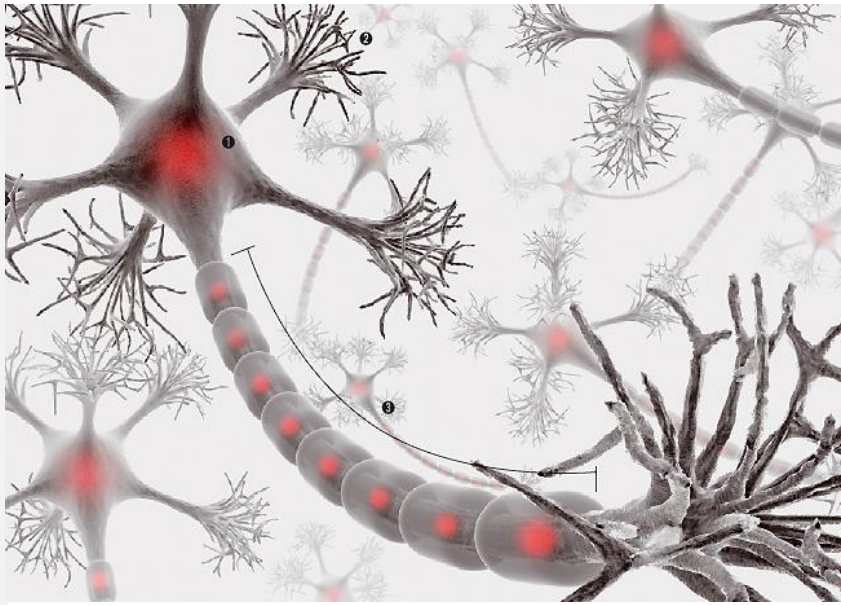
[복습] 순방향신경망과 학습

학습 목표

- Day1에 배웠던 내용들을 복습해본다.

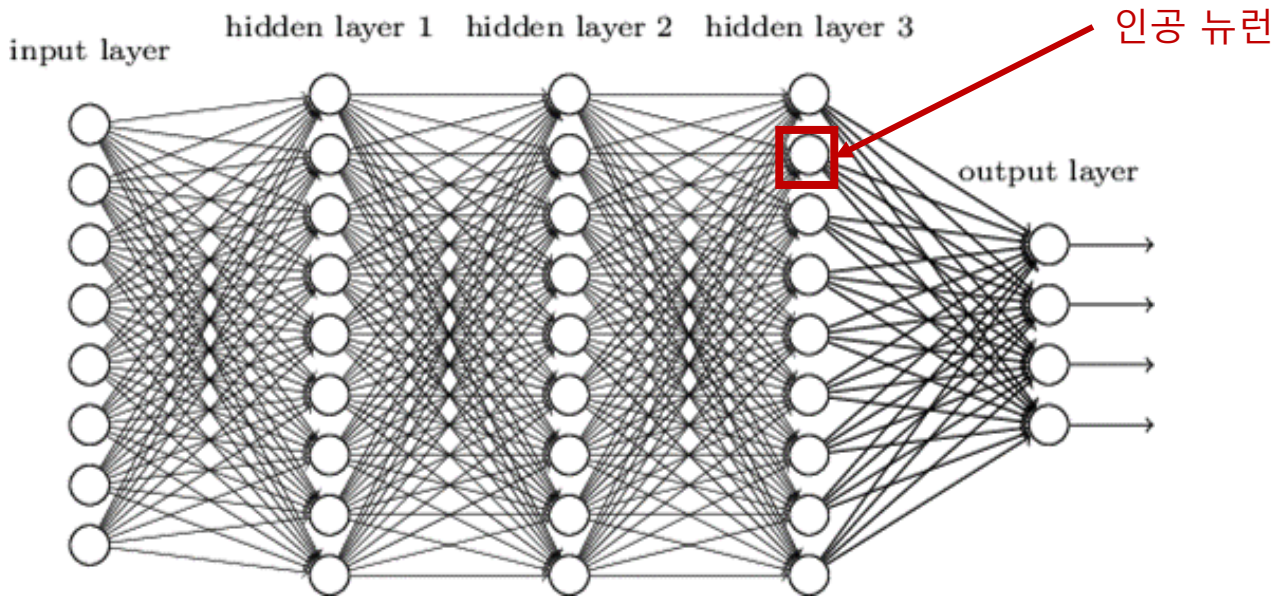


함수로서의 신경망



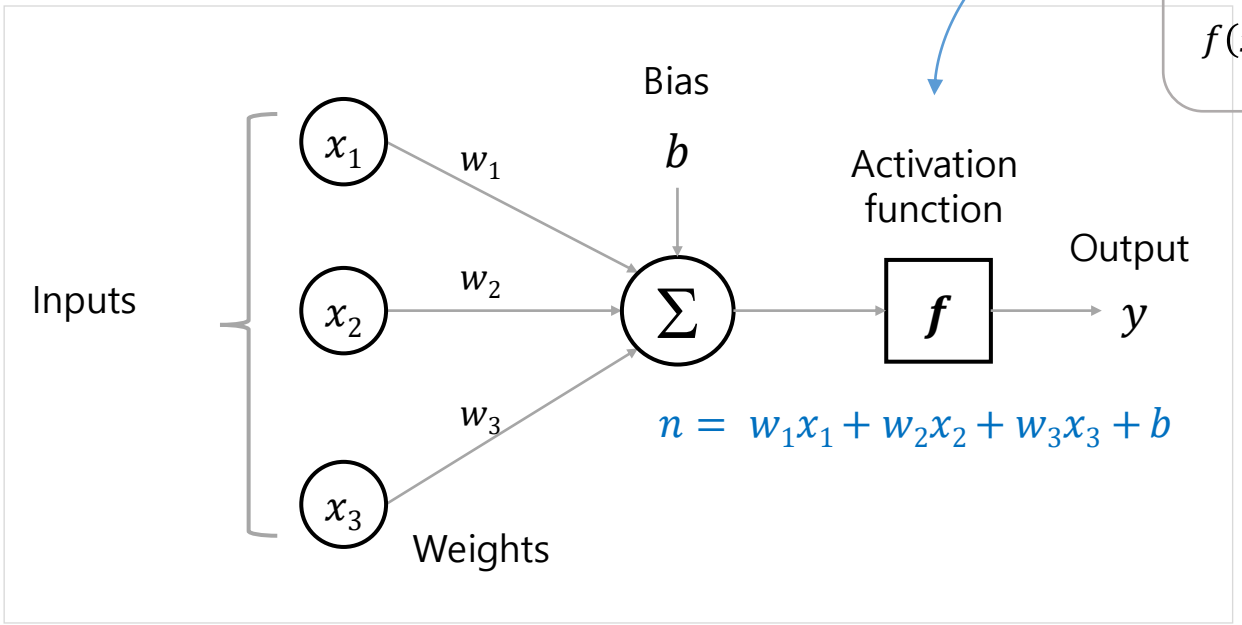
인공 신경망

인공 신경망 (Artificial Neural Network)

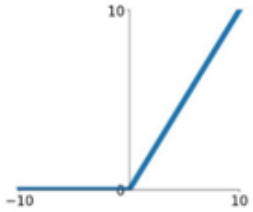


- 생체 신경망의 작동 원리를 모방해서 만든 인공 신경망
- 인공 뉴런들이 서로 복잡하게 연결되어 있는 네트워크
- 뉴런은 신호를 받아서 임계치 이상이 되면 신호를 발화

인공 신경망



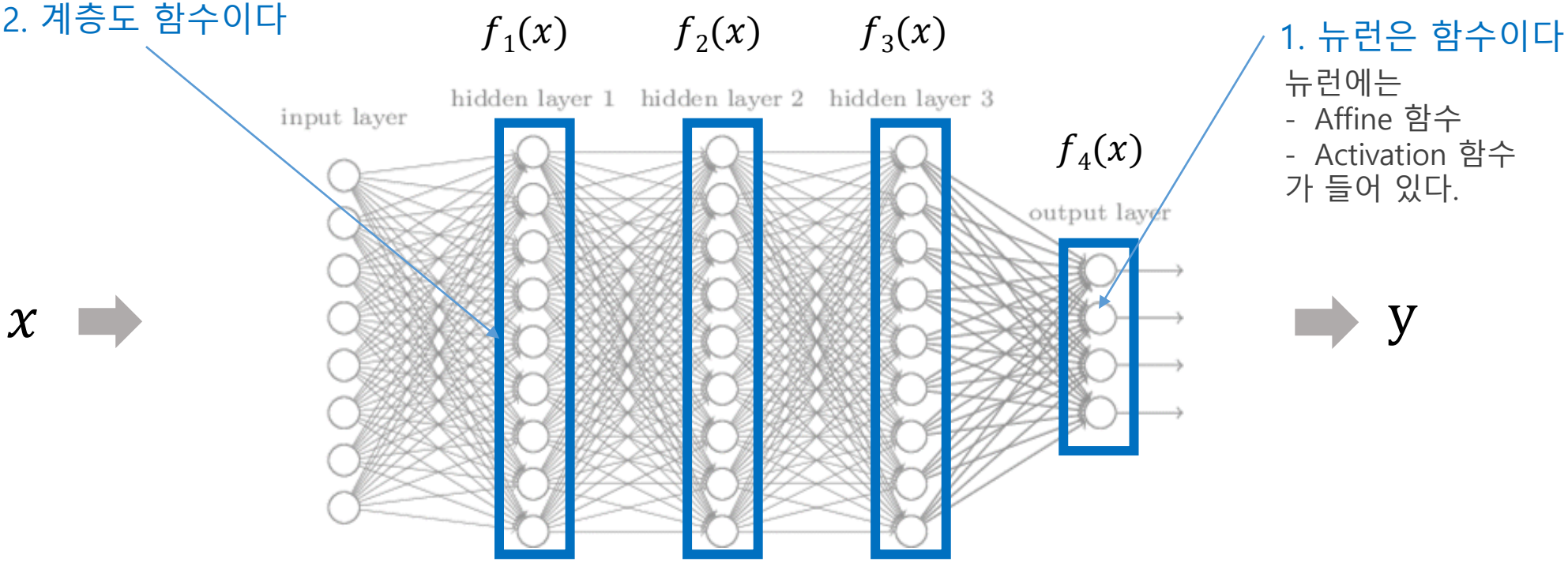
Activation function : ReLU



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

인공 신경망

2. 계층도 함수이다



1. 뉴런은 함수이다

뉴런에는
- Affine 함수
- Activation 함수

가 들어 있다.

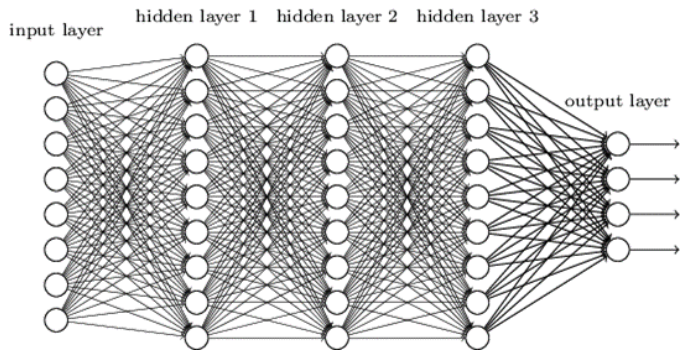
$$y = f_4(f_3(f_2(f_1(x))))$$

3. 여러 함수들이 네트워크를 형성하고 있는 **합성 함수**

인공 신경망

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

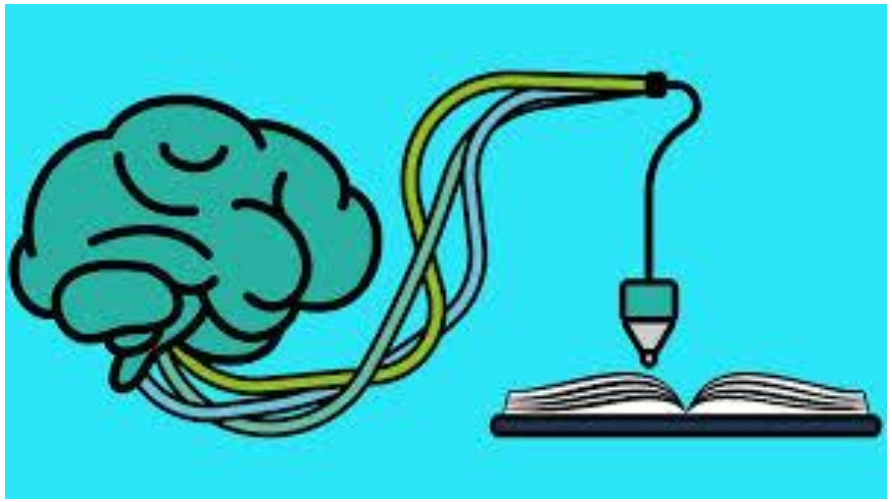
x ➡ Smooth and Continuous Geometric Transformation ➡ y



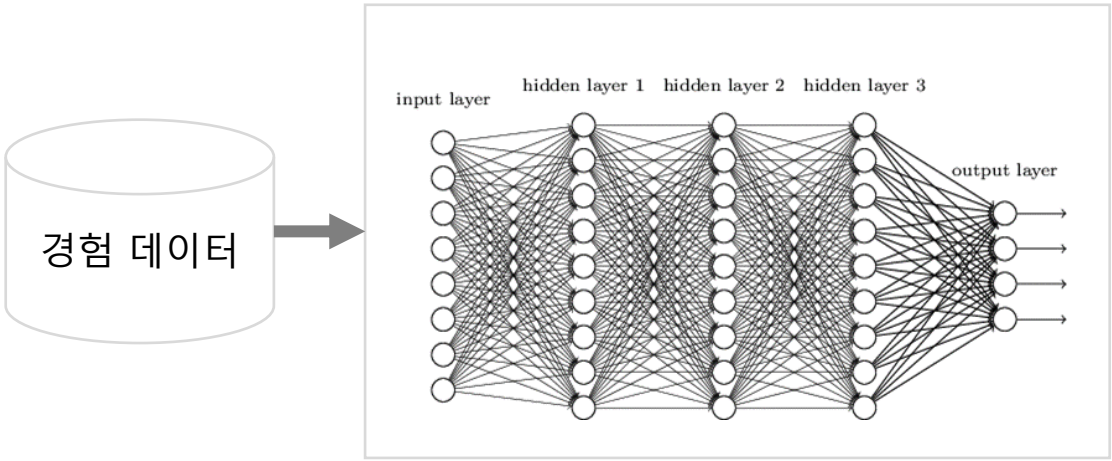
깊은 신경망은 아주 복잡한 맵핑 관계를 표현하는 **맵핑 함수**이다!

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!

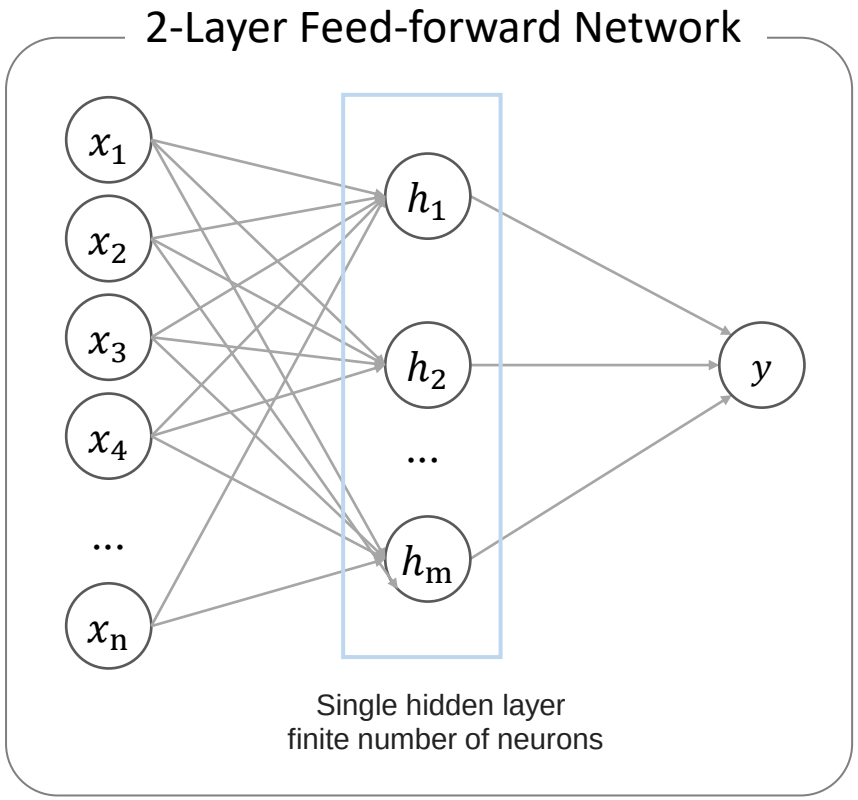
Universal Approximation Theorem

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 $\mathbf{R}^n$, under mild assumptions on the activation function.

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

Continuous function

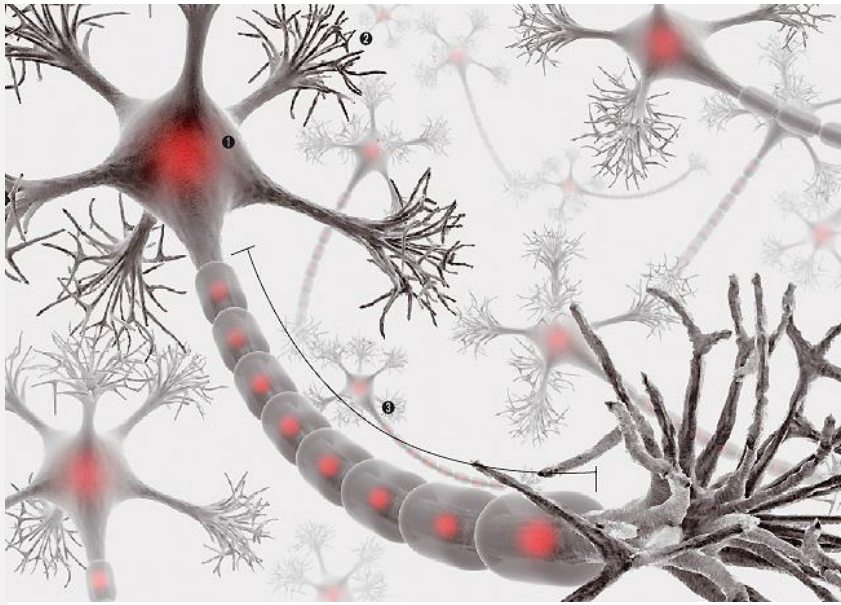


"더 깊은 신경망이 필요한가?"

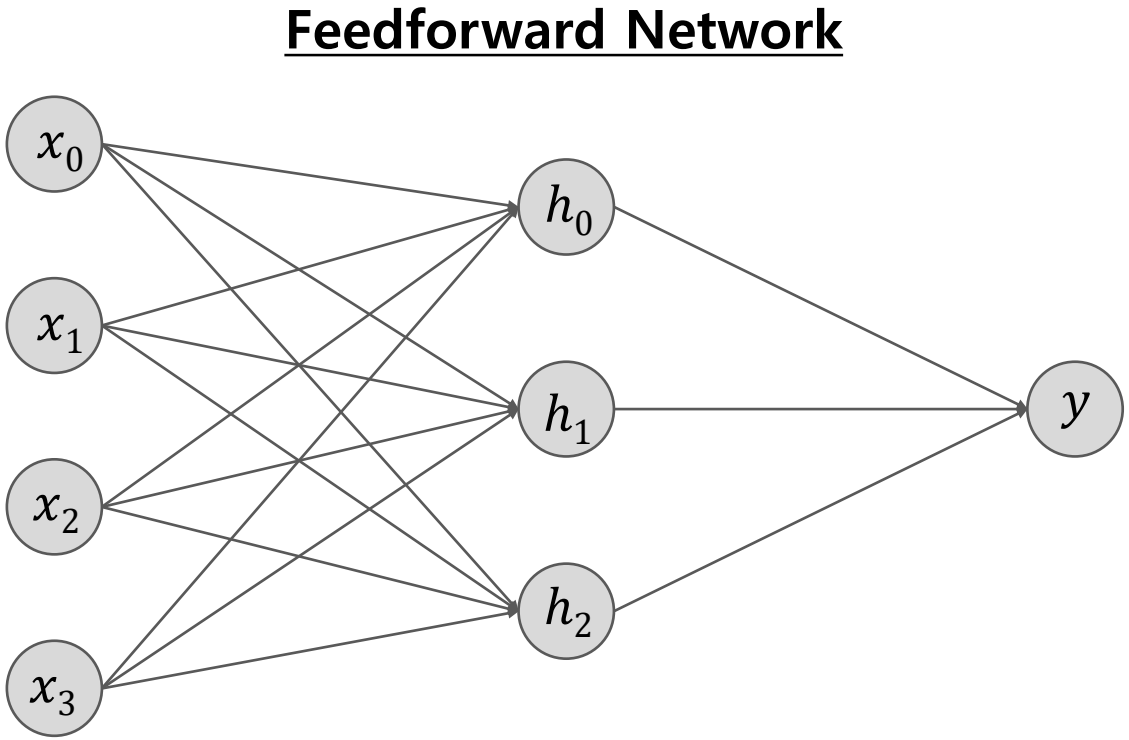
- 신경망을 깊게 하면 적은 수의 뉴런으로 함수를 구현할 수 있음
- 신경망이 깊어질수록 함수를 정확하게 근사할 수 있음 (임계점이 적어지고 Local Minima가 모여서 최적점을 잘 찾음)

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

순방향 신경망



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

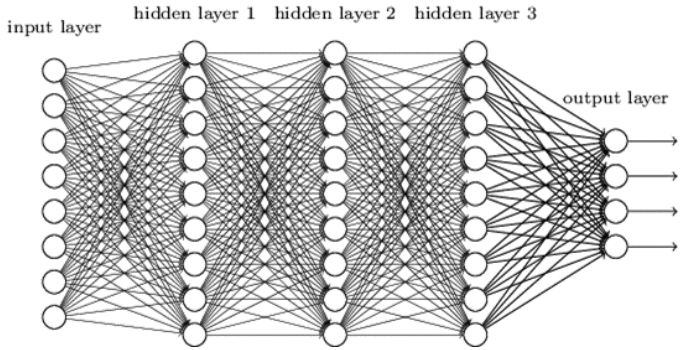
$$y = f(x;\theta)$$

1 Input

- 입력 형태

x

Smooth and Continuous
Geometric Transformation



y

2 Output

- 출력 형태
- Activation Function

3 Hidden

- Activation Function

4 Network Size

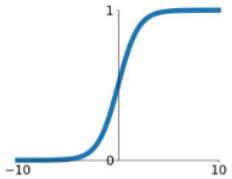
- 네트워크 깊이 (depth) : 레이어 수
- 네트워크 폭 (width) : 레이어 별 뉴런 수
- 연결 방식

Hidden Activation Function 종류

Activation Function

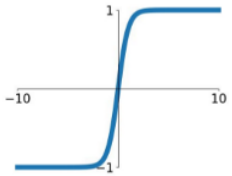
Sigmoid

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



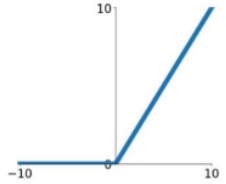
tanh

$$\tanh(x)$$



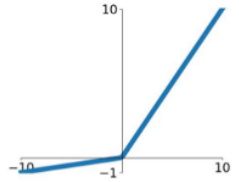
ReLU

$$\max(0, x)$$



Leaky ReLU

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

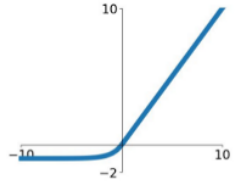


Maxout

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

ELU

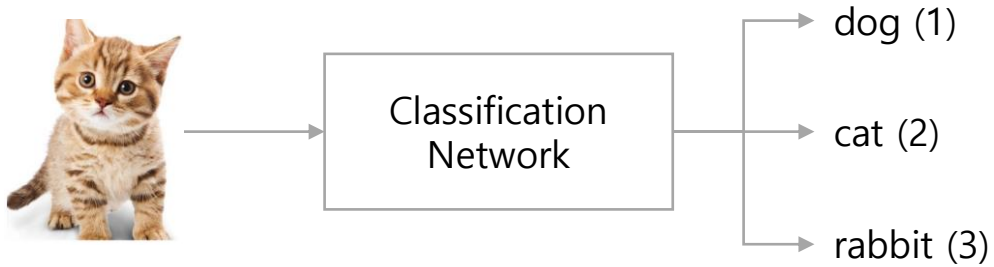
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



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

Output 출력 형태

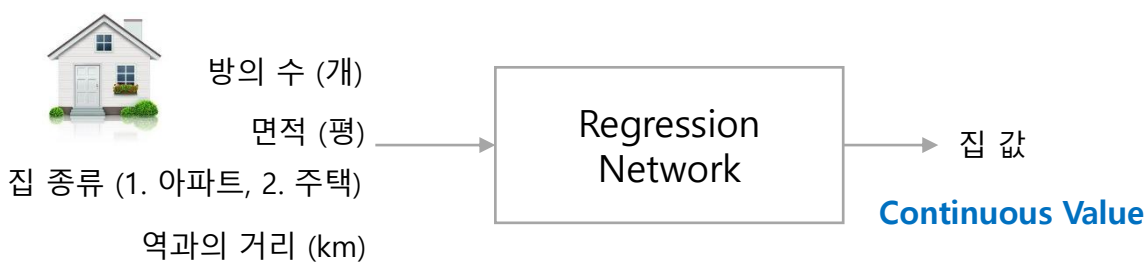
분류 문제 (Classification)



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)



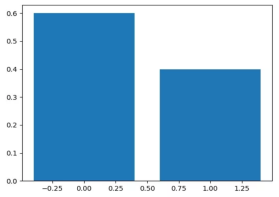
Continuous Value

- **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)

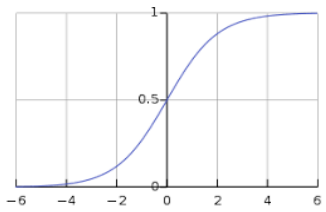
Bernoulli Distribution



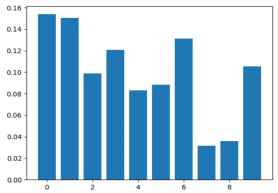
- 2개 카테고리로 분류된 이산 데이터

Sigmoid

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



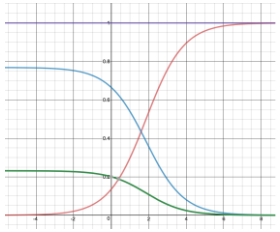
Categorical Distribution



- n개 카테고리로 분류된 이산 데이터

Softmax

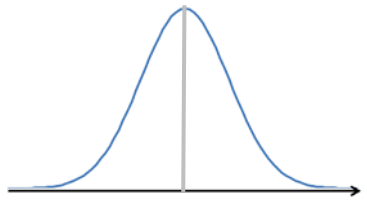
$$f(yi) = \frac{e^{y_i}}{\sum_{j=0}^{N} e^{y_j}}$$



Discrete Case

회귀 문제 (Regression)

Gaussian Distribution

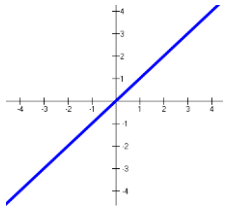


μ

- 연속 데이터는 대부분 가우시안으로 가정

Identity

$$f(n) = n$$



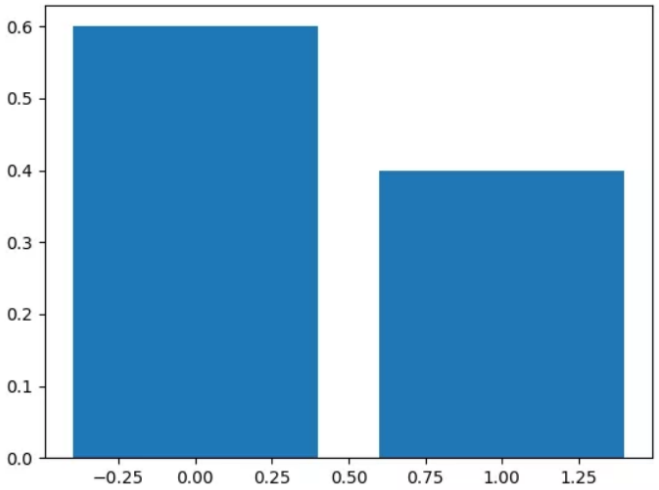
Continuous Case

Output 이진 분류 (Binary Classification)

이진 분류는
베르누이 분포를 추정하는 문제!

Bernoulli Distribution

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



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

$$x = 0, 1$$

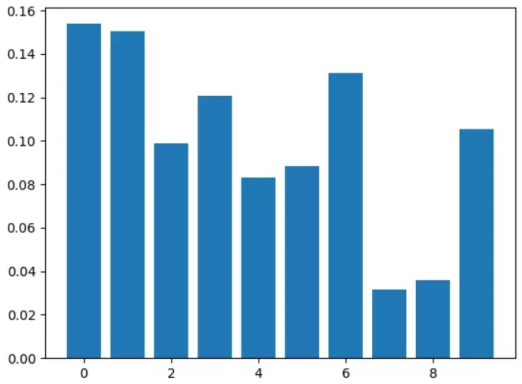
Output 다중 분류 (Multiclass Classification)

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

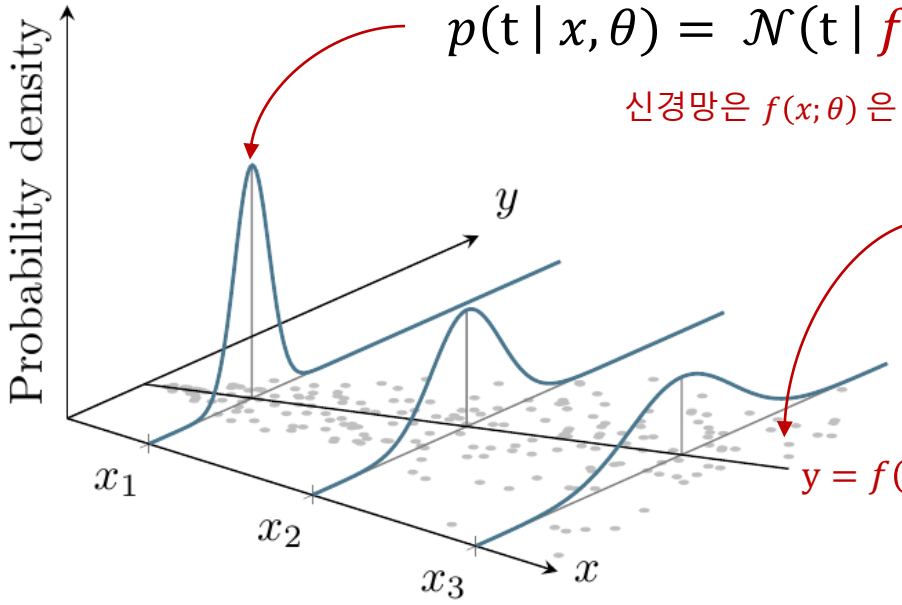
Output 연속 함수 값 예측 (Regression)

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

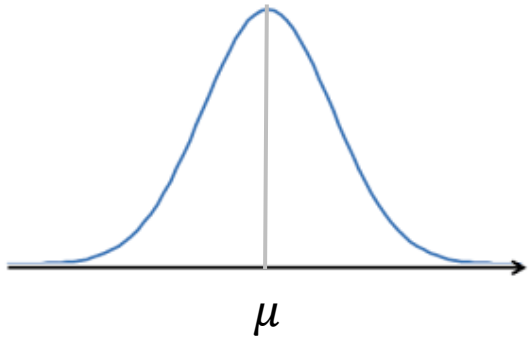
$$\mathrm{t} = f(\boldsymbol{x};\theta) + \epsilon$$

관측 오차

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

$\mathrm{y} = f(x;\theta)$ 신경망 예측 값

Gaussian Distribution

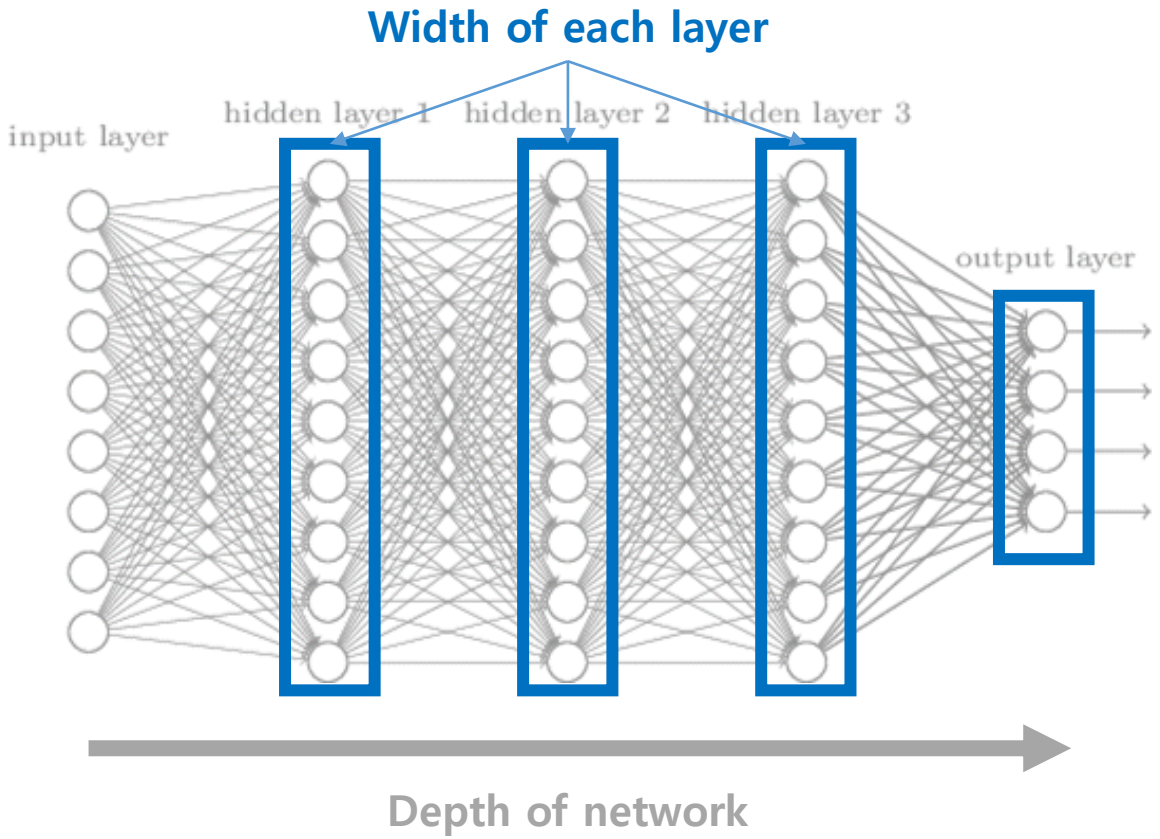


$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

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

Network Size 네트워크 크기 설계

Network의 Depth와 Width를 어떻게 정할 것인가?



네트워크 깊이가 깊을 수록

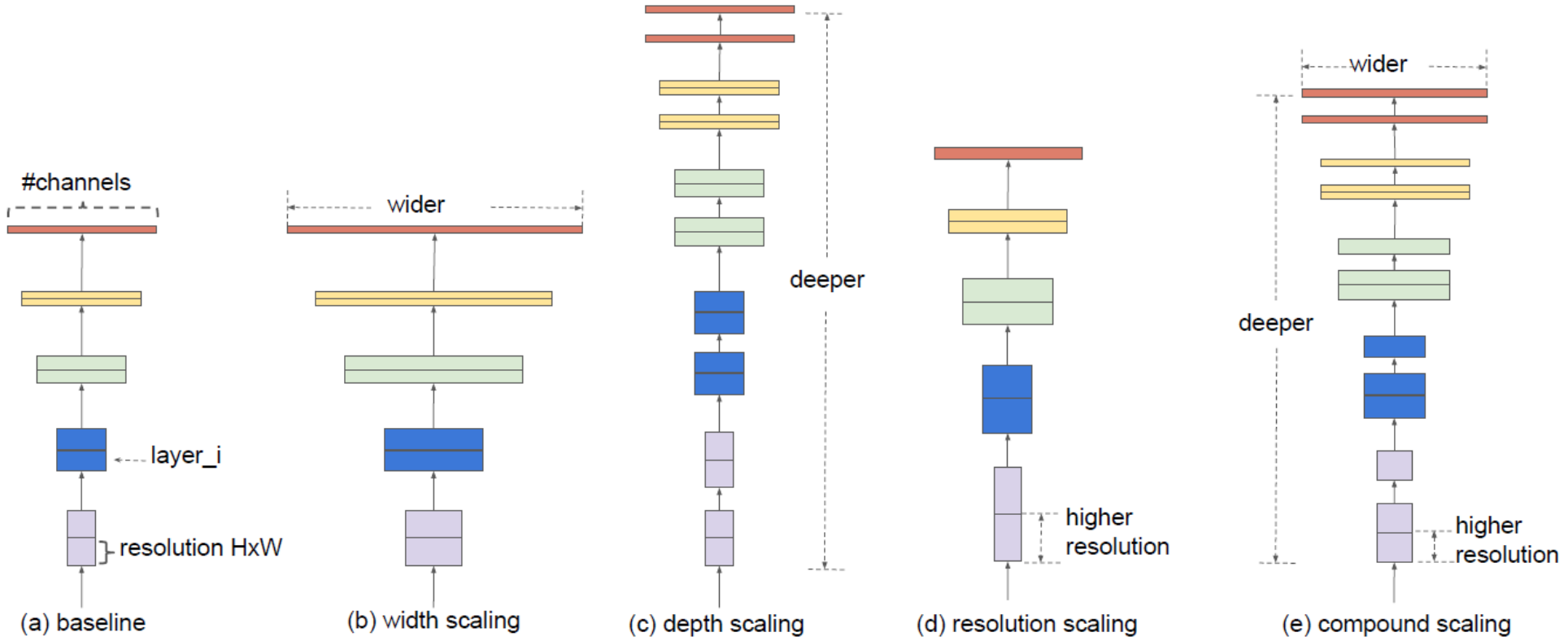
- 계층 별 뉴런을 적게 사용
- 적은 파라미터를 사용
- 일반화를 잘 함

하지만, 최적화가 어렵다.

문제에 따라 최적의 네트워크 구조는
Trial & Error로 찾아내야 한다!

Network Size 네트워크 깊이와 모델 크기

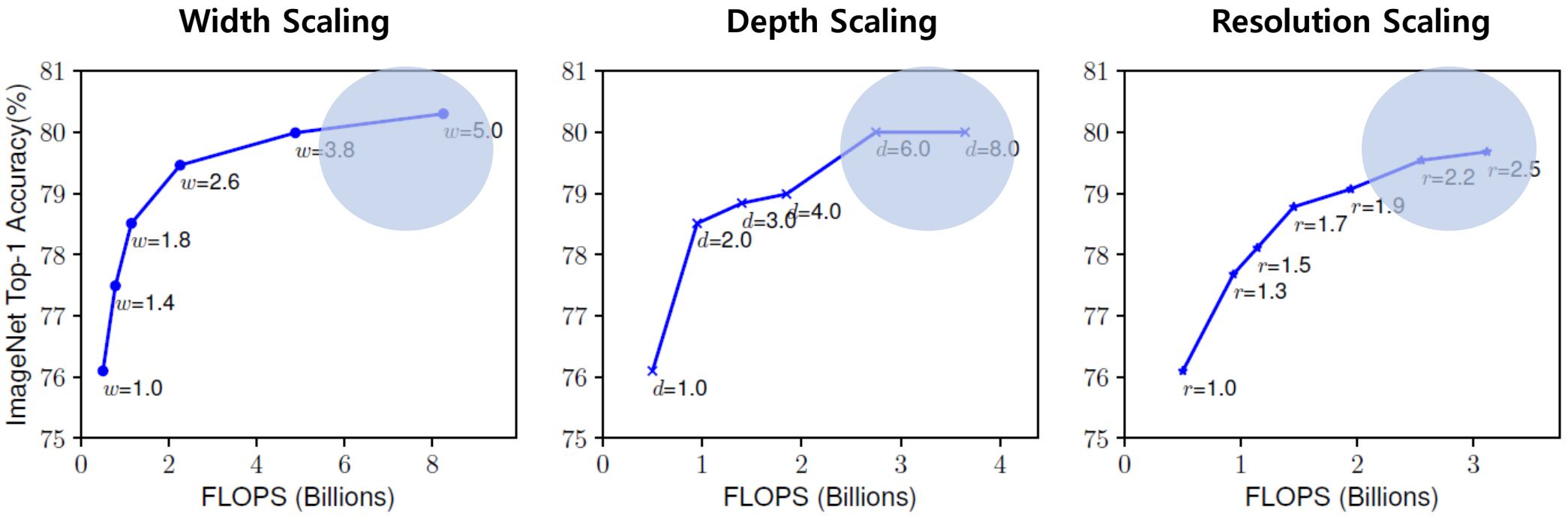
Model Scaling (Width, Depth, Resolution, Compounding)



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

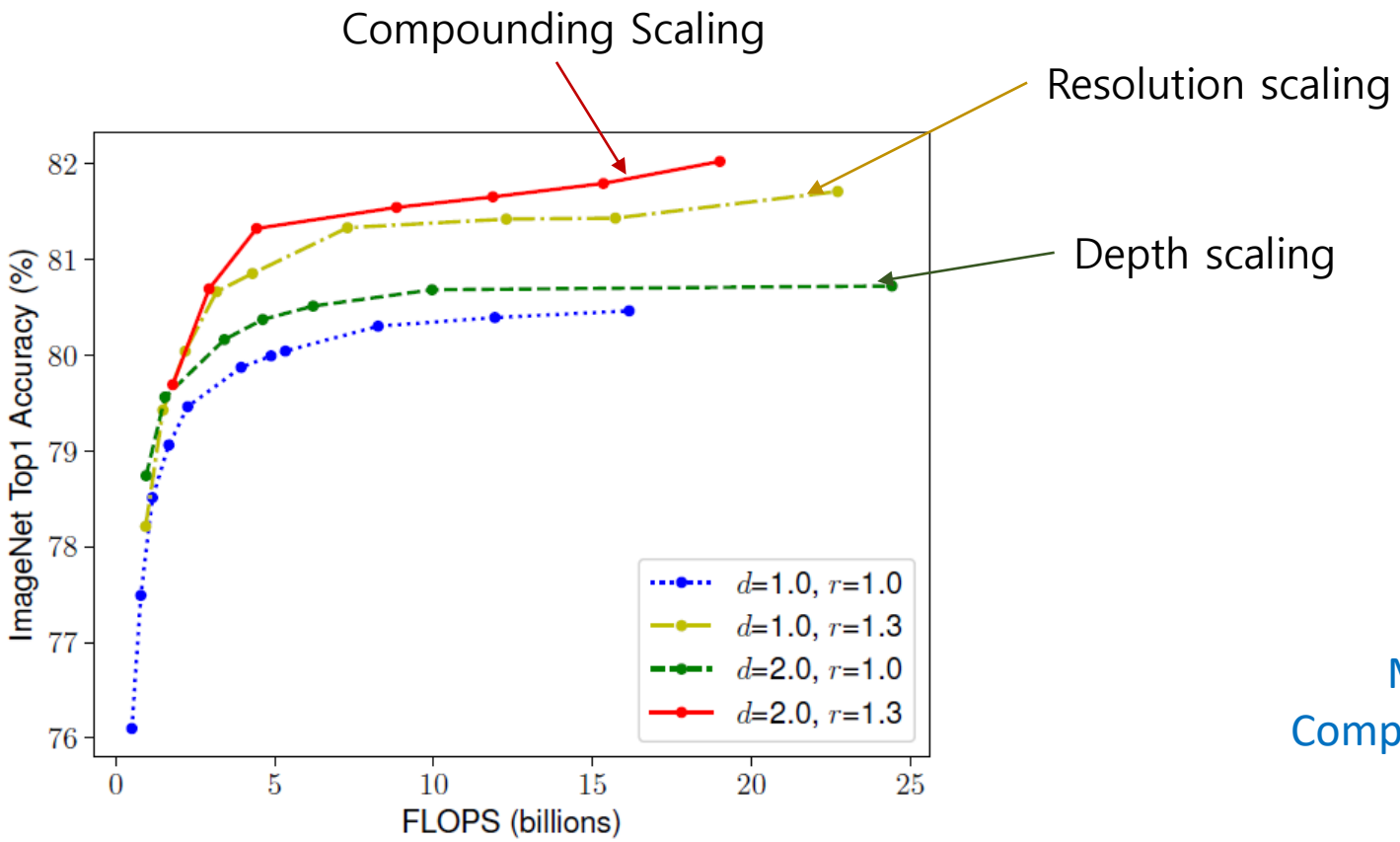
Network Size 네트워크 깊이와 모델 크기

모델이 커질수록 accuracy는 올라가지만 쉽게 saturation됨



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

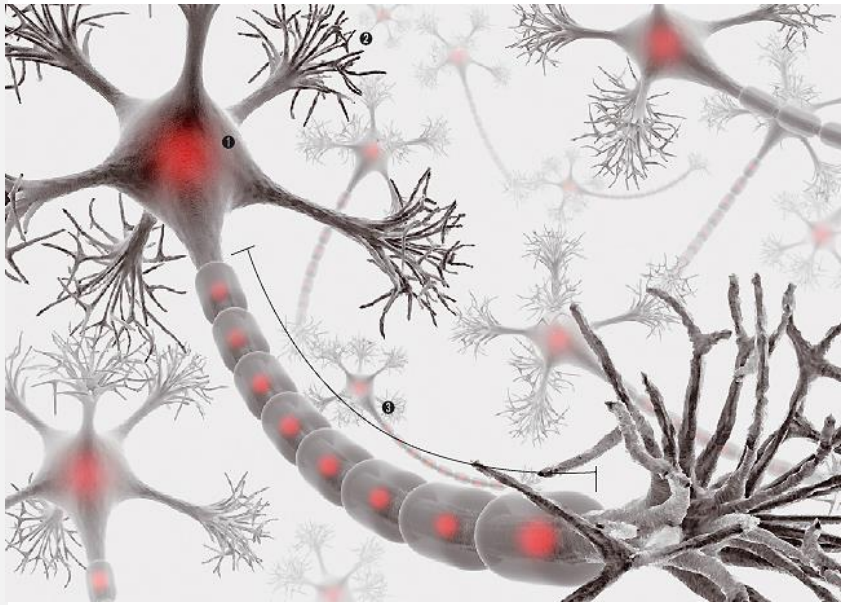
Network Size 네트워크 깊이와 모델 크기



Model Scaling을 할 경우엔
Compounding Scaling이 가장 효과적

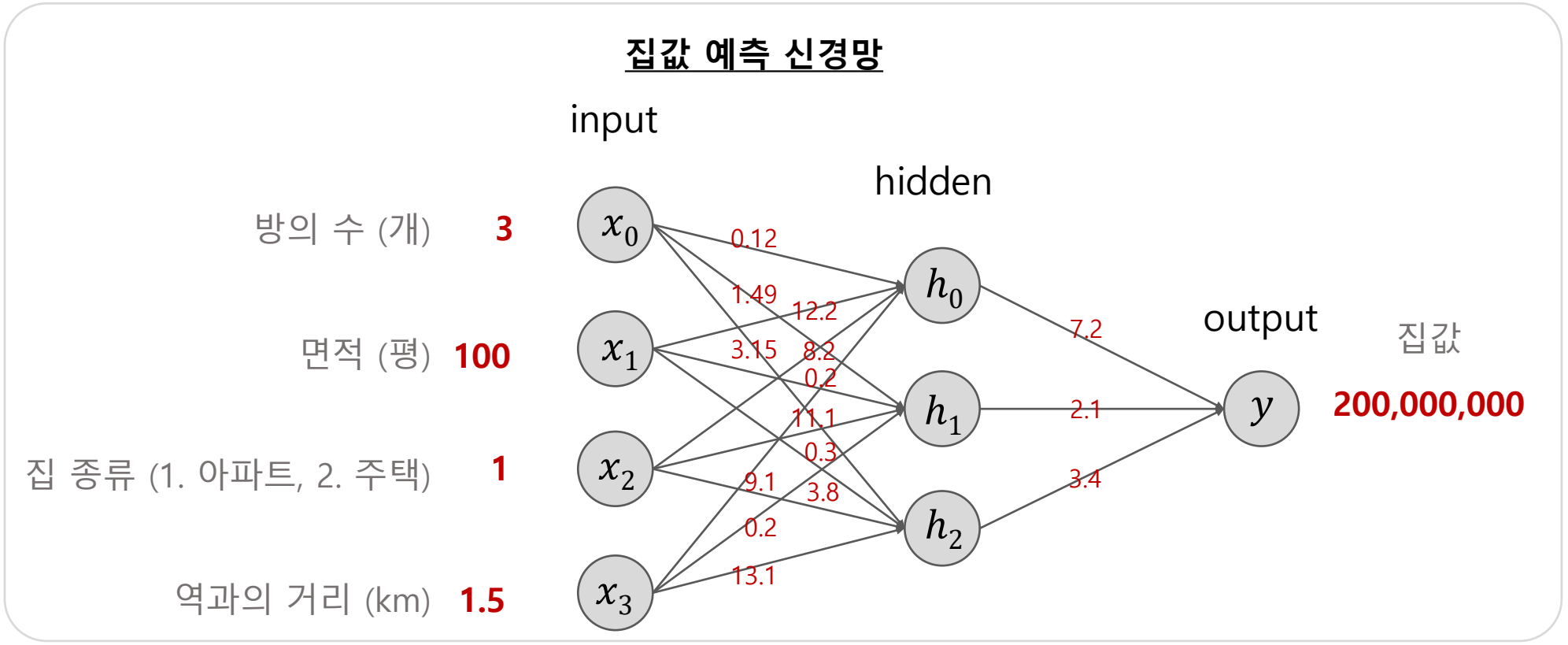
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Mingxing Tan Quoc V. Le (2019)

신경망 학습



인공 신경망의 학습

주어진 입력과 출력 데이터를 이용해서 신경망 스스로 함수를 찾아내는 것



신경망 스스로 파라미터를 찾아 함수를 정의하는 것을 학습이라고 한다!

최적화 문제

회귀 (Regression)

타깃과 인공신경망이 예측한 값의 차이를 최소화하는 파라미터를 찾아라.

파라미터 → $\min_{\theta} \frac{1}{n} \sum (t - f(x;\theta))^2$

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

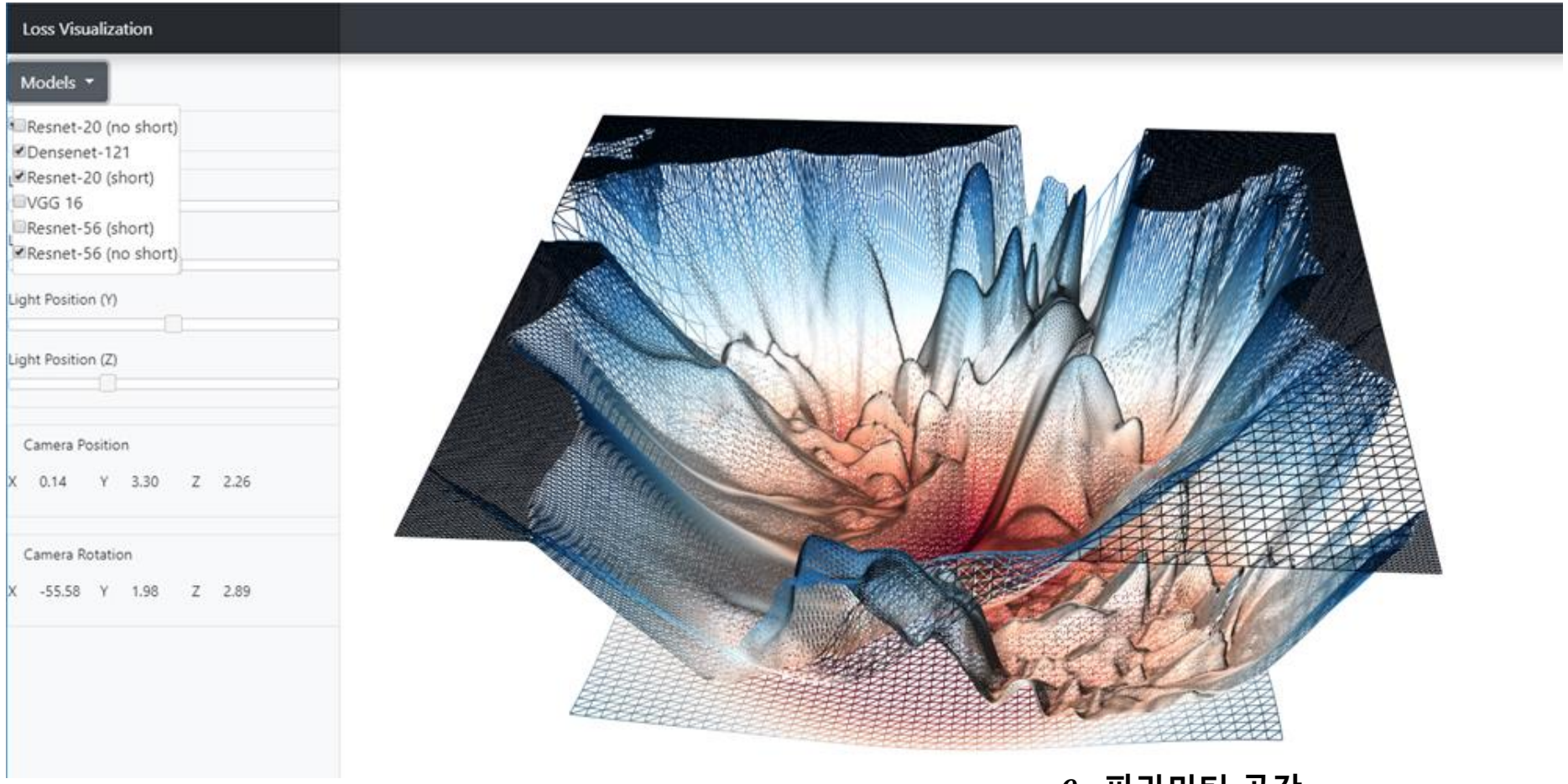
K : Class 개수

관측 분포 ↑ t_k

모델이 예측한 분포 ↑ $f(x;\theta)_k$

크로스 엔트로피 (Cross Entropy)

Loss Surface



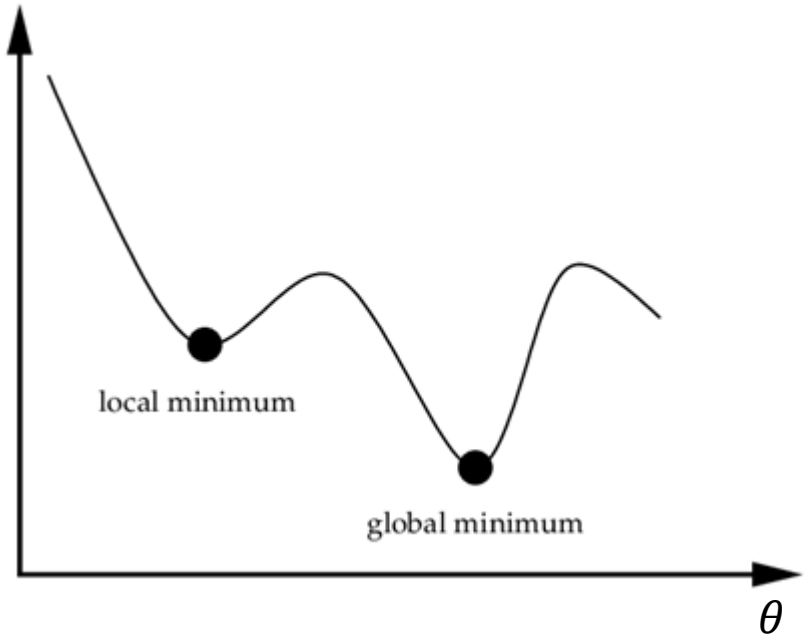
θ : 파라미터 공간

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

Loss를 최소화 하려면?

Loss Minimization

$J(f(x;\theta),t)$



최적화 알고리즘

1차 미분

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

Deep Learning에서 주로 사용하는 방법

1.5차 미분

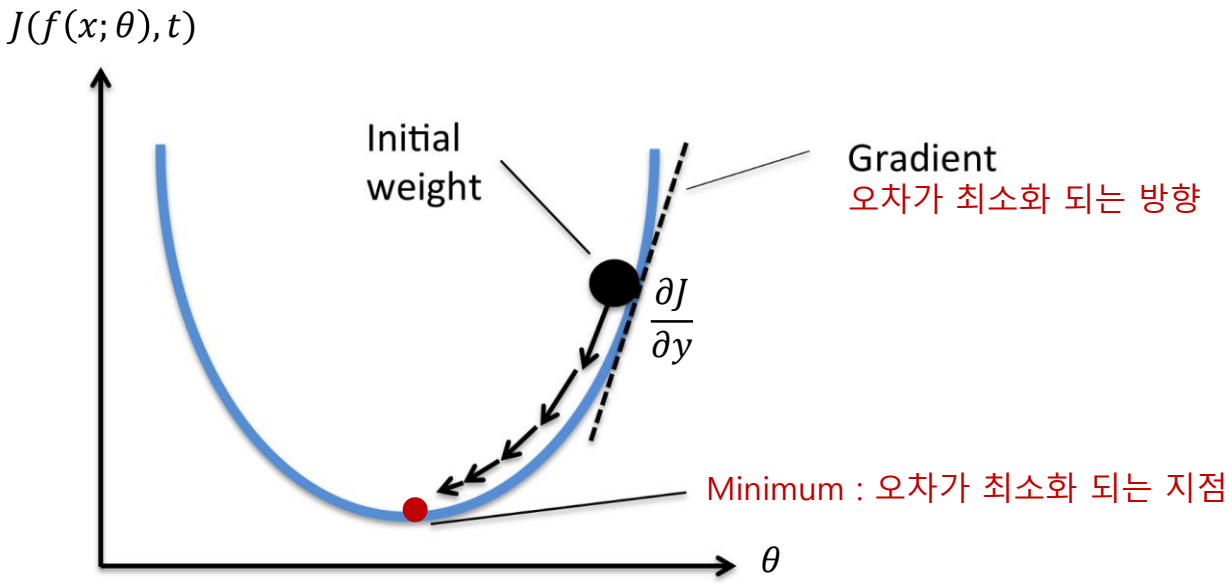
- Quasi-Newton Method
- Conjugate Gradient Descent
- Levenberg-Marquardt Method

2차 미분

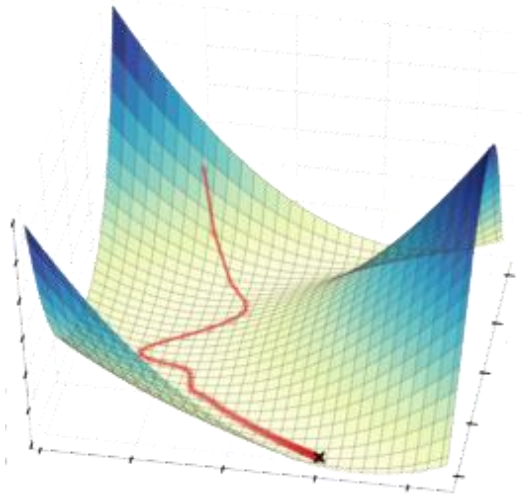
- Newton Method
- Interior Point Method

Gradient Descent

Gradient Descent



3D View

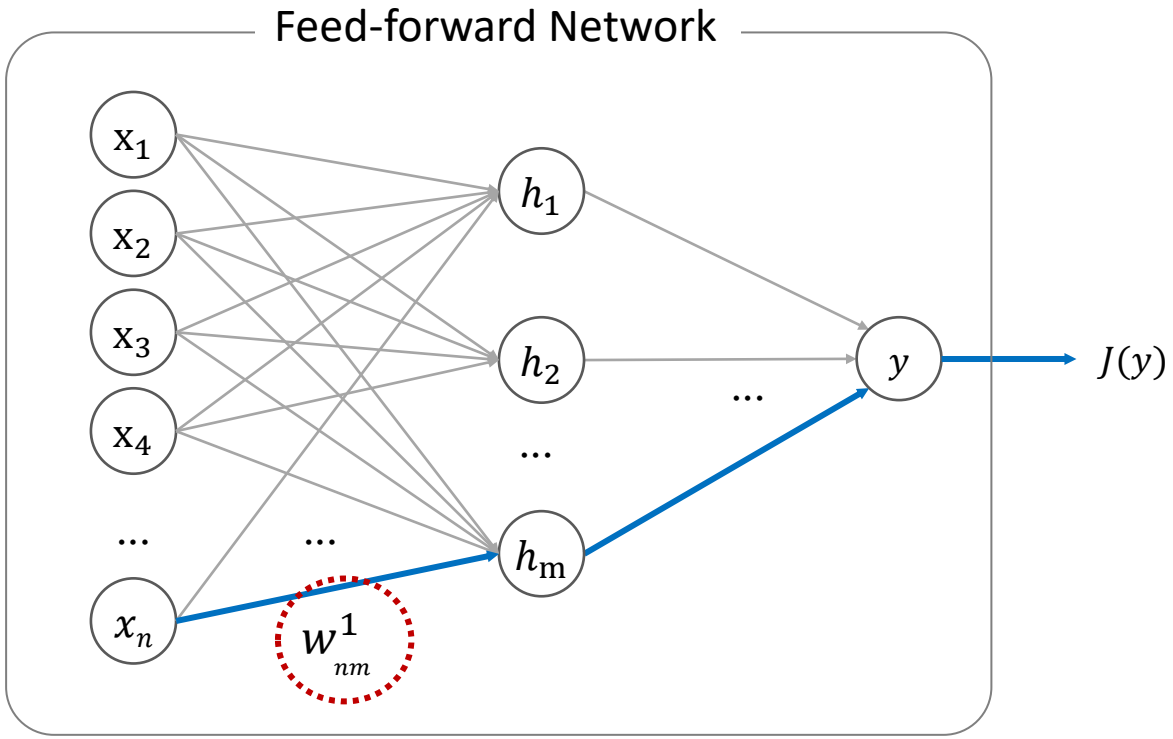


Parameter Update

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

Step Size — Gradient

Gradient Descent



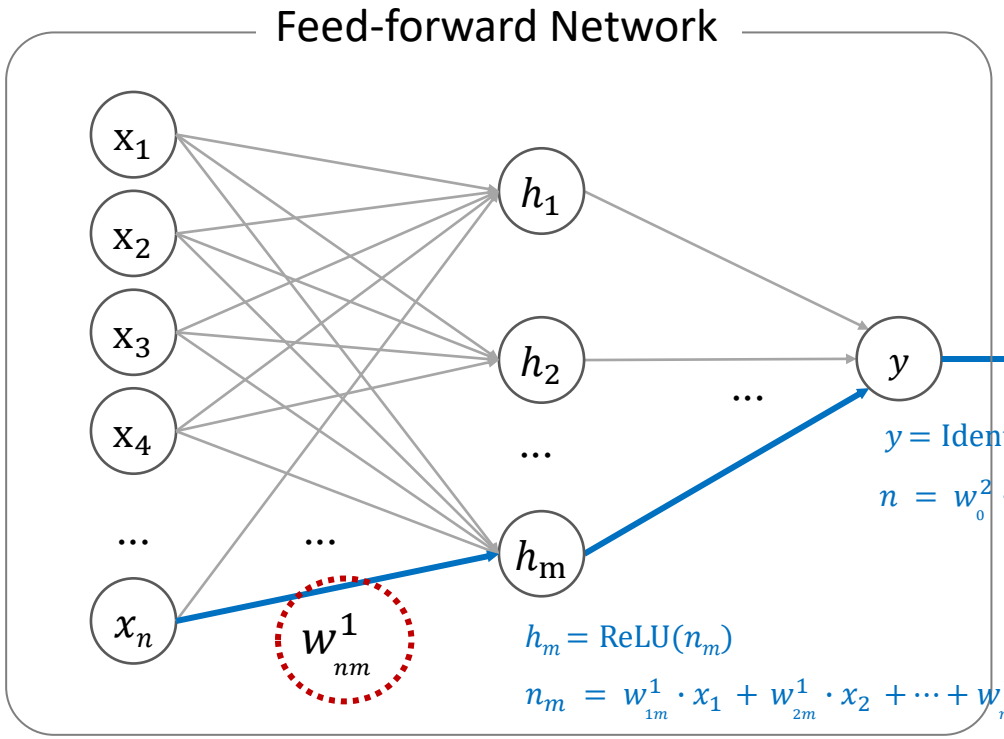
Parameter Update

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

Step Size

Gradient

Gradient Descent



Gradient of Parameter

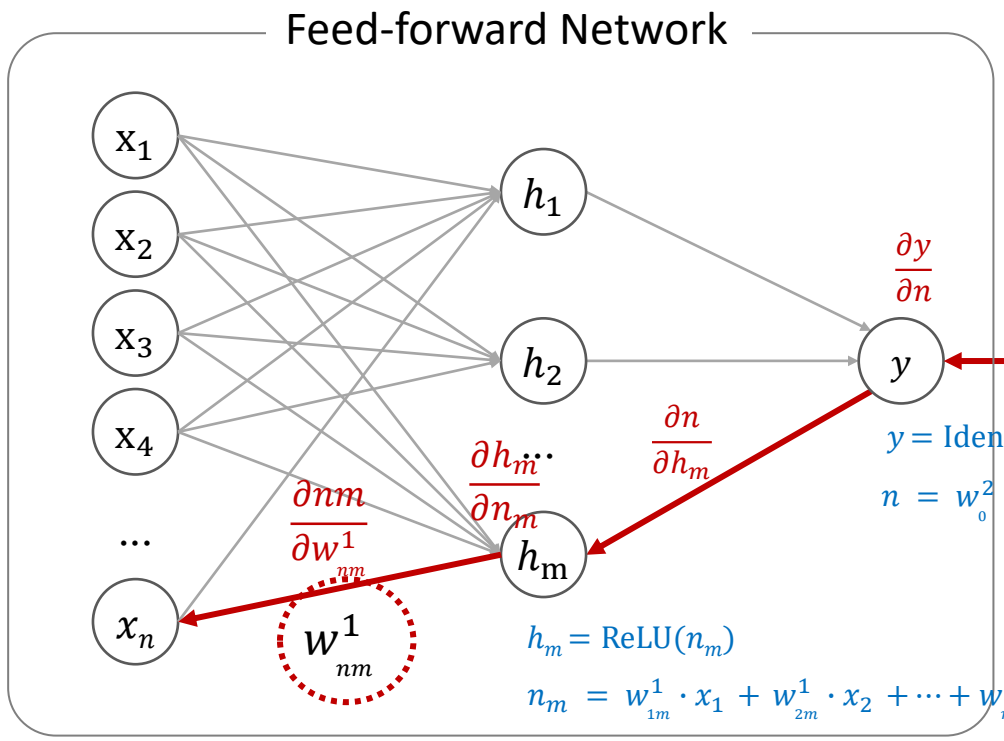
$$w^1_{nm}{}^{+} = w^1_{nm} - \alpha \frac{\partial J}{\partial w^1_{nm}}$$

Step Size

Gradient

"가중치는 Loss Function의 간접 파라미터이므로
직접 미분이 안됨"

Backpropagation



Gradient of Parameter

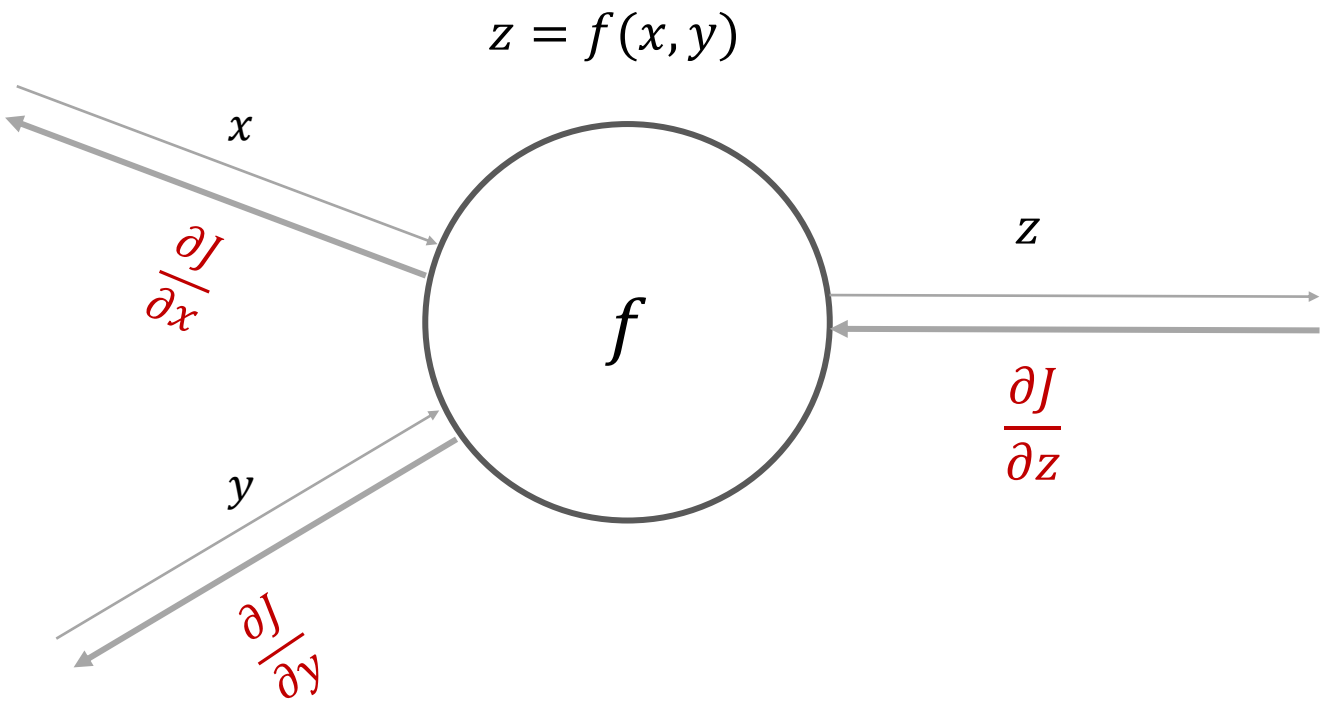
$$\frac{\partial J}{\partial w^1_{nm}} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial n} \cdot \frac{\partial n}{\partial h_m} \cdot \frac{\partial h_m}{\partial n_m} \cdot \frac{\partial n_m}{\partial w^1_{nm}}$$

연쇄 법칙 (Chain Rule) 사용

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

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

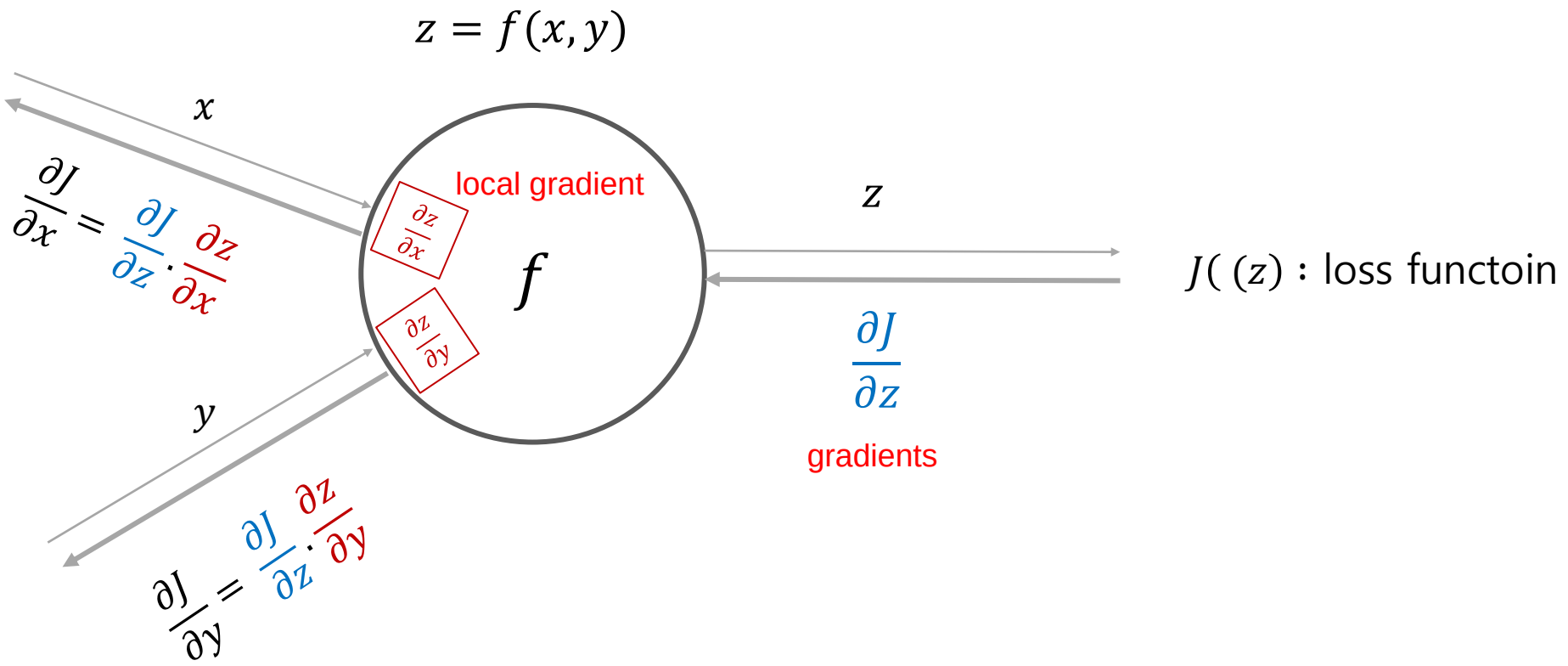
Backpropagation



$J(z)$: loss functoin

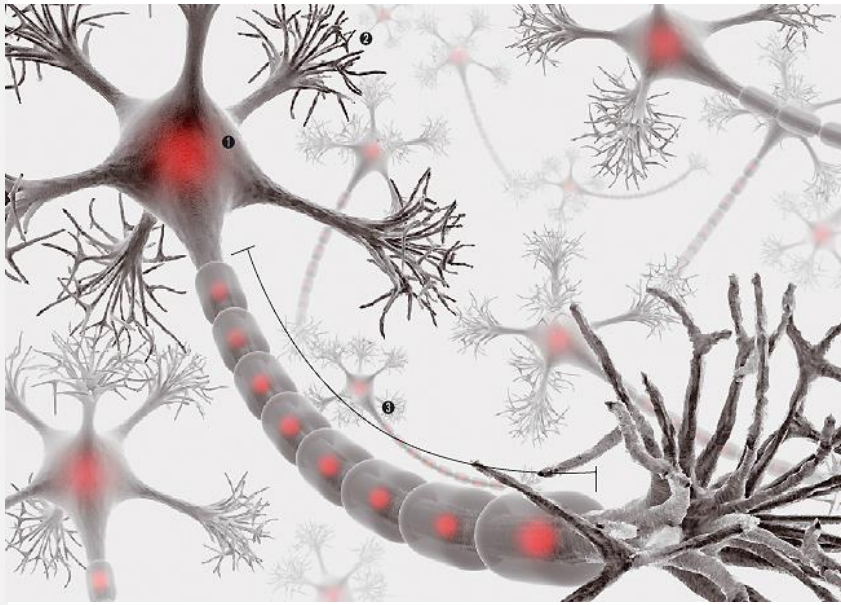
Loss $J(z)$에 대해 x, y, z의 미분을 구하라!

Backpropagation



각 노드에서 Local Gradient를 구한 후 전달 받은 Gradient와 곱해서 이전 노드에 전달

TensorFlow



Eager Execution

- 1.x : tf.enable_eager_execution()
- 2.x: Default
- **Define by Run** support (like PyTorch, Chainer)
- Rapid Development
- **Easy Debugging** (use Python toolchain) → easy to check bottlenecks
- Native Control Flow (if, for etc) → easy to make complex model
- Boost performance by **AutoGraph**

Keras in TensorFlow2.0 by 박해선님

Eager Execution

Define and Run에서 Define by Run 으로!

```
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

```
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

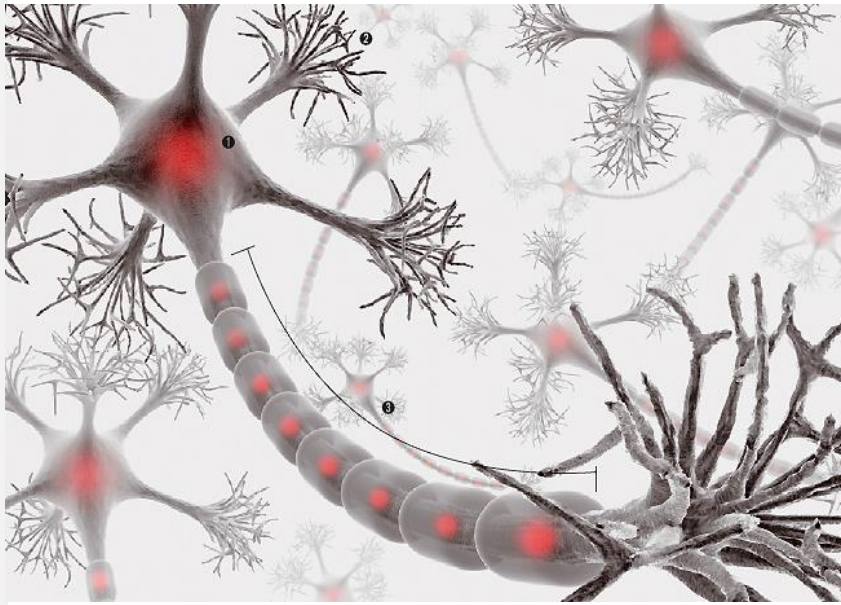
TensorFlow 1.x

```
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))
```

Keras



Keras

- **High-Level Neural Networks Specification** (https://keras.io) (2015. 03)
- Add to tf.contrib.keras at TensorFlow 1.2
- Promote to tf.keras at TensorFlow 1.4 (tf.layers → tf.keras)
- 1st Class Python API for TensorFlow 2.0
- Deprecated tf.layer, tf.contrib.layers(Slim)
- Keras 2.3.x is last major release of multi-backend Keras.
  Instead use tf.keras

Keras in TensorFlow2.0 by 박해선님

Class Hierarchy

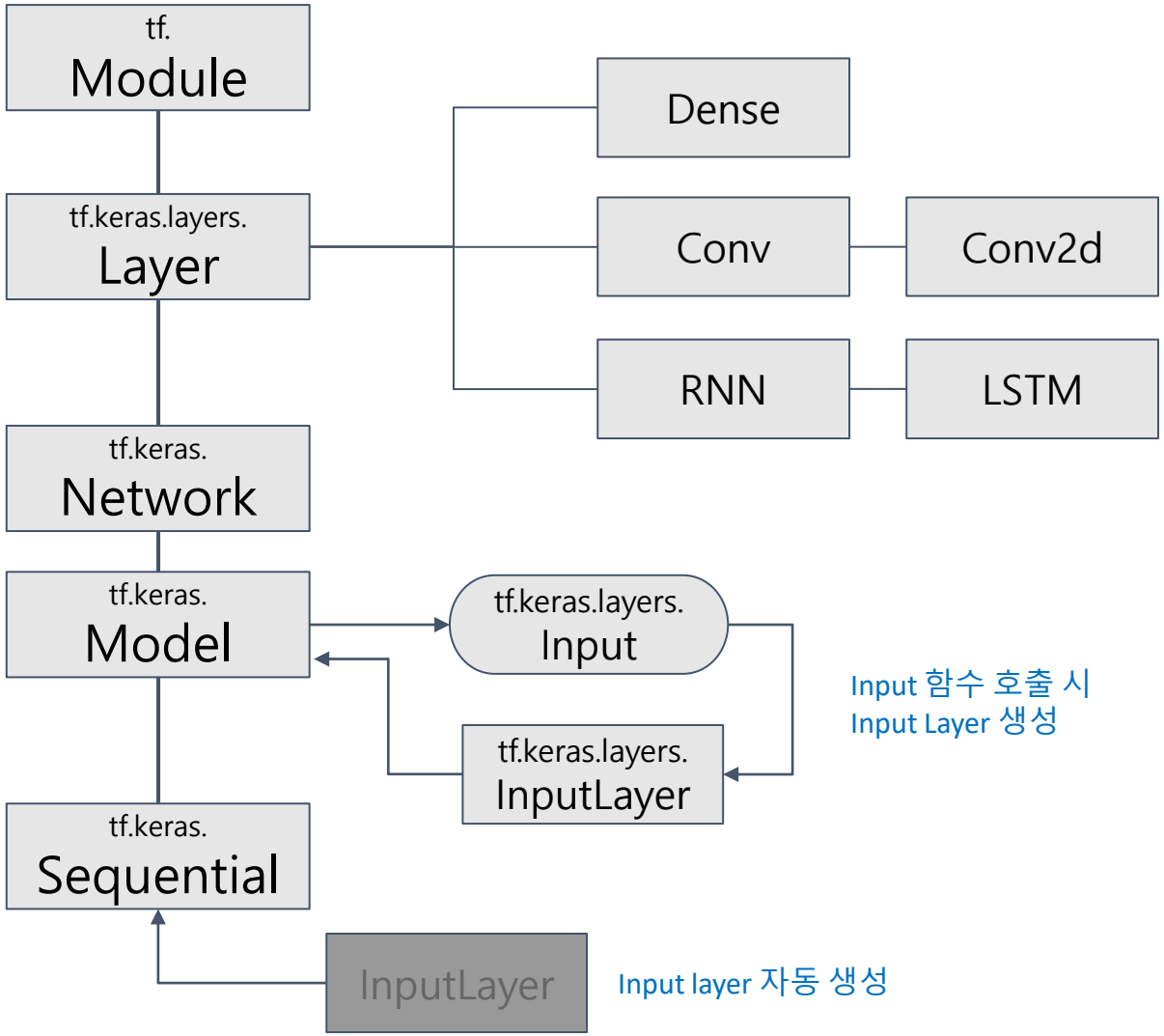
변수 컨테이너 (tf.Variable)
variables(), trainable_variables()

계층 정의 (파라미터, Forward Pass)
__call__() → build() → add_weights()
|→ call()
add_loss()

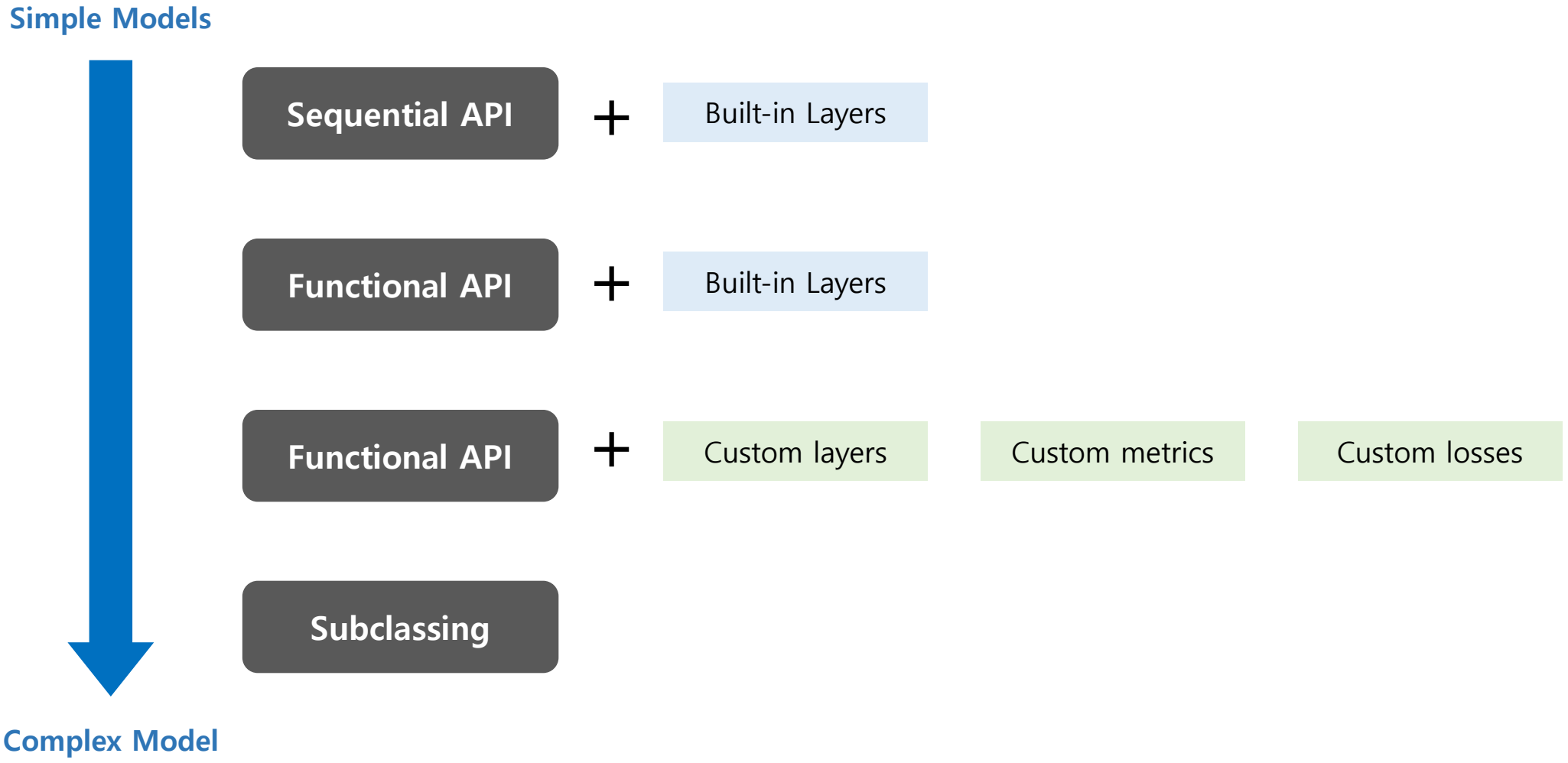
신경망 계층 통합
layers(), summary(), save()

모델 훈련/검증/테스트
compile(), fit(), evaluate(), predict()

순차 모델 구성
add()

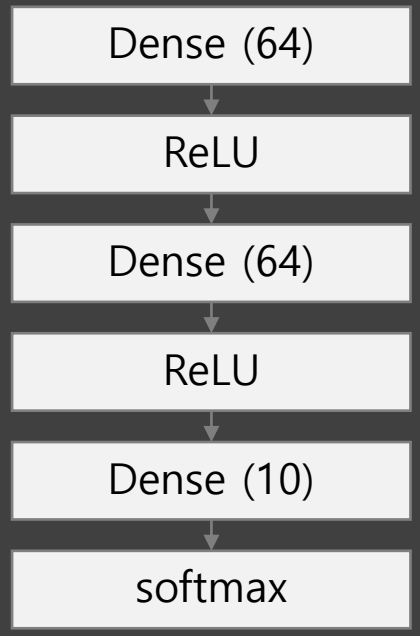


Keras 모델 정의



Sequential API

```
from tensorflow import tf

model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# 훈련 설정
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 모델 훈련
model.fit(train_data, labels, epochs=10, batch_size=32)

# 모델 평가
model.evaluate(test_data, labels)

# 샘플 예측
model.predict(new_sample)
```




```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(10),
])
```

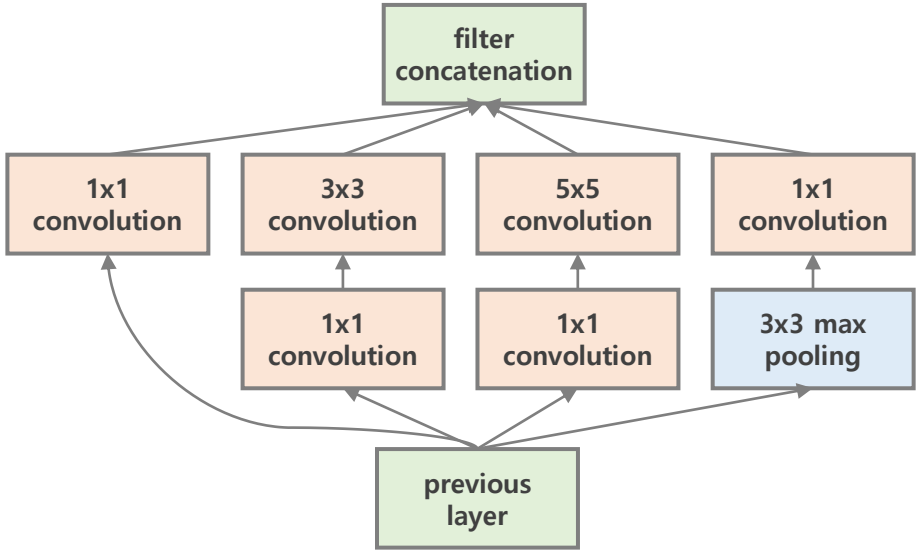
Functional API

```
from tensorflow import tf

# 입력과 출력을 연결해서 임의의 모델 그래프 생성
input = tf.keras.Input(shape=(784,), name='img’)  # 입력 플레이스 홀더 반환
h1 = tf.keras.layers.Dense(64, activation='relu')(inputs) # 각 계층 별로 Tensor를 전달하고 리턴 받음
h2 = tf.keras.layers.Dense(64, activation='relu')(h1)
output = tf.keras.layers.Dense(10, activation='softmax')(h2)

# 모델 생성
model = tf.keras.Model(input, output) # 입력 Tensor와 Output Tensor를 모델에 지정

# 훈련 설정
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
                    loss='categorical_crossentropy',
                    metrics=['accuracy’])
…
```

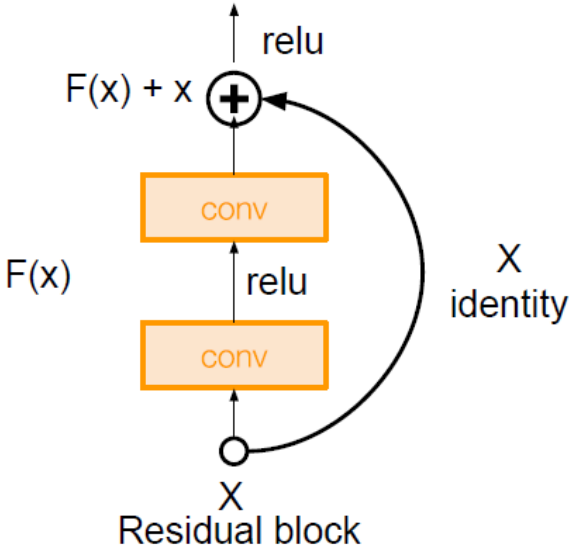
Keras in TensorFlow2.0 by 박해선님

Functional API

“Inception module”



“Residual block”



- 다중 입력 모델
- 다중 출력 모델
- 층을 공유하는 모델 (동일한 층을 여러 번 호출합니다)
- 데이터 흐름이 차례대로 진행되지 않는 모델 (예를 들면 잔차 연결(residual connections)).

Custom Layer

```
from tensorflow import tf
class MyLayer(tf.keras.layers.Layer):

    def __init__(self, units, activation=None, **kwargs):
        self.units = units
        self.activation = keras.activations.get(activation)
        super().__init__(**kwargs)

    def build(self, input_shape):
        self.weight = self.add_weight(name='kernel',
                                      shape=(input_shape[1], self.units),
                                      initializer='uniform')
        self.bias = self.add_weight(name='bias',
                                    shape=(self.units,),
                                    initializer='zeros')
        super().build(input_shape)

    def call(self, X):
        z = tf.matmul(X, self.weight) + self.bias
        return self.activation(z)
```

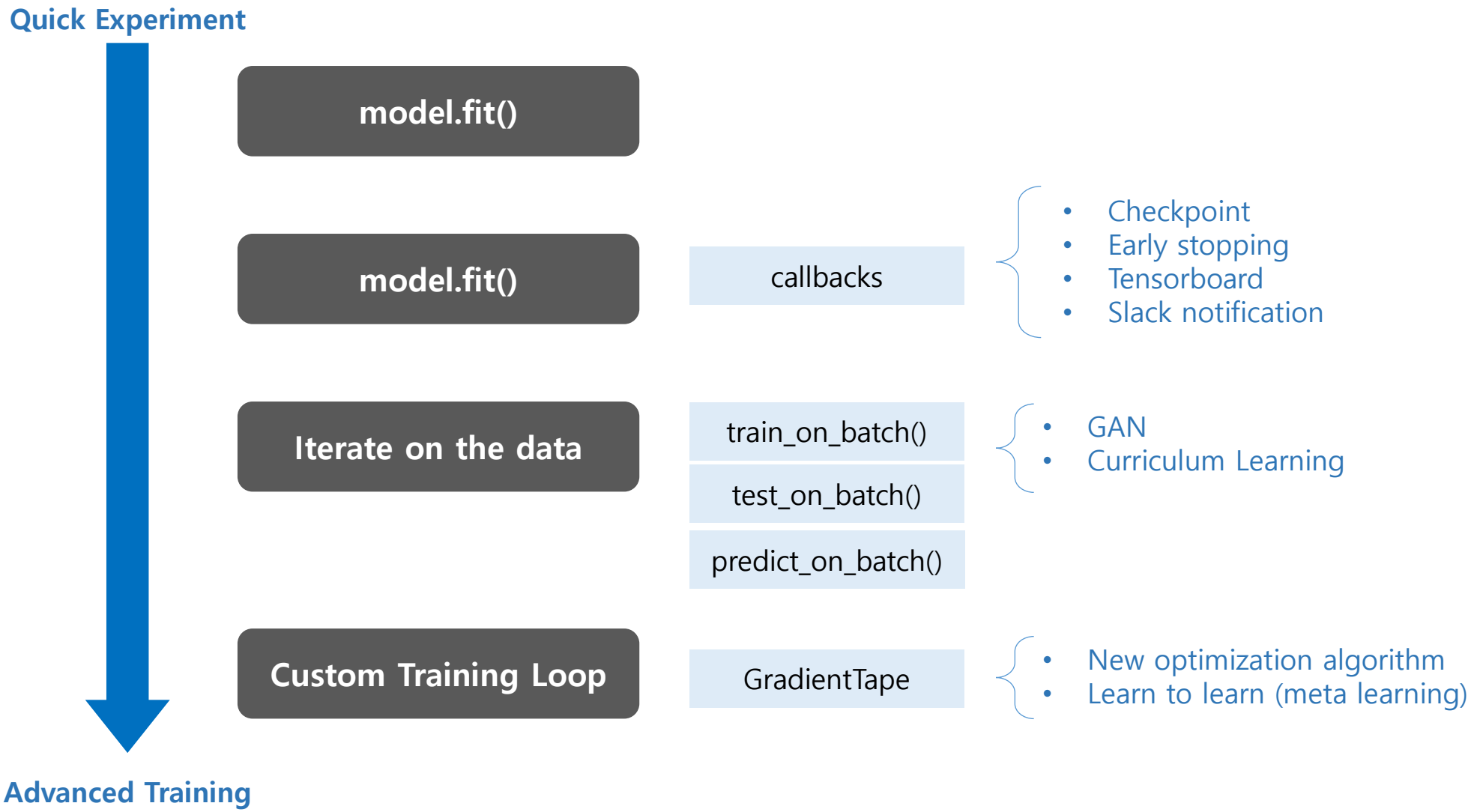
Keras in TensorFlow2.0 by 박해선님

Custom Model

```
from tensorflow import tf

class MyModel(tf.keras.Model):

    def __init__(self, **kwargs):
        self.hidden = MyLayer(10, activation="relu")
        self.output = MyLayer(1)
        super().__init__(**kwargs)

    def call(self, input):
        h = self.hidden(input)
        return self.output(h)

model = MyModel()
```

Keras in TensorFlow2.0 by 박해선님

Training 방식



model.compile

훈련에 필요한 Optimizer, Loss, Metric을 설정하는 단계

1. 이름으로 지정 (Default 값으로 실행할 때)

회귀 모델 예시

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss='mse',       # 평균 제곱 오차
              metrics=['mae'])  # 평균 절댓값 오차
```

2. 객체를 생성해서 전달 (파라미터를 지정할 필요가 있을 때)

분류 모델 예시

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(0.01),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=[tf.keras.metrics.CategoricalAccuracy()])
```

model.fit

모델을 고정된 epoch 수로 훈련

```
history = model.fit( normed_train_data, train_labels,
                     epochs=1000, validation_split = 0.2, verbose=0,
                     callbacks=[Earlystopping(),
                                Tensorboard(),
                                ModelCheckpoint()])
```

- **batch_size**: 배치 크기 (default 32)
- **epochs**: 총 epoch 수 (epoch는 training set을 한번 실행하는 단위)
- **validation_split**: training set에서 validation set으로 사용할 비율 ((0,1) 사이의 값)
- **verbose**: 훈련 진행 상황 모드 0 = silent, 1 = progress bar, 2 = one line per epoch
- **callbacks**: 훈련하면서 실행할 콜백 리스트

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
```

Forward Pass

Gradient 계산

Parameter Update

Thank you!