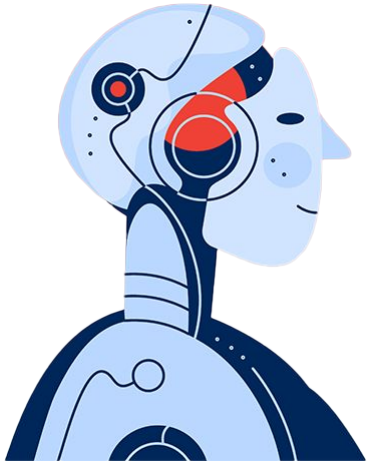
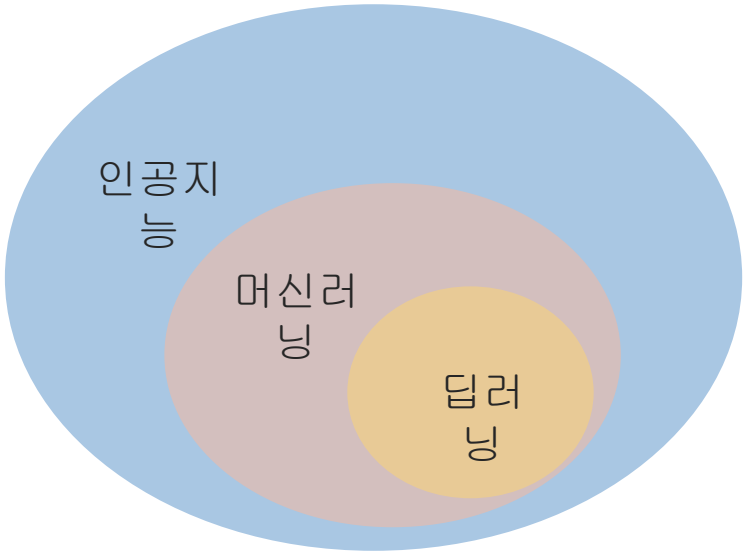
딥러닝

딥러닝 이해

딥러닝



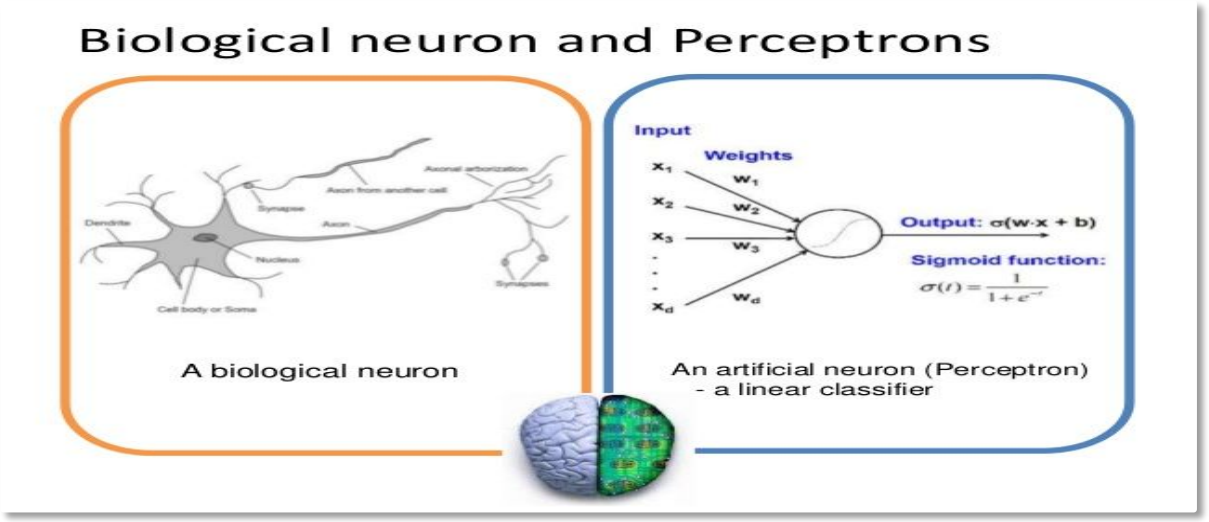


- 딥러닝
  신경망을 층층히 쌓아서 문제를 해결하는 기법의 총칭

- 머신러닝
  데이터를 이용해서 명시적으로 정의되지 않은 패턴을 컴퓨터로 학습하여 결과를 만들어내는 학문 분야

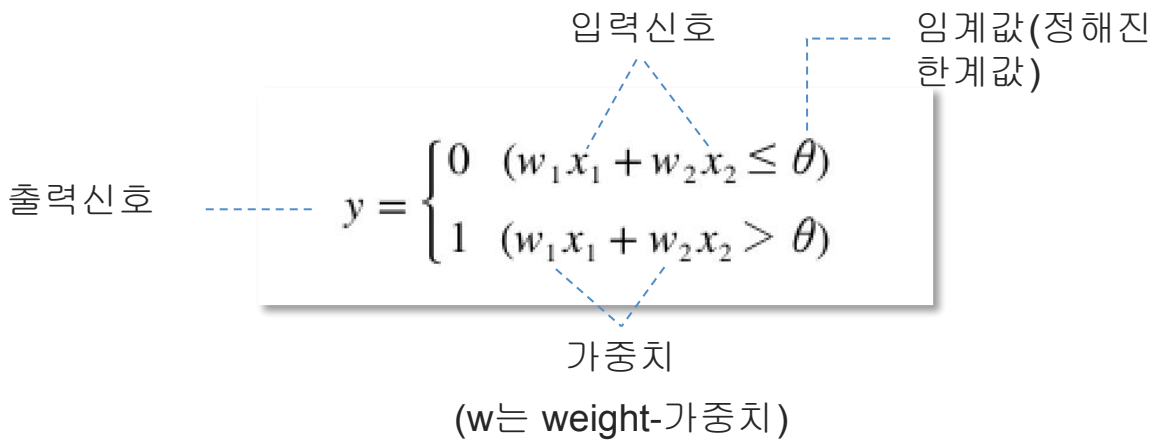
Perceptron

- 퍼셉트론(인공뉴런)
  - 1957년에 고안된 알고리즘
  - 딥러닝(신경망)의 기원이되는 알고리즘
  - 간단한 형태의 선형 분류기이자 일종의 수식
  - 다수의 신호(x1, x2, ...)를 입력받아 하나의 신호(y)를 출력
    → 신호는 전류나 물의 흐름을 생각해 볼 수 있음
  - 신호의 흐름을 표현할 때 두가지 값을 갖는데 0, 1의 값을 갖음
    → 0은 신호가 흐리지 않음, 1은 신호가 흐름



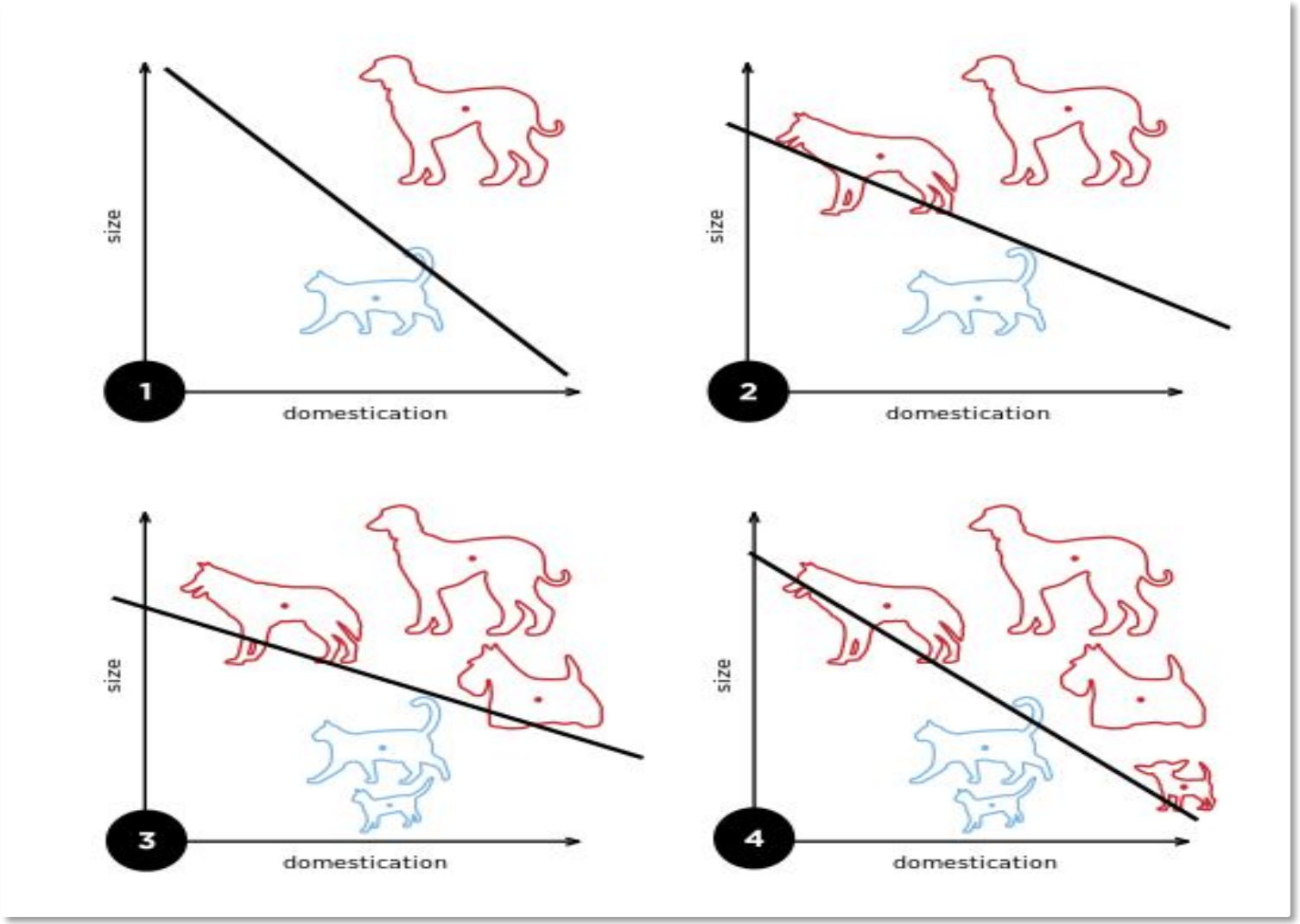
Perceptron

- 퍼셉트론 수식



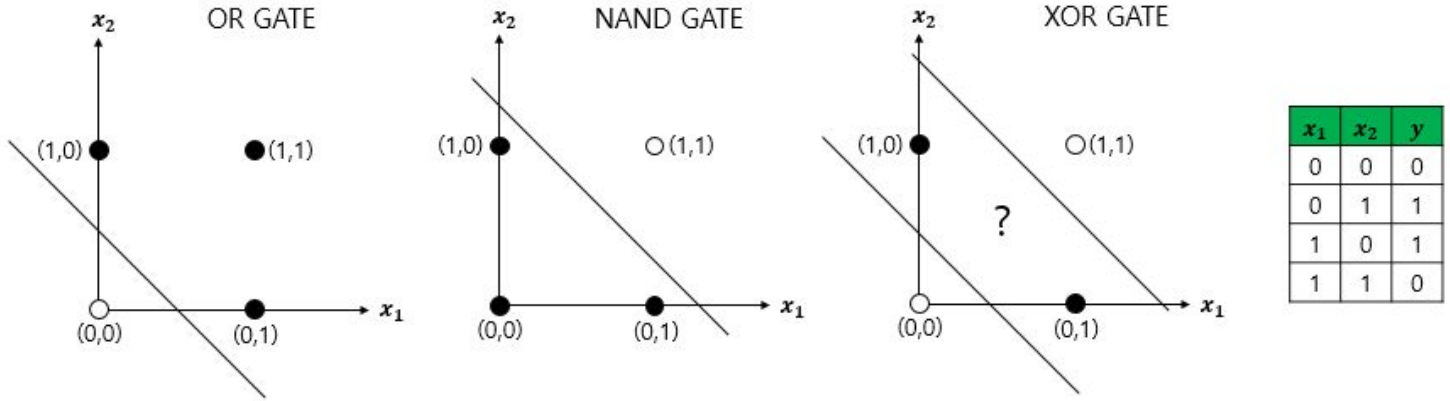
$w_1x_1 + w_2x_2$ 값이 임계값(θ) 이하 일때는 0을 출력,
임계값보다 클때 1을 출력

Perceptron

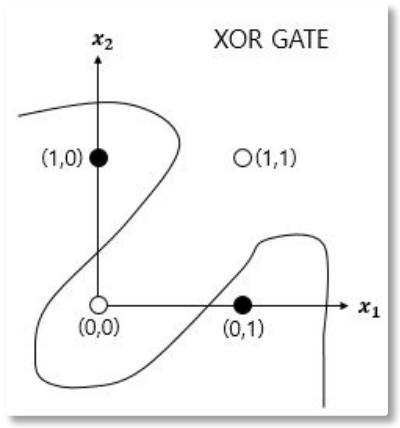

size
1
domestication
size
2
domestication
size
3
domestication
size
4
domestication

Perceptron의 과제

- **단층 퍼셉트론 한계**
  - 직선에 의해서만 분할 가능하므로 직선으로 비선형 분할 불가능



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



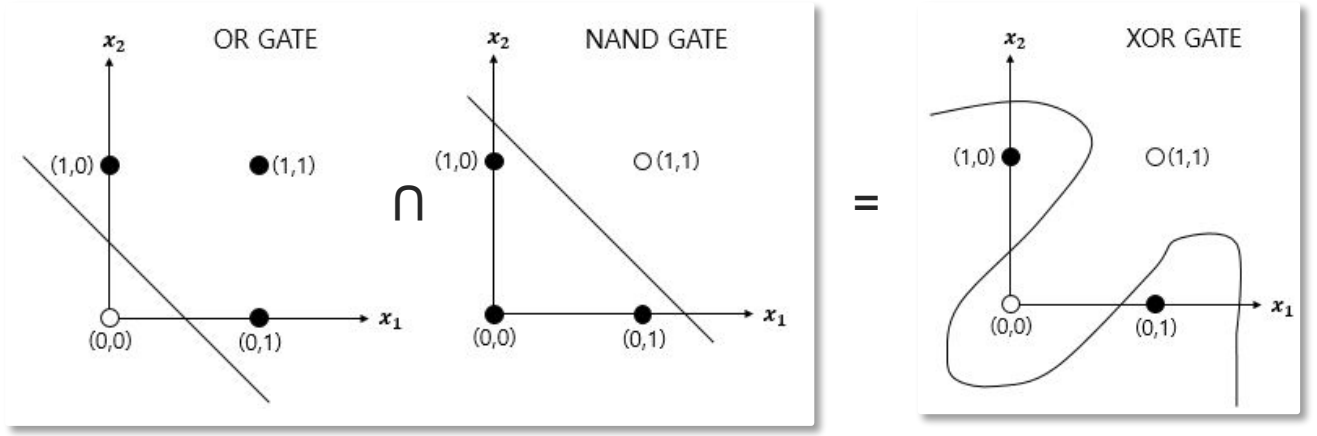
XOR 게이트
Input(x1, x2)중 1이 한 개만 존재하면 y=1인 게이트

직선으로 분할 불가능
즉, 비선형 분할만 가능!!

Perceptron의 과제

- **다층 퍼셉트론(Multi Layer Perceptron)**
  - 다수 퍼셉트론으로 연결하여 복합적 값을 전달
  - 단층 퍼셉트론의 한계를 극복
  - 단층 퍼셉트론으로는 XOR게이트를 표현할 수 없음

    → 즉, 단층 퍼셉트론으로는 비선형 영역을 분리할 수 없음
  - 기존 게이트(AND, OR, NAND) 조합하여 층을 쌓으면 XOR 게이트를 구현할 수 있음

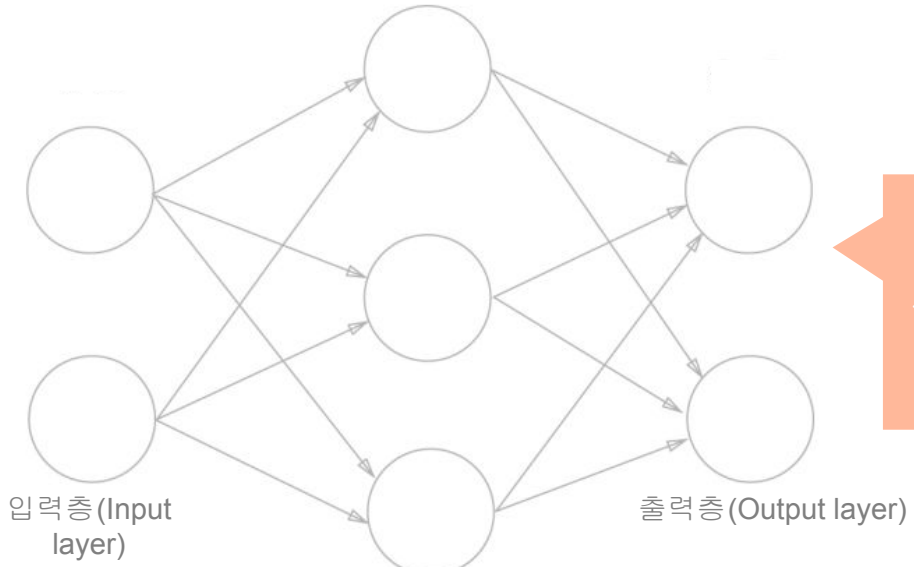


<단층 퍼셉트론의 한계 해결>

ANN (Artificial Intelligence)

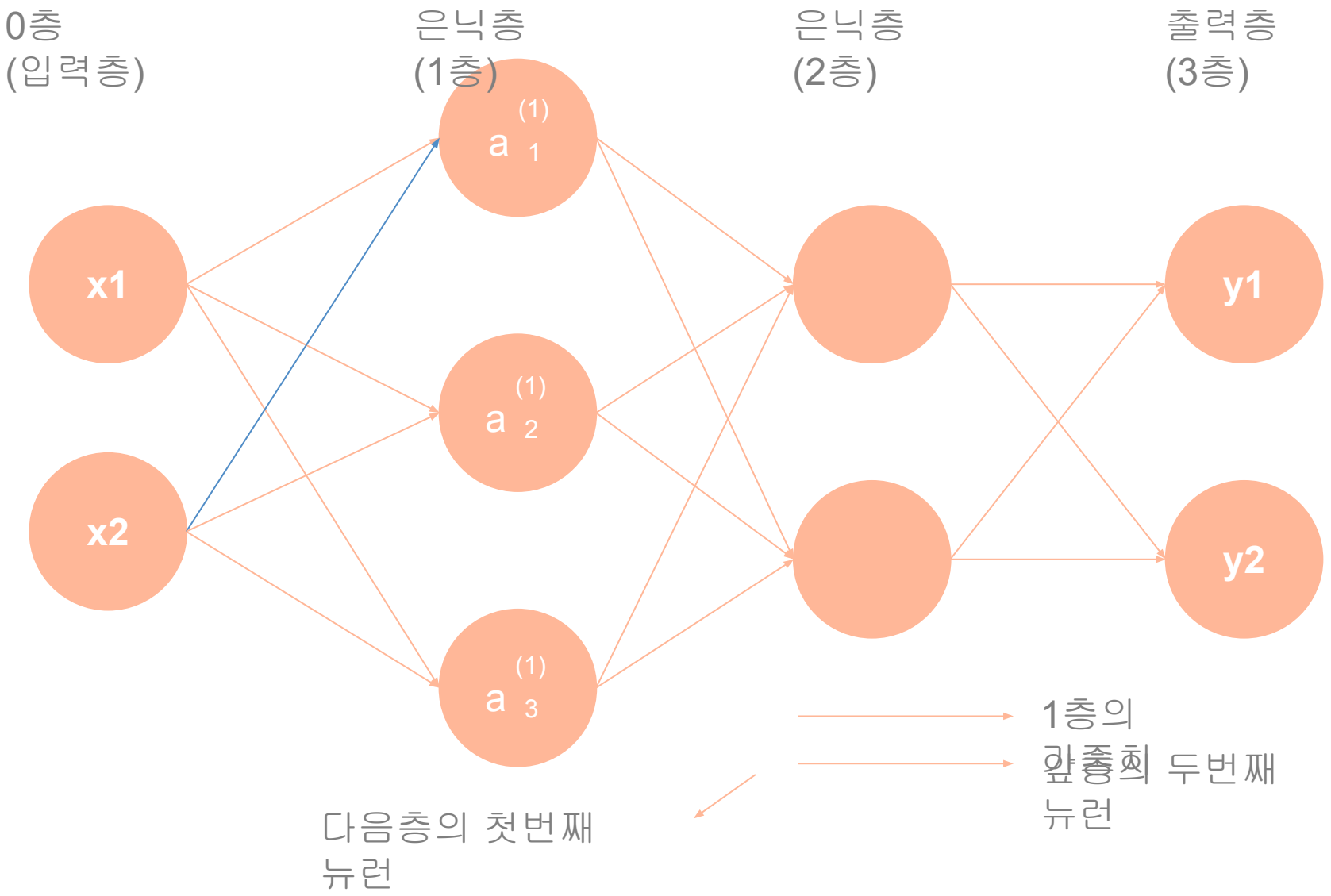
- 인공
  신경망
  - 이미지 분류, 텍스트 마이닝 등등 많은 일들을 할 수 있음

- 신경
  망
  - 퍼셉트론 생성 시 사용자가 수동으로 제공하던 가중치를 자동으로 입력되도록 할 수 있기 위해 개념화



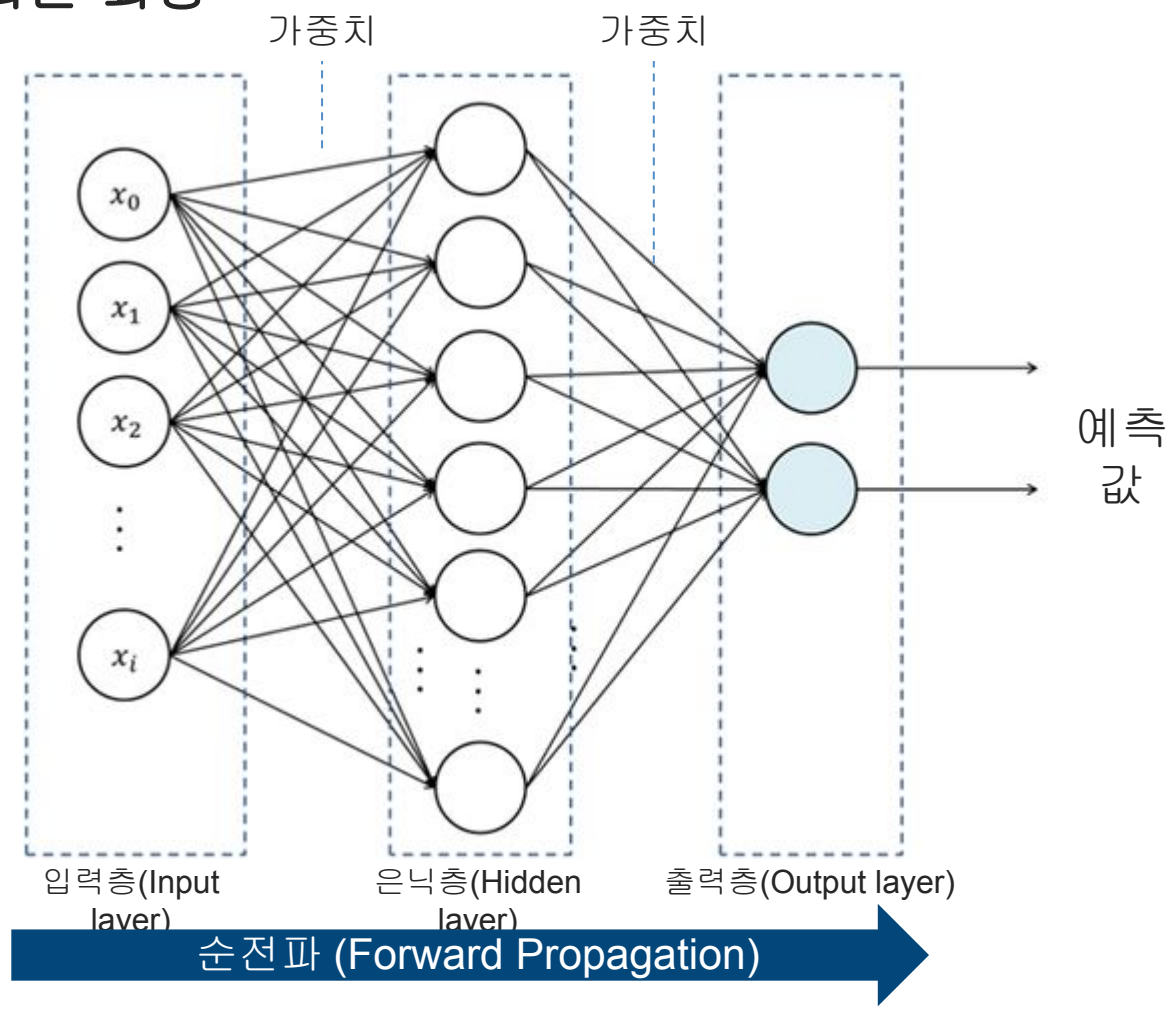
다층 퍼셉트론의 신호전달 방식과 공통점 있음

DNN (Deep Neural Network)



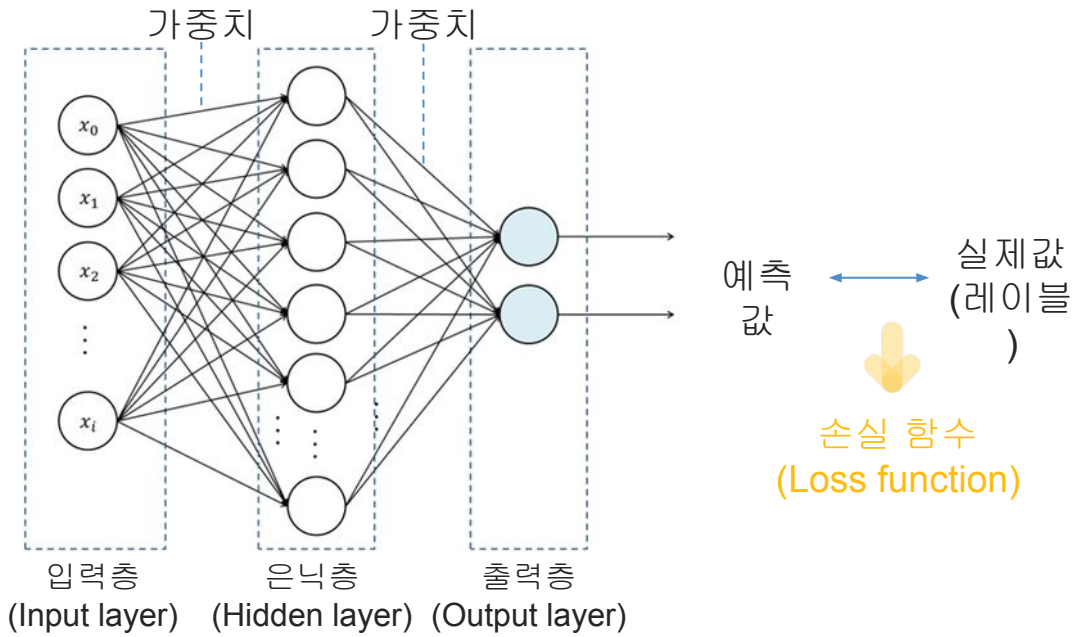
순전파 (Forward Propagation)

- 입력층에서 출력층 방향으로 예측값의 연산이 진행되는 과정



손실 함수 (Loss Function)

- 손실 함수의 값을 최소화하는 두 개의 매개변수인 가중치 **W**와 편향 **b**를 찾아가는 것이 딥 러닝의 학습 과정

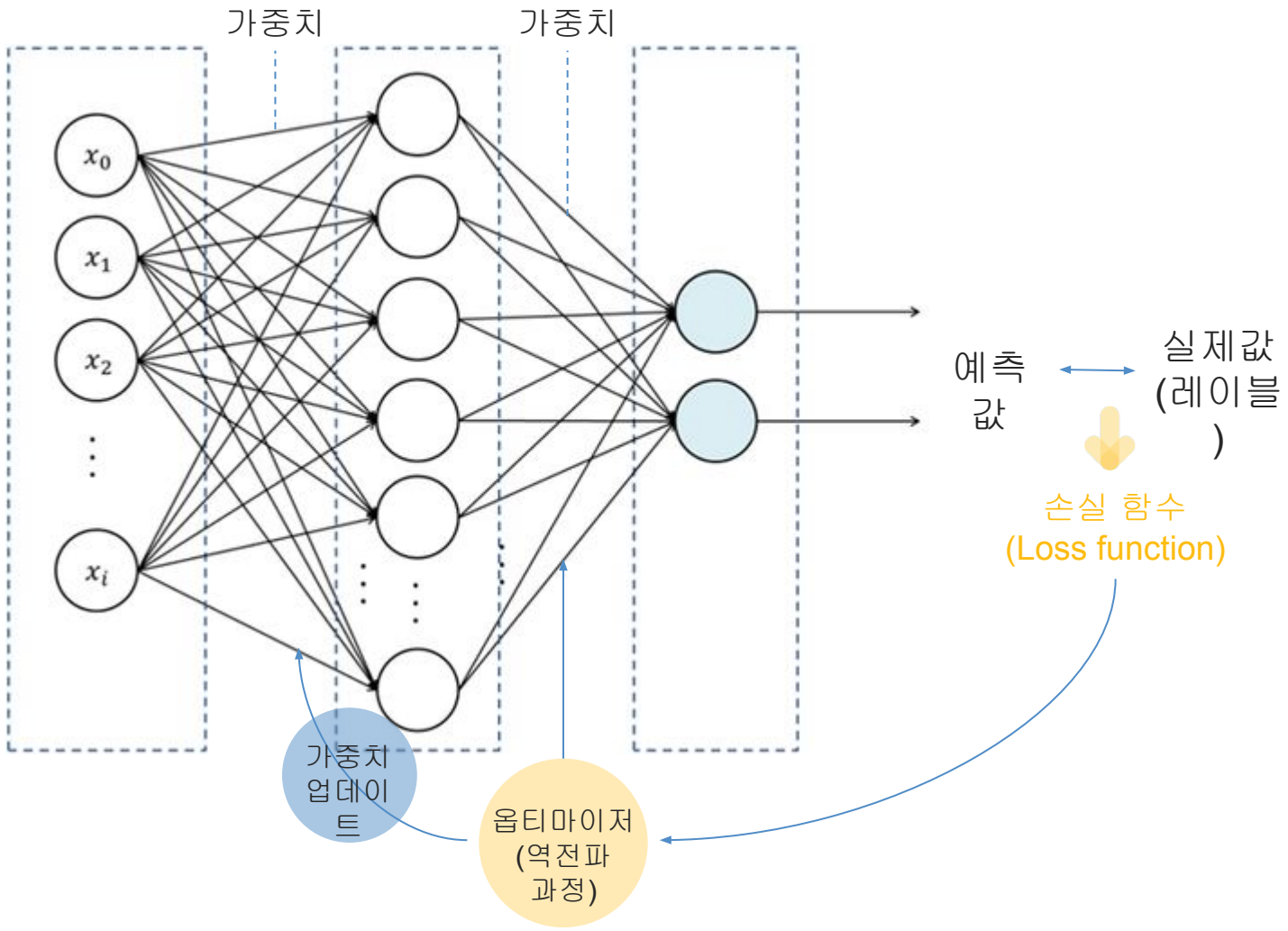


- 대표적

MSE (mean square error)	분류오차 (classfication error)	교차 엔트로피 (cross entropy)
회귀문제에서 주로 사용	이진 분류 문제에서 주로 사용	분류문제에서 주로 사용

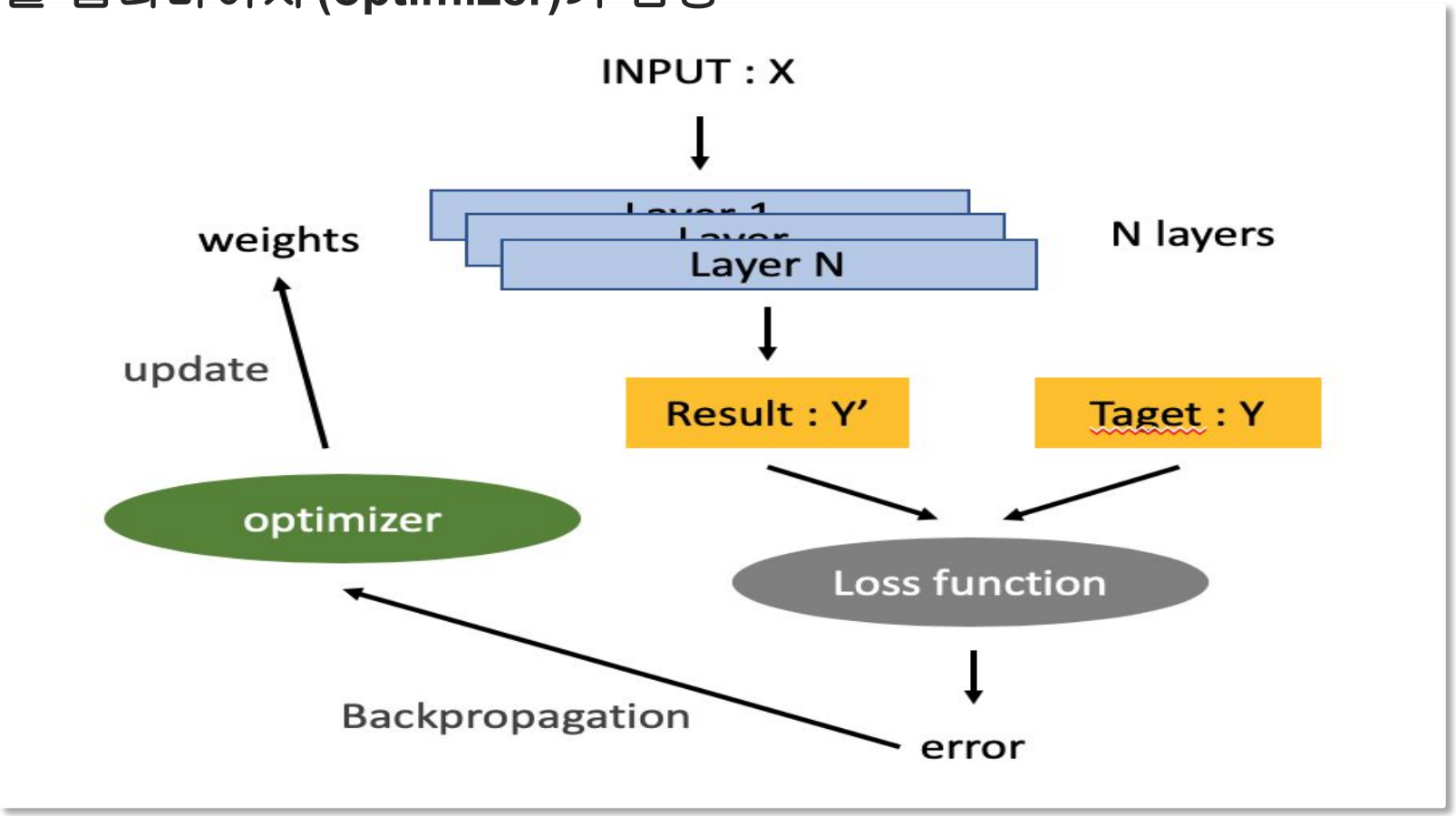
옵티마이저 (Optimizer)

- 손실 함수의 값을 줄여나가면서 학습하는 방법은 어떤 옵티마이저를 사용하느냐에 따라 달라짐



역전파 (Back Propagation)

- 손실함수의 결과를 개선하기 위해서 다시 결과에서부터 가중치를 수정하는 과정
- 이를 옵티마이저(**optimizer**)가 담당



기울기 소실 문제와 활성화 함수

활성화 함수

정의

- 입력값이 0차원이며, 출력값이 0차원인, 이전 층(layer) 들에서 출력된 결과값을 전체 층(layer)의 출력값 혹은 다음 층(layer)의 입력값 형태로 변환시켜주는 함수
- 스칼라값을 입력하면 스칼라값을 출력하는 함수로, 레이어 간에 입출력 형태를 맞춰주거나, 선형 함수로만 이루어진 신경망에서 비선형 함수인 활성화 함수를 사용하여, 비선형의 특성을 신경망에 지니게하는 역할을 가지고 있음

활성화 함수를 이용한 퍼셉트론 식

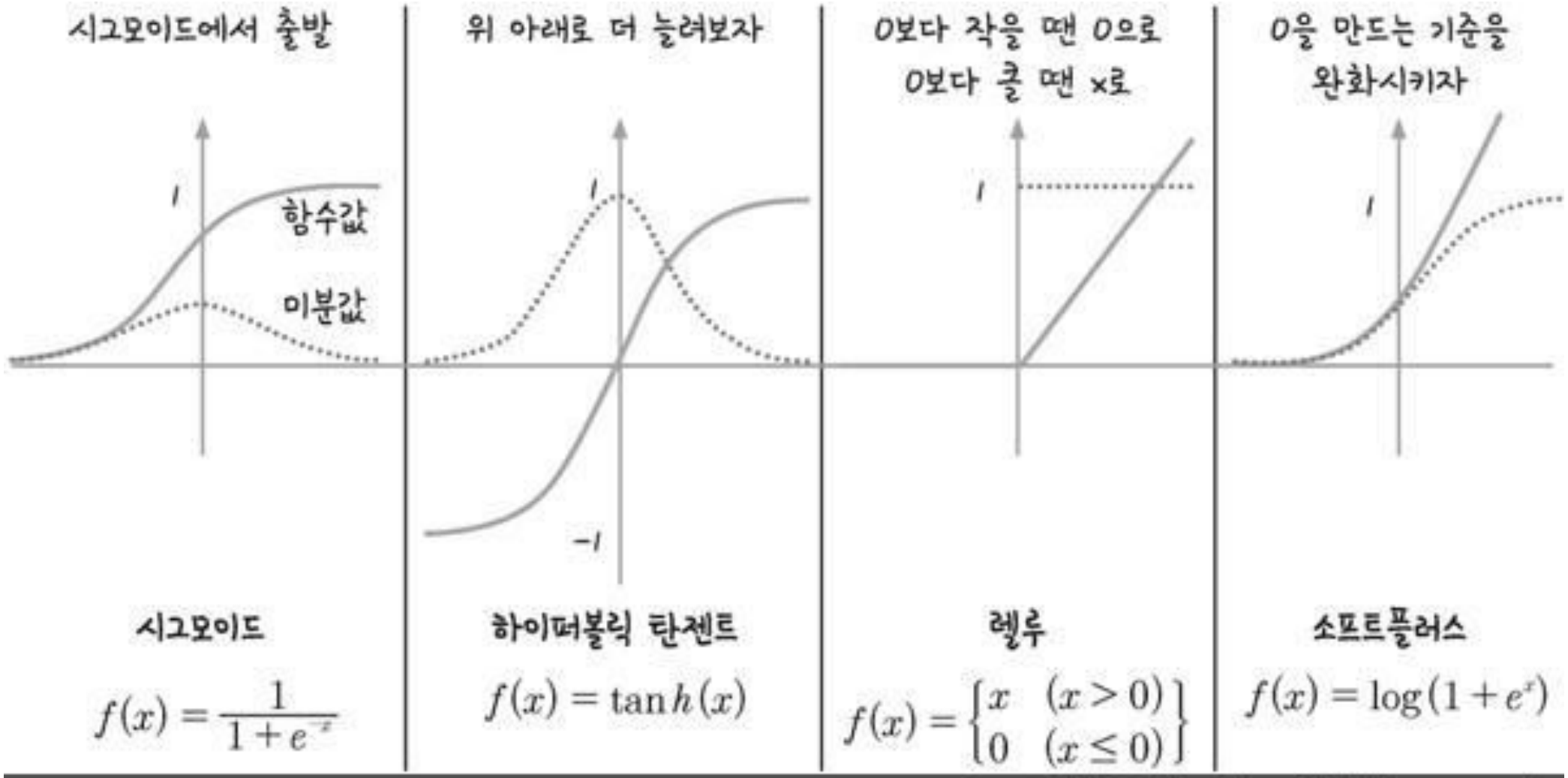
- $y = h(b + w_1 x_1 + w_2 x_2)$
- $$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$

※ x는 입력신호의 총합을 의미 함

대표적 활성화 함수

- 계단함수
- 시그모이드 함수
- ReLU 함수
- 항등함수
- 소프트맥스 함수

기울기 소실 문제와 활성화 함수



Tensorflow와 Keras

Tensorflow와 Keras의 역사

Francois Chollet

<구글 AI 개발자/연구원>

2015년 3월 27일 Keras 첫 공개 (백엔드 : Theano)

- Torch, Theano, Caffe와 같은 딥러닝 라이브러리가 있었지만, 어셈블리나 C++을 사용해야 했고, 시간이 많이 소요되고 비효율적.
- Keras는 사용하기도 쉽고, 작업 능률도 좋았음.

  그러나, 백엔드 (backend)가 필요함

2015년 11월 9일 TensorFlow 첫 공개

- 천천히, 조금씩 Keras는 백엔드로 TensorFlow를 지원하기 시작

2016년 9월 20일 Keras 1.1.0 공개

- 기본 백엔드가 TensorFlow로 변경됨

Tensorflow와 Keras의 역사

2018년 8월 9일 TensorFlow 1.10.0 공개

- 서브 모듈에 tf.keras가 포함됨. 즉, TensorFlow와 Keras와의 통합이 시작됨
- 이제 keras는 tf.keras는 별도의 패키지!

2019년 6월 구글이 TensorFlow 2.0 개발을 알림

- Keras가 TensorFlow의 공식 고수준 API라고 선언!

2019년 9월 18일 Keras 2.3.0 공개(Francois Chollet said!)

- Keras와 tf.keras가 이제 같음
- 이제 앞으로는 멀티 백엔드 지원 안함
- 앞으로 TensorFlow 2.0과 tf.keras를 사용
- Keras는 버그 정도만 수정할 예정

Tensorflow

- **tensorflow 프로그램 구조**
  - 그래프 생성
  - 그래프 실행

그래프

tensor들의 연산 모음

다차원 숫자 배열형의 데이터

0차원 텐서는 스칼라

1차원 텐서는 리스트(벡터)

2차원 텐서는 행렬

3차원 이상일 때는 다차원 행렬

지연실행(lazy evaluation)

- 함수형 프로그래밍에서 많이 사용
- tensor와 tensor의 연산들을 미리 정의하여 그래프를 만들고, 필요할 때 연산을 실행하는 코드를 넣어 원하는 시점에 실제 연산을 수행하도록 하는 방식

장점

모델 구성과 실행을 분리하여 프로그램을 깔끔하게 작성할 수 있다.

Tensorflow 2.x

TensorFlow 1.x

- 예전 코드가 되었음 (Legacy code)
- 구글은 보안과 관련된 업데이트만 제공할 예정
- 기본 데이터셋 역시 잘 제공하지 않음

TensorFlow 2.0

- 이전과 마찬가지로 구글이 만든 오픈 소스 프레임워크
- 머신러닝과 딥러닝을 위한 라이브러리
- Apache 라이선스
- TensorFlow 1.x에 존재한 중복된 API 제거, 빠른 프로토타입 및 쉬운 디버깅이 가능!
- 기본 실행 모드가 deferred execution이 아니라, eager execution임
- Session을 굳이 열어서 사용할 필요가 없음
- 핵심 기능의 일부로 Keras를 채택

Tensorflow 2.x

TensorFlow 2.0

- Keras를 위해 개발하고 최적화해서 당연히 Keras의 모든 기능을 사용 가능
- 딥러닝 모델 (CNN, RNN, LSTM 등)을 사용할 경우, **tf.keras** 네임스페이스를 사용
- 하위 호환성을 제공

  → tf.compat.v1 모듈 제공 (tf.contrib 미포함)

  → tf_upgrade_v2 라는 변환 스크립트를 제공

- tensorflow.js v1.0 을 제공
- 분산 데이터를 위한 TensorFlow Federated를 제공
- ragged tensors를 지원

  → tf.ragged.constant([[3, 1, 4, 1], [], [5, 9, 2], [6], []])

- 확률적 추론 및 통계분석을 위한 TensorFlow Probability를 제공

Tensorflow 2.x

TensorFlow 2.0

- @tf.function decorator를 지원
  → 파이썬 decorato
  → 그래프를 정의하고 session 실행을 함
  → TensorFlow 1.x의 tf.Session()과 유사

- AutoGraph를 지원
  → 파이썬 소스코드를 그래프로 변환
  → @tf.function 사용시 자동으로 적용

- Automatic differentiation and GradientTape

Tensorflow 1.x와 Tensorflow 2.x 비교

TensorFlow 1.x (legacy)

```
with tf.Session() as session:
session.run(tf.global_variables_init
ializer())
session.run(tf.tables_initializer())
model.fit(
  X_train,  y_train, validation_data
  a=(
       X_valid,
       y_valid
   ),
  epochs=10,  batch_size=64
 )
```

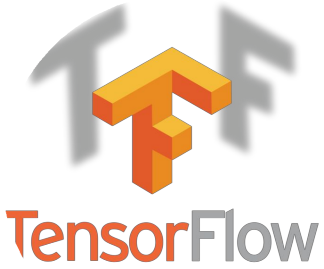
TensorFlow 2.0

```
model.fit(
X_train,  y_train,
validation_data=(
X_valid,
y_valid
),
epochs=10,  batch_size=64
)
```

Tensorflow Code와 Keras Code

TensorFlow

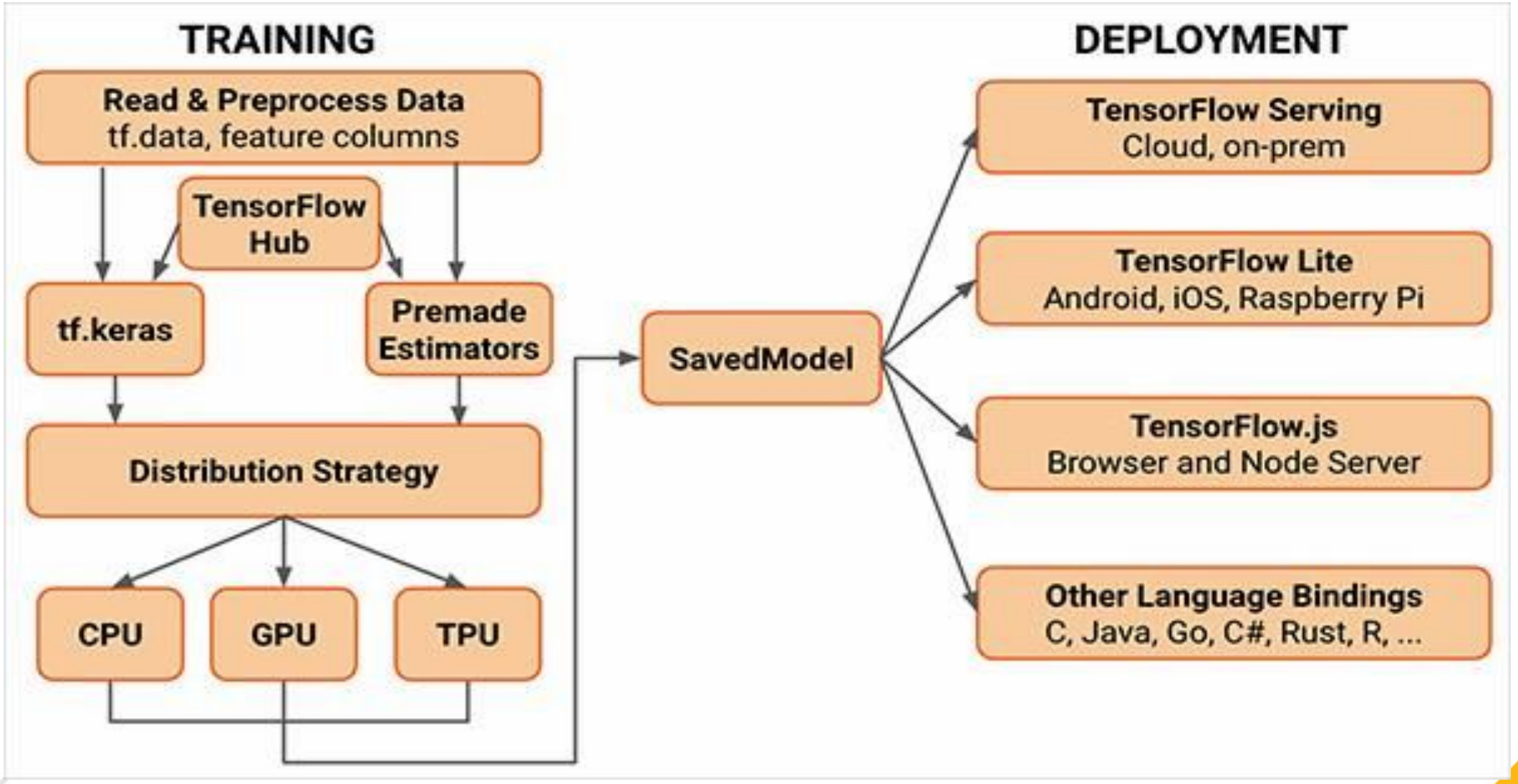
```
import tensorflow as tf

with tf.variable_scope('input'):
   X = tf.placeholder(tf.float32, shape=(None, 10), name="X")

with tf.variable_scope('layer_1'):
   weights = tf.get_variable("weights1", shape=[10, 50], initializer=tf.contrib.layers.xavier_initializer())
   biases = tf.get_variable(name="biases1", shape=[50], initializer=tf.zeros_initializer())
   layer_1_output = tf.nn.relu(tf.matmul(X, weights) + biases)

with tf.variable_scope('layer_2'):
   weights = tf.get_variable("weights2", shape=[50, 100], initializer=tf.contrib.layers.xavier_initializer())
   biases = tf.get_variable(name="biases2", shape=[100], initializer=tf.zeros_initializer())
   layer_2_output = tf.nn.relu(tf.matmul(layer_1_output, weights) + biases)

with tf.variable_scope('layer_3'):
   weights = tf.get_variable("weights3", shape=[100, 50], initializer=tf.contrib.layers.xavier_initializer())
   biases = tf.get_variable(name="biases3", shape=[50], initializer=tf.zeros_initializer())
   layer_3_output = tf.nn.relu(tf.matmul(layer_2_output, weights) + biases)

with tf.variable_scope('output'):
   weights = tf.get_variable("weights4", shape=[50, 1], initializer=tf.contrib.layers.xavier_initializer())
   biases = tf.get_variable(name="biases4", shape=[1], initializer=tf.zeros_initializer())
   prediction = tf.matmul(layer_3_output, weights) + biases

with tf.variable_scope('cost'):
   Y = tf.placeholder(tf.float32, shape=(None, 1), name="Y")
   cost = tf.reduce_mean(tf.squared_difference(prediction, Y))

with tf.variable_scope('train'):
   optimizer = tf.train.AdamOptimizer(0.05).minimize(cost)
```



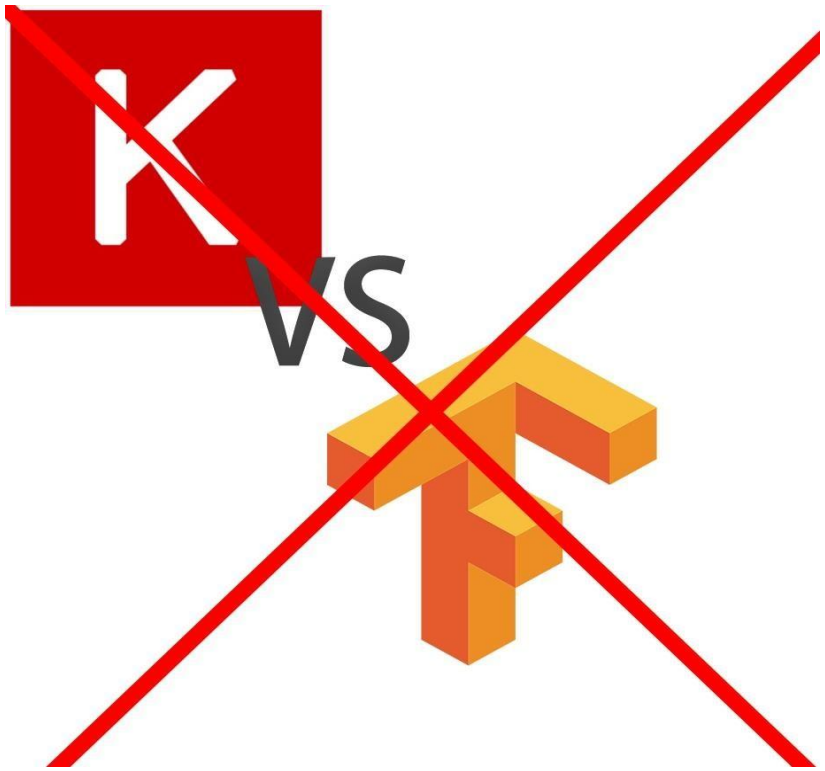
Keras

```
from keras.models import Sequential
from keras.layers import *

model = Sequential()
model.add(Dense(50, input_dim=10, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

Tensorflow 2.0 Eco System

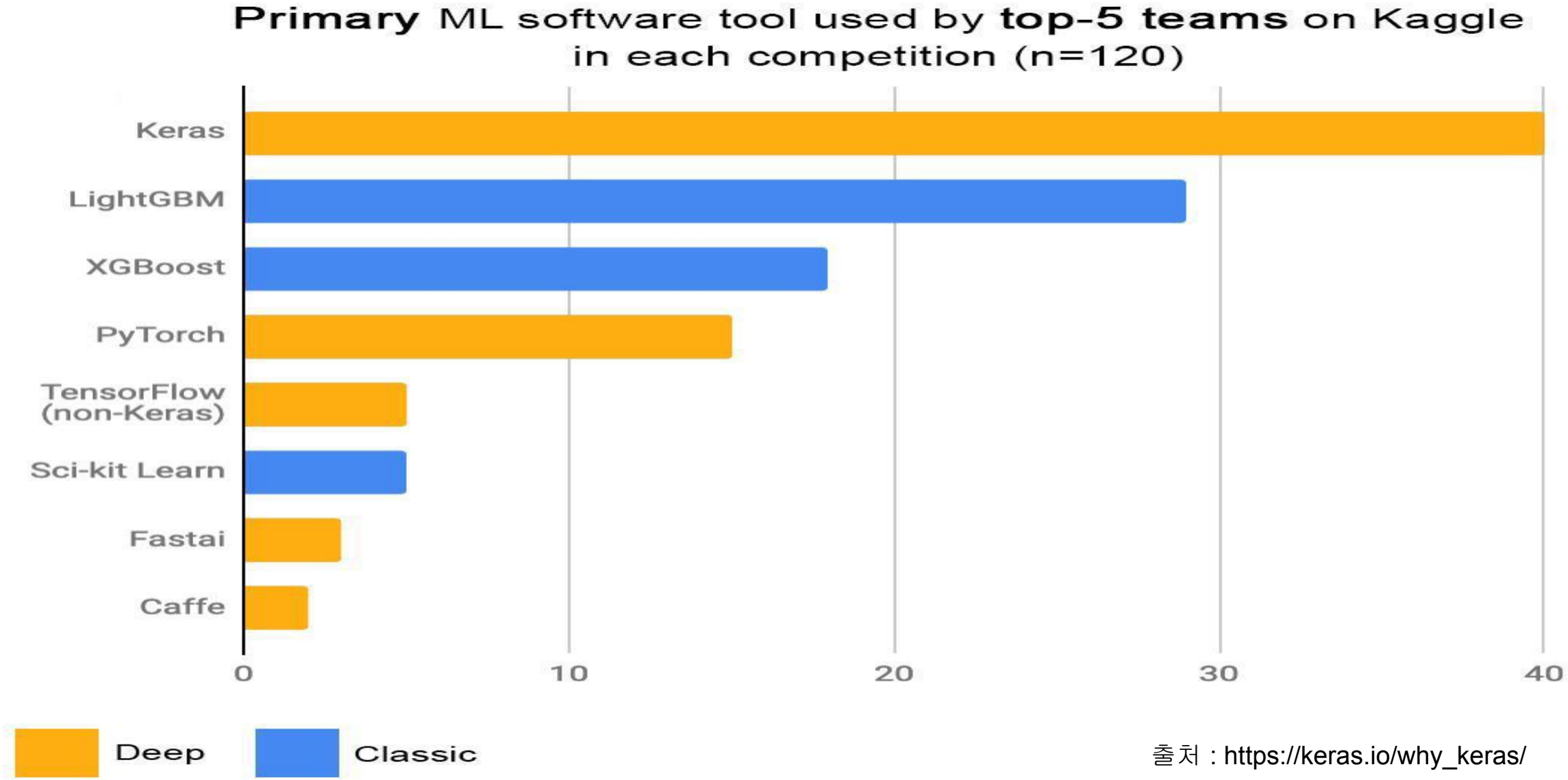


Tensorflow VS Keras



이제 Keras가 좋냐
TensorFlow가 좋냐라는
질문은 무의미해짐

Why choose Keras?



출처 : https://keras.io/why_keras/

Keras API

- Sequential Model
  - 아주 쉽습니다
  - 1개 입력과 1개 출력으로 계속 처리합니다
  - 70%의 경우에 적용이 가능합니다

```
from keras.models import Sequential
from keras.layers import Dense

model = keras.Sequential()
model.add(Dense(20, input_shape=(10, ), activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Keras API

- Functional API
  - 레고 블록하는 것과 같습니다
  - 다중 입력, 다중 출력 등으로 처리합니다
  - 95%의 경우에 적용이 가능합니다

```
from keras.models import Sequential
from keras.layers import Dense

input = keras.Input(shape=(10, ))
x = Dense(20, activation='relu')(x)
x = Dense(20, activation='relu')(x)
outputs = Dense(10, activation='softman')(x)

model = Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

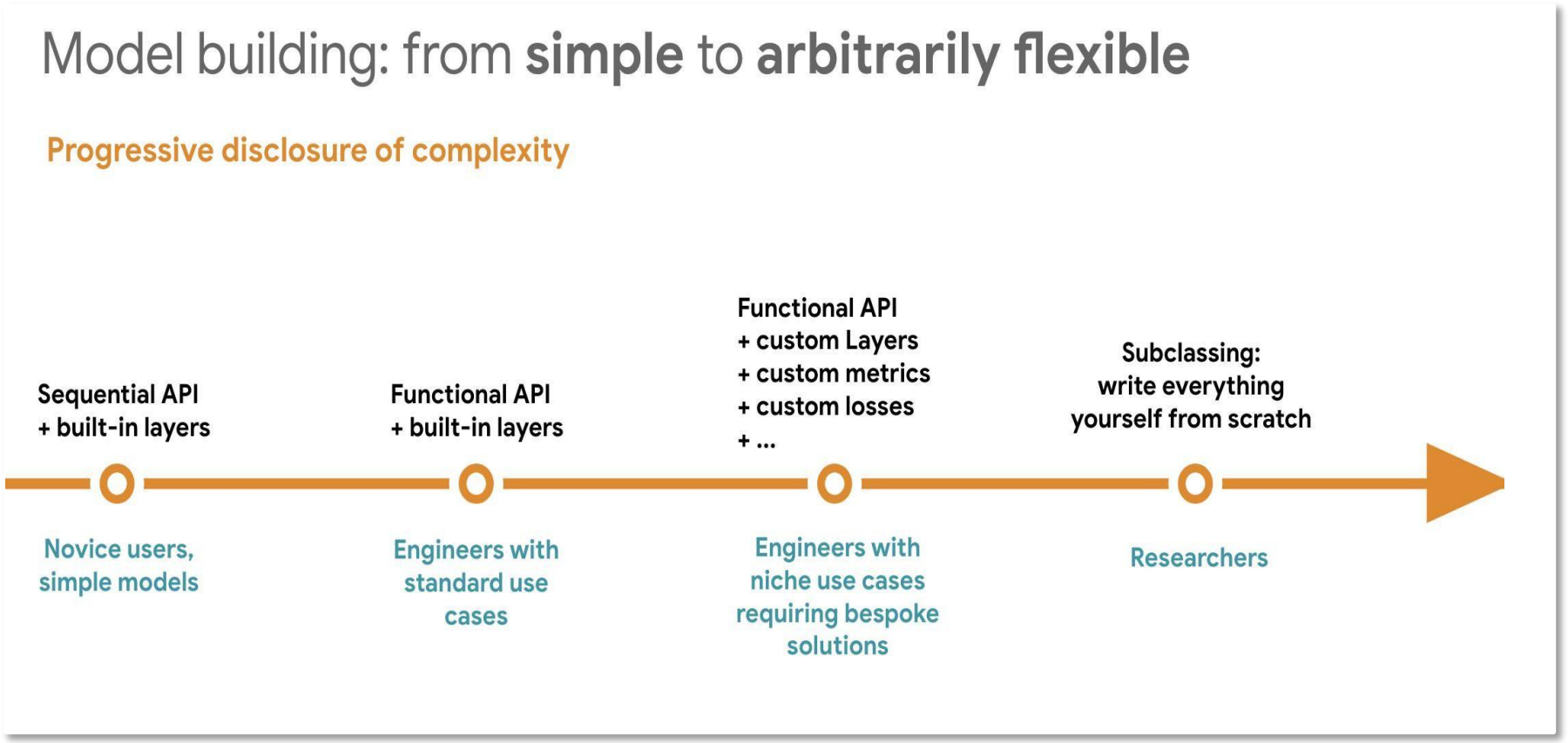
Keras API

- Model SubClass
  - 가장 유연하게 사용할 수 있습니다. 그러나, 어렵고 복잡합니다.
  - 잠재적으로 오류를 유발시킬 수 있습니다

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

Keras API

- 어떤 것을 사용하느냐?
  - 개발자 개인의 선택입니다!

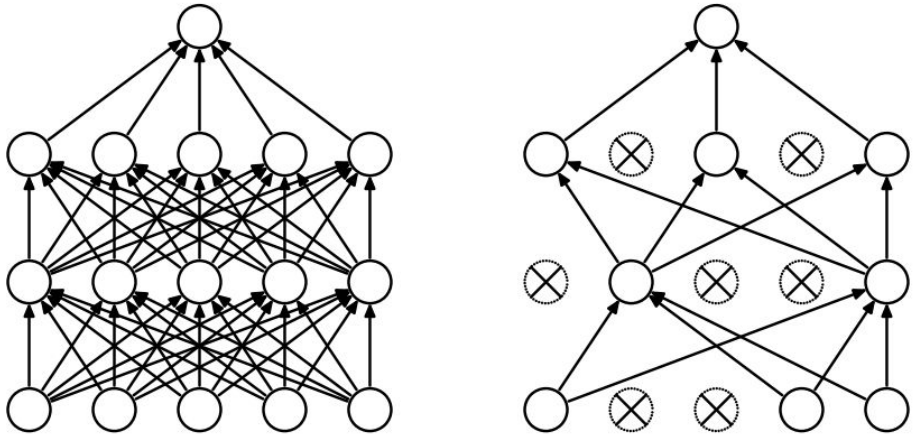


자주 사용하는 Keras 기능

Layer 객체

- **드롭아웃(Dropout) 레이어**

- 네트워크 속의 일부 유닛을 무효화 하는 기능을 제공
- 파라미터가 많고 표현력이 높은 네트워크의 자유도를 억제, 과적합 방지
- 테스트 데이터의 정확도를 향상시키기 위한 역할



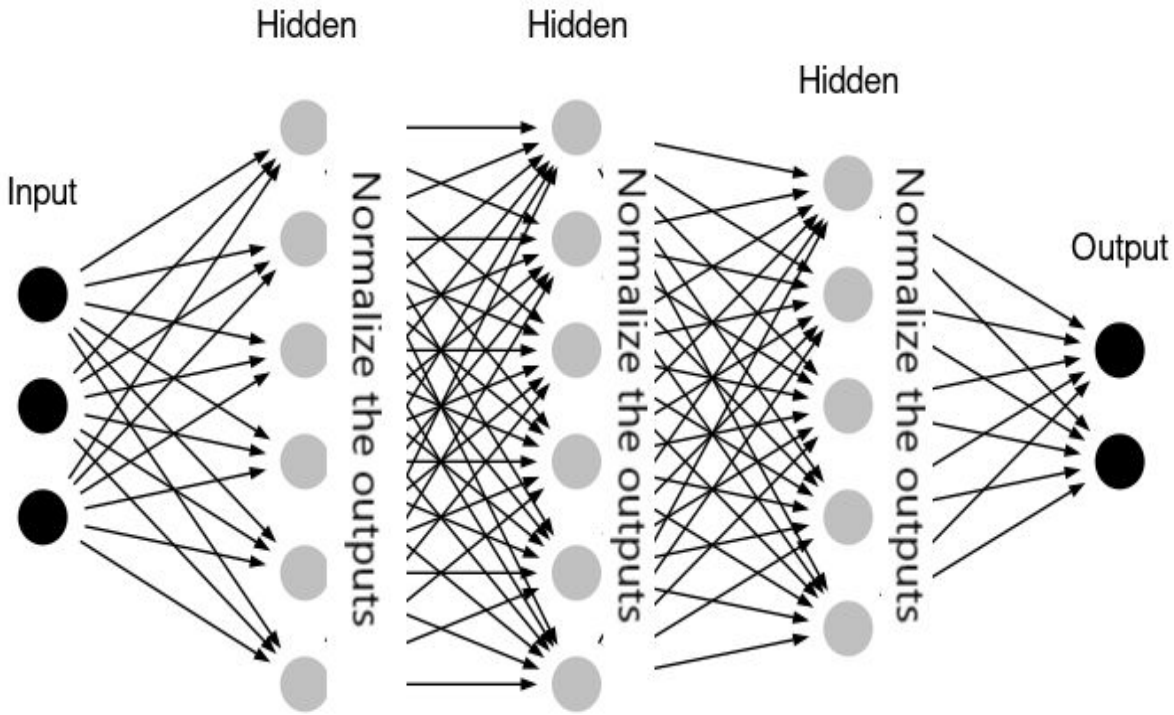
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Layer 객체

- **배치정규화 (BatchNormalization) 레이어**

- 배치마다 다음 레이어로의 입력값을 정규화.
- 각 레이어마다 정규화 하는 레이어를 두어, 변형된 분포가 나오지 않도록 조절하게 하는 것.
- 많은 경우 학습이 안정되고, 정확도도 향상됨.
- 학습이 안정되면 학습률을 크게 할 수 있기 때문에 학습 속도의 향상도 기대할 수 있음.
- Regularization 기능이 있기 때문에 드롭아웃과 같은 것들을 별도로 추가할 필요성이 작아짐.



Layer 객체

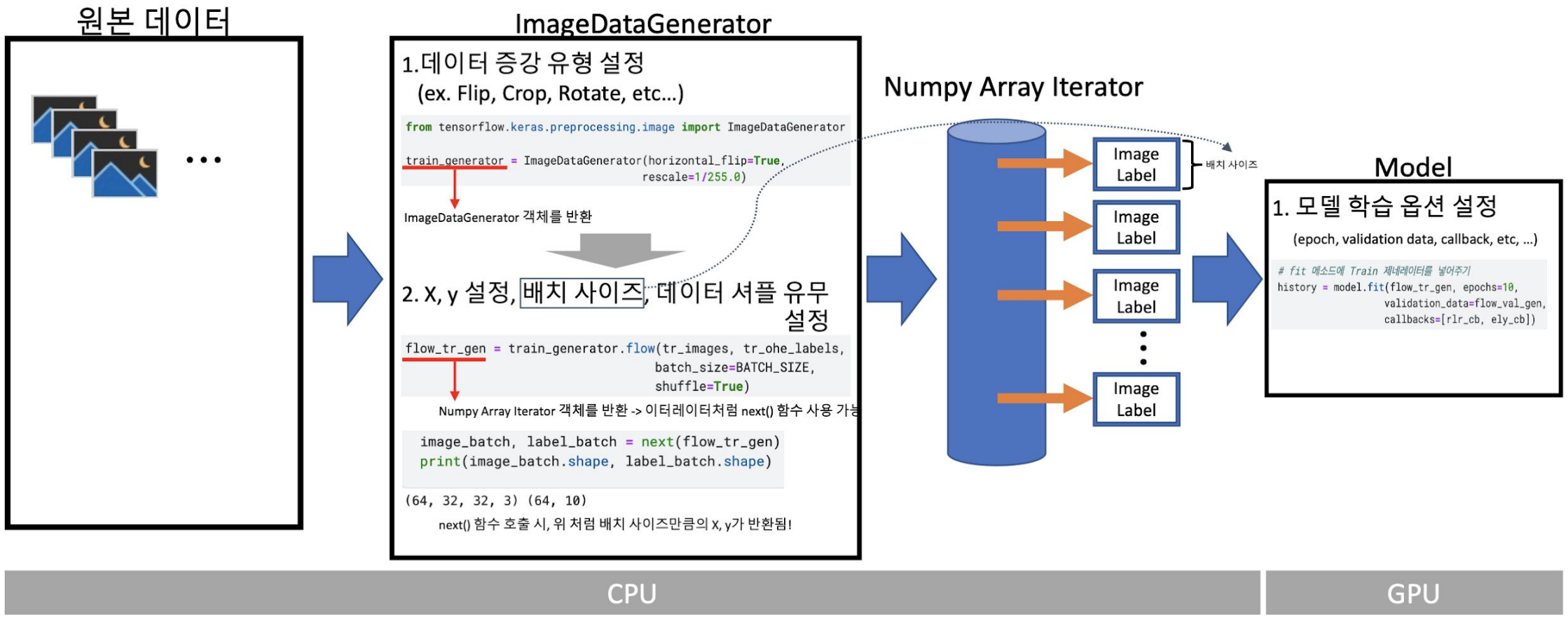
- **람다(Lambda) 레이어**

• 임의의 식과 함수를 케라스 레이어 객체로 네트워크에 추가 가능

데이터 증강 객체

- **ImageDataGenerator**

- Data Augmentation : 하나의 원본 이미지를 다양한 버전으로 만들어서 학습시키는 방법



출처 : https://blog.kakaocdn.net/dn/l2kMQ/btrf5jbWX5x/Xe2mFVJpLWZ9yOmd5Nfqck/img.png

ImageDataGenerator Options

featurewise_center	Boolean. Set input mean to 0 over the dataset, feature-wise.
samplewise_center	Boolean. Set each sample mean to 0.
featurewise_std_normalization	Boolean. Divide inputs by std of the dataset, feature-wise.
samplewise_std_normalization	Boolean. Divide each input by its std.
zca_epsilon	epsilon for ZCA whitening. Default is 1e-6.
zca_whitening	Boolean. Apply ZCA whitening.
rotation_range	Int. Degree range for random rotations.
width_shift_range	Float, 1-D array-like or int •float: fraction of total width, if < 1, or pixels if >= 1. •1-D array-like: random elements from the array. •int: integer number of pixels from interval (-width_shift_range, +width_shift_range) - With width_shift_range=2 possible values are integers [-1, 0, +1], same as with width_shift_range=[-1, 0, +1], while with width_shift_range=1.0 possible values are floats in the interval [-1.0, +1.0).

ImageDataGenerator Options

height_shift_range	Float, 1-D array-like or int • float: fraction of total height, if < 1, or pixels if >= 1. • 1-D array-like: random elements from the array. • int: integer number of pixels from interval (-height_shift_range, +height_shift_range) - With height_shift_range=2 possible values are integers [-1, 0, +1], same as with height_shift_range=[-1, 0, +1], while with height_shift_range=1.0 possible values are floats in the interval [-1.0, +1.0).
brightness_range	Tuple or list of two floats. Range for picking a brightness shift value from.
shear_range	Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)
zoom_range	Float or [lower, upper]. Range for random zoom. If a float, [lower, upper] = [1-zoom_range, 1+zoom_range].
channel_shift_range	Float. Range for random channel shifts.
fill_mode	One of {"constant", "nearest", "reflect" or "wrap"}. Default is 'nearest'. Points outside the boundaries of the input are filled according to the given mode: • 'constant': kkkkkkkk|abcd|kkkkkkkk (cval=k) • 'nearest': aaaaaaaa|abcd|dddddddd • 'reflect': abcddcba|abcd|dcbaabcd • 'wrap': abcdabcd|abcd|abcdabcd

ImageDataGenerator Options

cval	Float or Int. Value used for points outside the boundaries when fill_mode = "constant".
horizontal_flip	Boolean. Randomly flip inputs horizontally.
vertical_flip	Boolean. Randomly flip inputs vertically.
rescale	rescaling factor. Defaults to None. If None or 0, no rescaling is applied, otherwise we multiply the data by the value provided (after applying all other transformations).
preprocessing_function	function that will be applied on each input. The function will run after the image is resized and augmented. The function should take one argument: one image (Numpy tensor with rank 3), and should output a Numpy tensor with the same shape.
data_format	Image data format, either "channels_first" or "channels_last". "channels_last" mode means that the images should have shape (samples, height, width, channels), "channels_first" mode means that the images should have shape (samples, channels, height,width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
validation_split	Float. Fraction of images reserved for validation (strictly between 0 and 1).
dtype	Dtype to use for the generated arrays.

신경망을 위한 요소들

신경망을 위한 데이터 전처리

- **목적**
  - 주어진 원본 데이터를 신경망에 적용하기 쉽도록 만드는 것

벡터화

신경망에서 모든 입력과 타깃은 부동소수데이터로 이루어진 텐서이어야 함 (특정 경우에는 정수도 가능)

data vectorization

값 정규화

일반적으로 비교적 큰 값이나, 균일하지 않은 데이터를 네트워크에 주입하는 것은 지양

누락된 값 다루기

Missing completely at random (MCAR)
Missing at random (MAR)
Not missing at random (NMAR)

Overfitting / Underfitting

- **머신러닝의 근본적인 작업은 최적화와 일반화 사이의 줄다리기**

최적화

가능한 훈련 데이터에서 최고의 성능을 얻기 위해 모델을 조정하는 과정 ➡ 학습

훈련된 모델이 이전에 본 적이 없는 데이터에서 얼마나 잘 수행되는가?

일반화

Overfitting / Underfitting

과대 적합 (Overfitting)

훈련 데이터에서 여러 번 학습하고 나면 어느 시점부터 일반화 성능이 더 이상 좋아지지 않고 검증 세트의 성능이 멈추고 감소하기 시작하는 형태

훈련 데이터의 손실이 낮아질 수록 데이터의 손실도 낮아지는 형태. 즉 네트워크가 훈련 데이터에 있는 관련 특성을 모두 학습하지 못함.

과소 적합 (Underfitting)

과적합을 막는 방법들

1. 데이터의 양을 늘리기
2. 모델의 복잡도 줄이기
3. 가중치 규제(Regularization) 적용하기
4. **드롭아웃(dropout)**

Epoch and Batch size and Iteration

- **같은 문제지와 정답지를 주더라도 기계마다 공부 방법을 다르게 설정할 수 있음**

에포크(Epoch)

- 인공 신경망에서 전체 데이터에 대해서 순전파와 역전파가 끝난 상태
- 공부를 한 번 끝낸 상태

배치 크기(Batch size)

- 몇 개의 데이터 단위로 매개변수를 업데이트 하는지

이터레이션(Iteration)

- 한 번의 에포크를 끝내기 위해서 필요한 배치의 수
- 한 번의 에포크 내에서 이루어지는 매개변수의 업데이트 횟수