10장

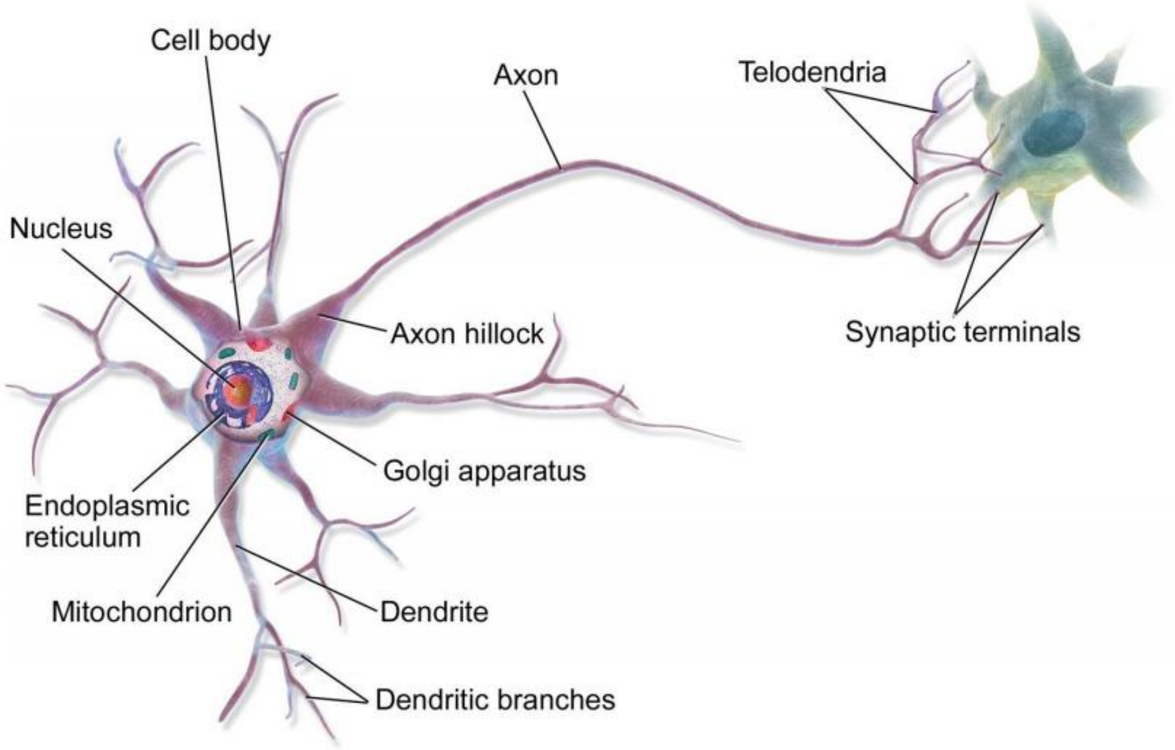
인공신경망 소개

(Introduction to Artificial Neural Networks)

인공신경망(Artificial Neural Network)

- 뇌의 구조에서 영감을 받아 만든, 인공지능을 위한 계산 모델
- 뇌의 구조를 그대로 흉내 내려는 것은 아님.
- 뉴런(신경세포)이 서로 연결된 네트워크
- 딥러닝으로 발전
  - 구글 이미지 : 이미지 분류
  - 애플 시리 : 음성인식
  - 유투브 : 비디오 추천
  - 알파고 : 바둑기보 학습

생물학적 뉴런(Neuron. 신경세포)



- 세포체(cell body) : 세포핵(Neucleus)를 포함. 정보 처리
- 수상돌기(dendrite) : 다른 셀에서 정보를 받음
- 축색돌기(axon) : 다른 셀(들)에게 정보를 보냄
- 시냅스(synapse. synaptic terminal) : 하나의 셀과 다른 셀이 연결되는 곳. (축색돌기의 끝과 수상돌기 가 연결됨)

- 동작
  - 시냅스를 통해 다른 뉴론으로 부터 입력 신호를 받음
  - 입력 신호가 임계치를 넘으면 세포가 활성화 됨
  - 시냅스마다 시냅스효능(synaptic efficacy)이 다름. 즉, 입력을 받을 때 어떤 뉴론의 신호는 크게, 어떤 뉴론의 크기는 작게 반영함
  - 시냅스효능이 바꿈으로 학습이 일어남

생물학적 뉴런(Neuron. 신경세포)

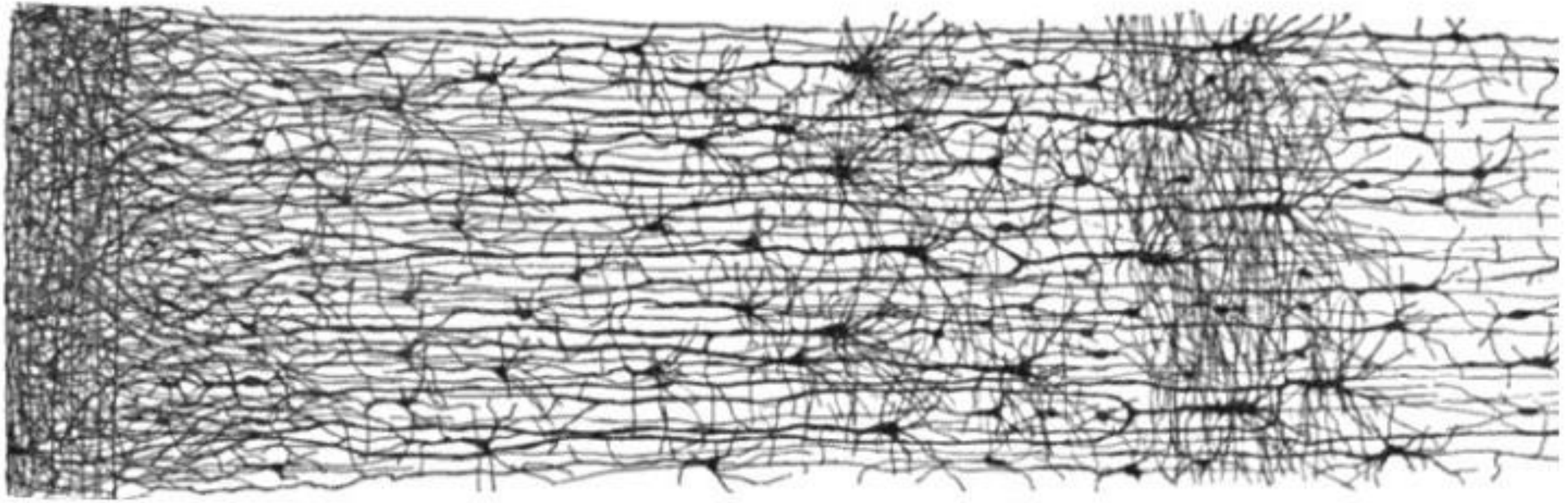


Figure 10-2. Multiple layers in a biological neural network (human cortex)[5]

뉴런을 사용한 논리 연산

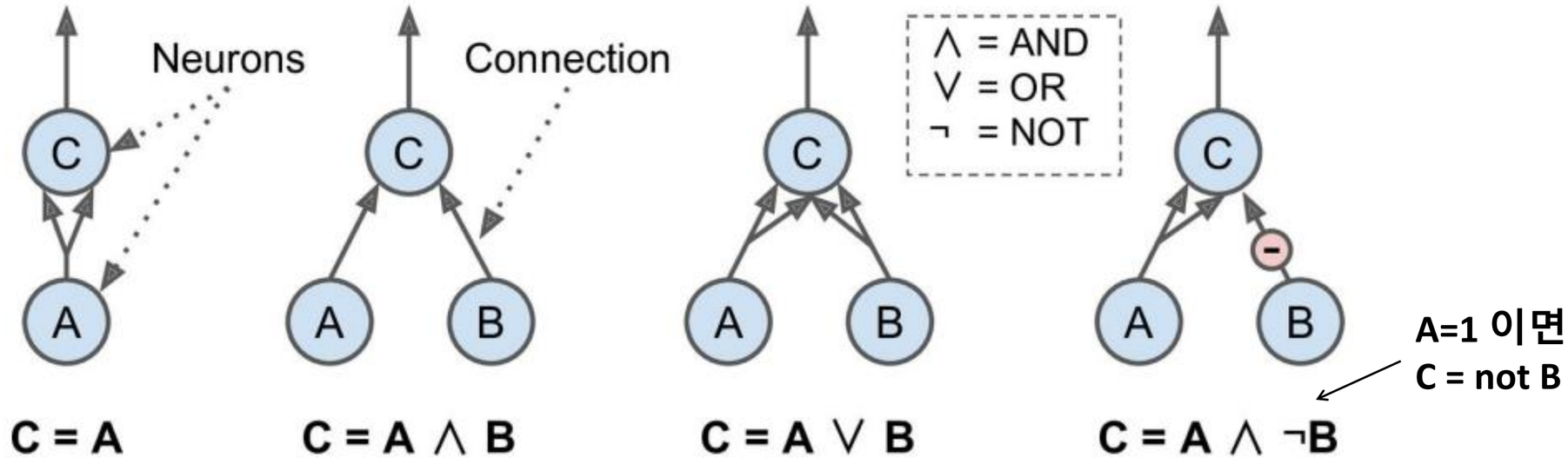


Figure 10-3. ANNs performing simple logical computations

- 1943년 맥클록(McClloch)과 피트(Pitts)가 제안 : AND, OR, NOT을 뉴런으로 계산할 수 있다
  ➔ 인공뉴런(Artificial Neuron)의 시초
  - 뉴런은 입력이 2 이상이 되면 활성화(출력이 1 됨) 된다고 가정
  - 가지 하나는 가중치 1 이라고 가정

퍼셉트론(Perceptron)

- 학습이 되는 간단한 인공신경망
- 신경세포 ➔ TLU(Threshold Logic Unit)
- Rosenblatt 1957

TLU

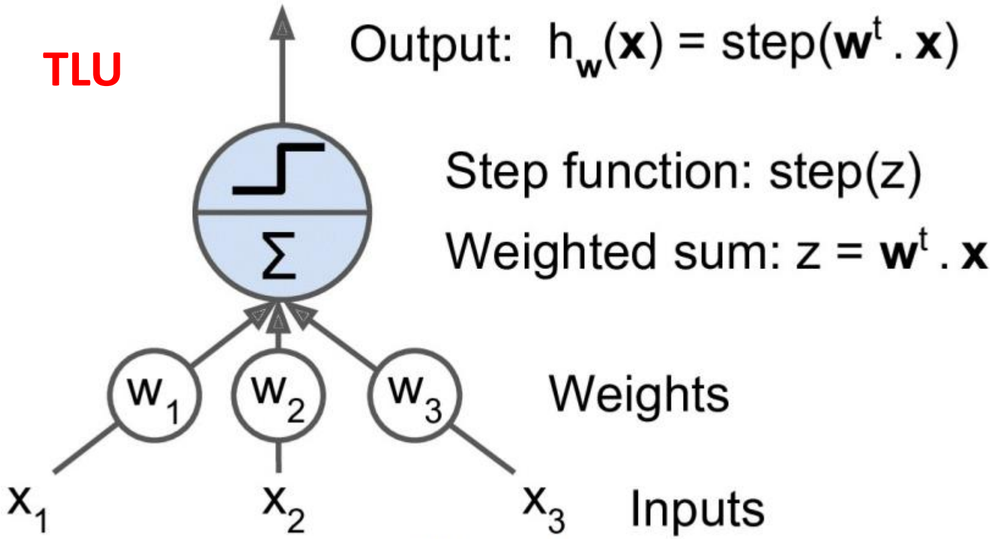


Figure 10-4. Linear threshold unit

Peceptron

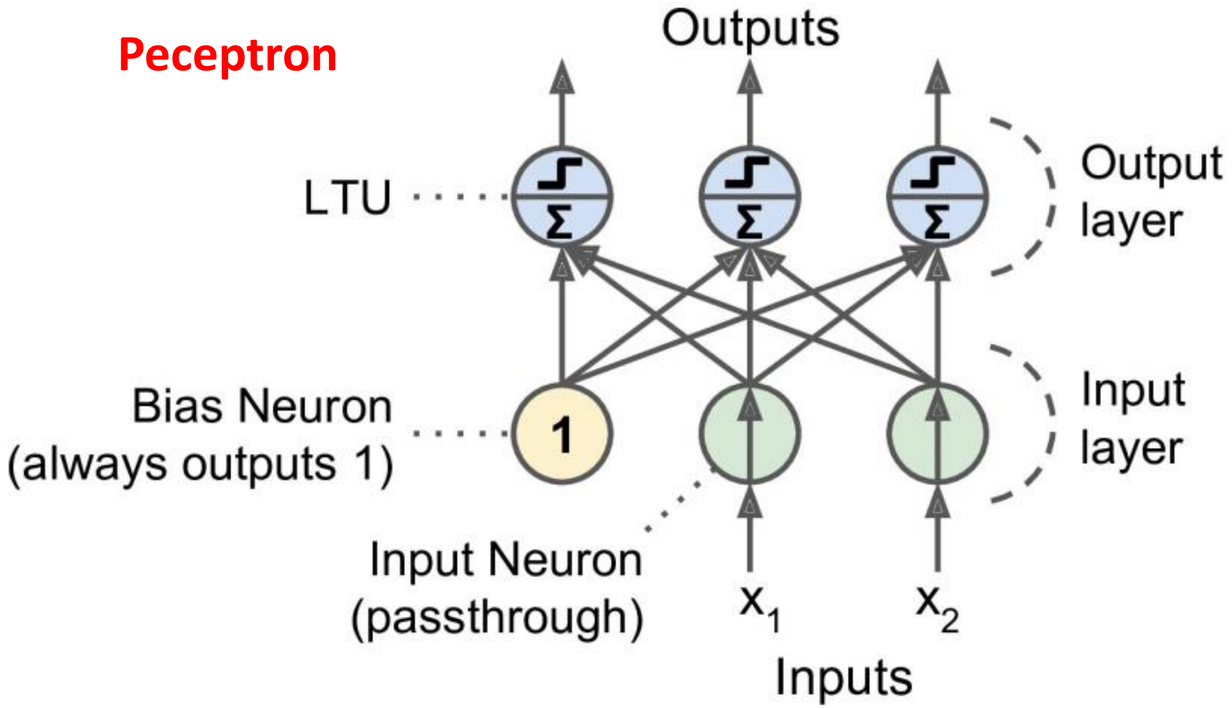


Figure 10-5. Perceptron diagram

Step functions (z : threshold)

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Peceptron Learning Rule

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(\hat{y}_j - y_j)x_i$$

퍼셉트론(Perceptron) : 프로그램

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)]  # petal length, petal width
y = (iris.target == 0).astype(np.int)  # Iris Setosa?

per_clf = Perceptron(random_state=42)
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

XOR 문제 와 다층퍼셉트론(MLP)

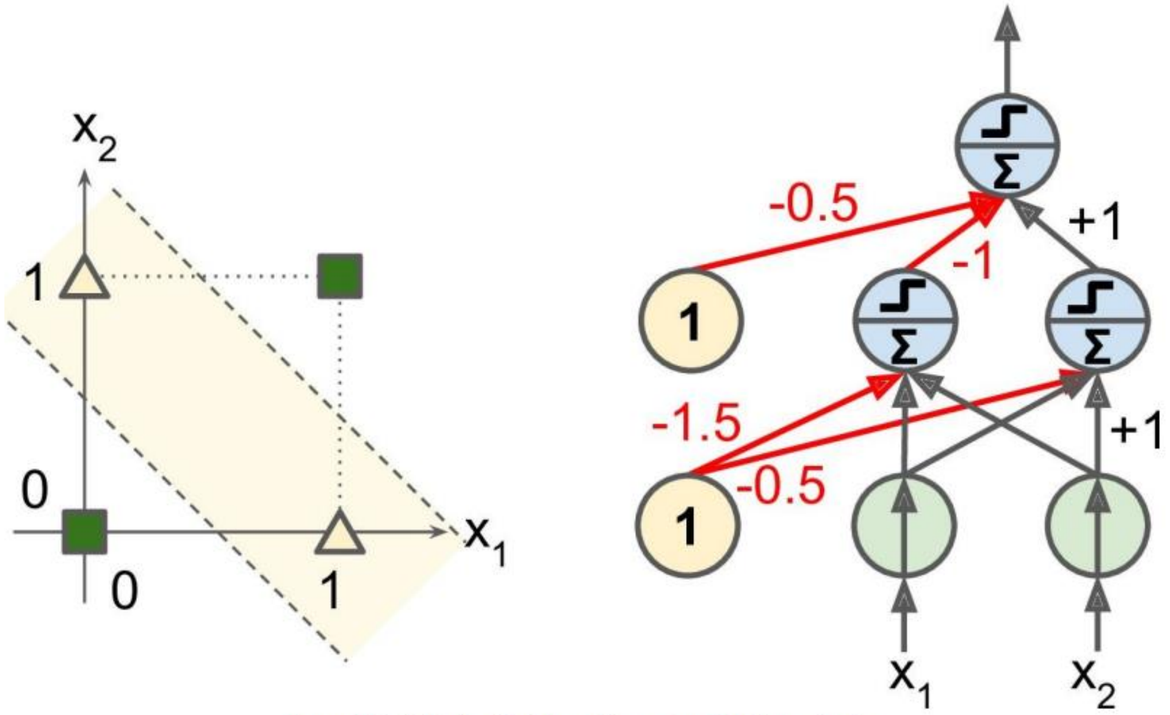


Figure 10-6. XOR classification problem and an MLP that solves it

다층퍼셉트론 (MultiLayerPerceptron)
XOR 문제 풀수 있음 : 1980년대

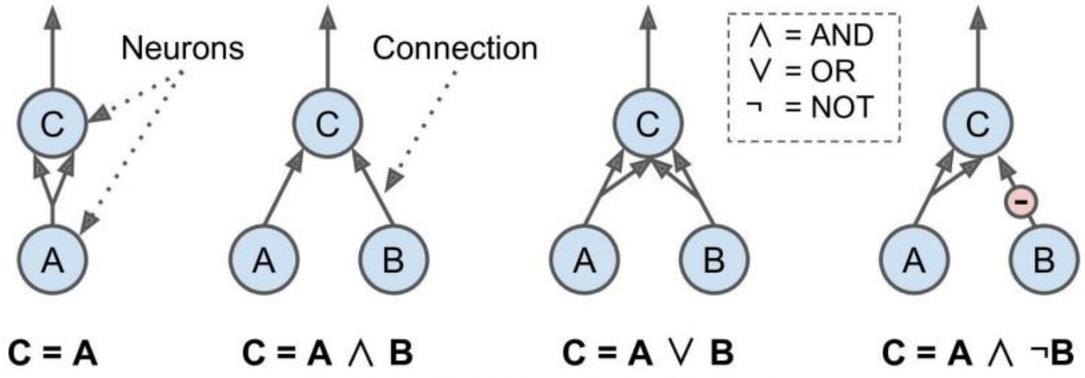


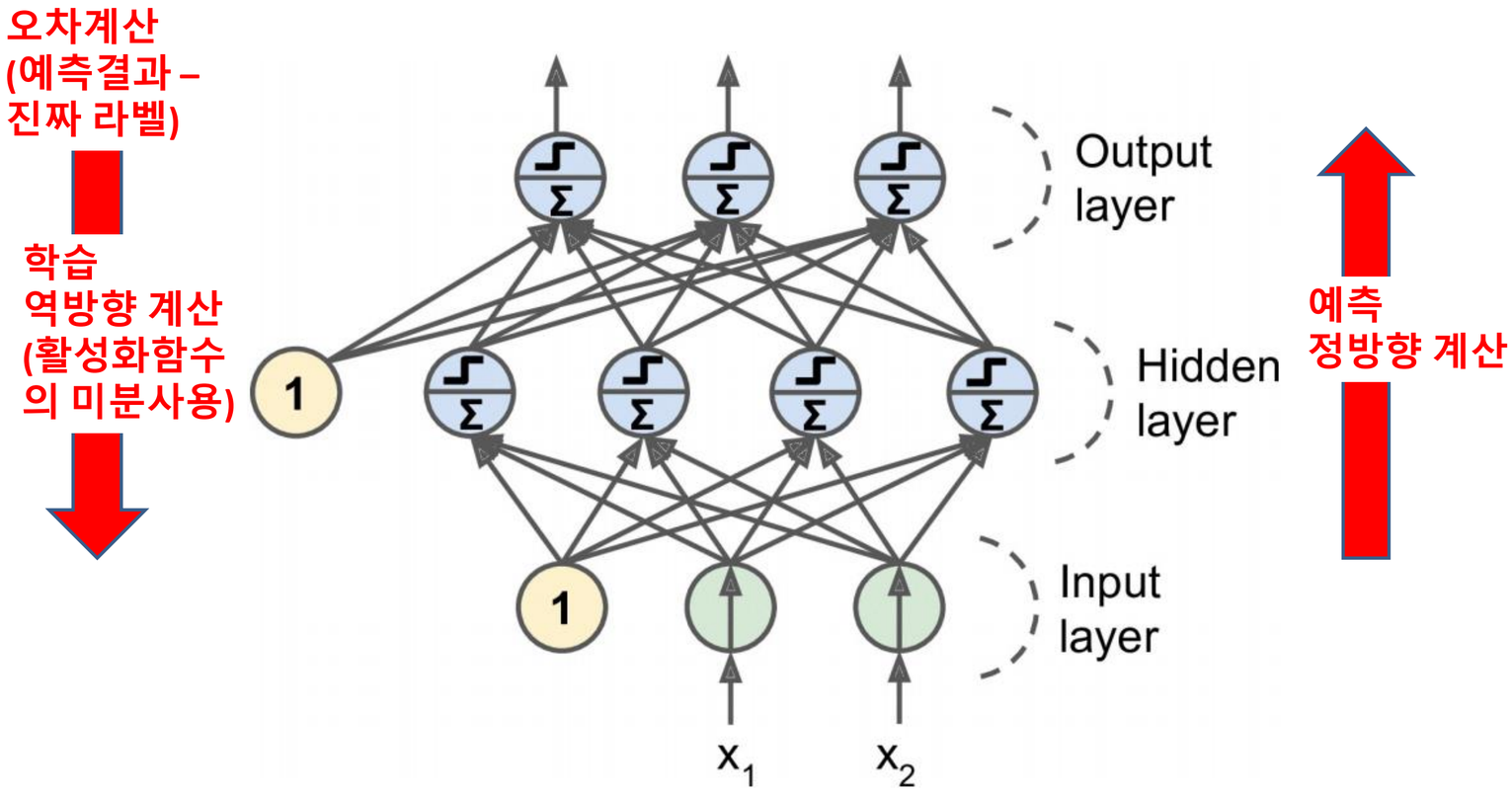
Figure 10-3. ANNs performing simple logical computations

(단층)퍼셉트론
XOR 문제 풀수 없음 : 1969년 Minsky와 Papert가 지적

다층퍼셉트론과 오류역전파(backpropagation)

1986년 다층퍼셉트론 (MultiLayerPerceptron) : 인공신경망 분야 활성화 기여

- 구조 : 단층(sigle layer) ➔ 다층(multi layer)
- 활성화 함수 : step function(미분불가능) ➔ logistic fuction등 (미분가능)
- 학습규칙 : 퍼셉트론 ➔ 오류역전파
- 성능 : 선형으로 분리 가능한 문제(분포)만 해결 ➔ 어떤한 문제도 풀 수 있음 (XOR 문제 포함)



다층퍼셉트론 : 활성화 함수

활성화 함수

- lostic $\sigma(z) = 1 / (1 + \exp(-z))$
- hyperbolic tangent $tanh\,(z) = 2\sigma(2z) - 1$
- ReLU (Rectified Linear Unit) $\mathrm{ReLU}\,(z) = \max\,(0, z)$

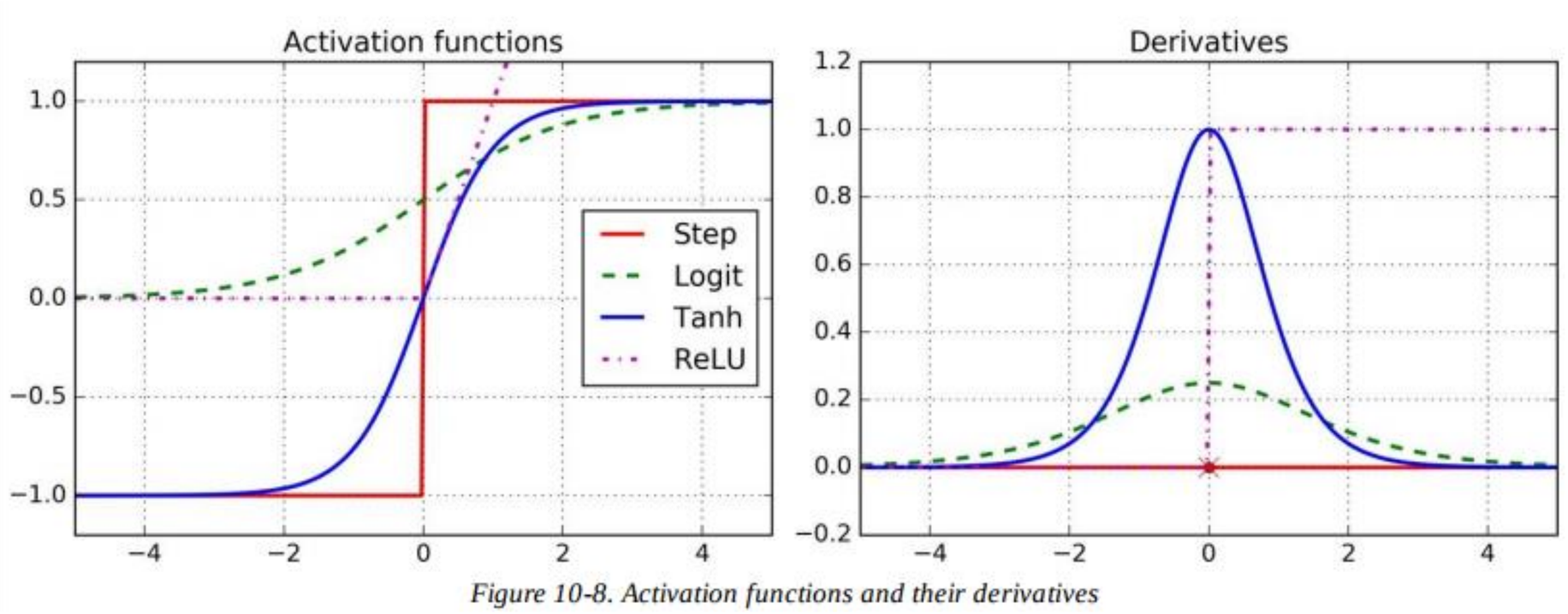


Figure 10-8. Activation functions and their derivatives

다층퍼셉트론(MLP, FNN) : 요즘 일반적 형태

최근 다층퍼셉트론

- FNN(Feed Forward Neural Network) 이라고도 함
- 활성화 함수 : ReLU
- 출력 인코딩 : 원핫 인코딩
- 출력층 활성화 함수 : 소프트맥스

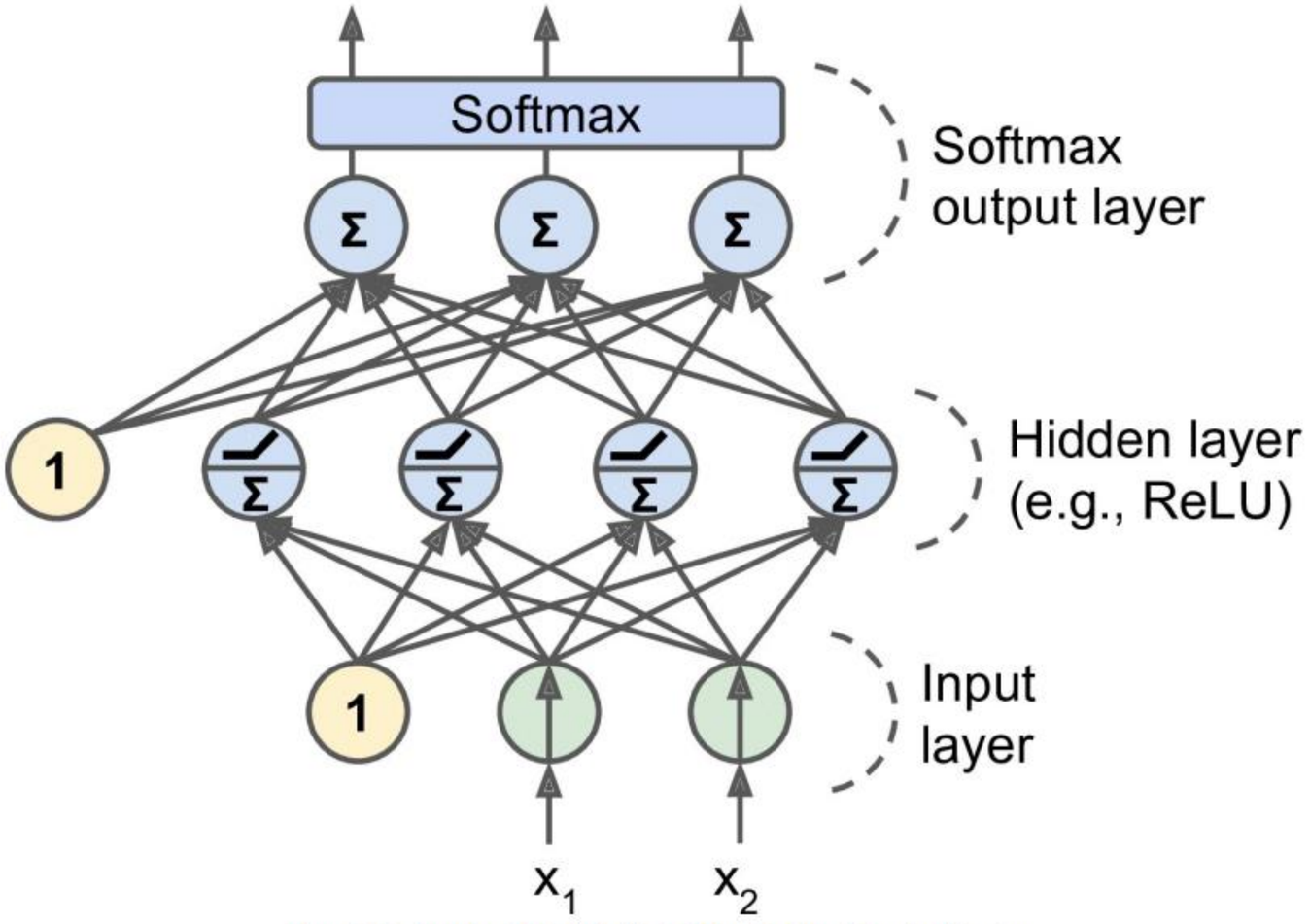


Figure 10-9. A modern MLP (including ReLU and softmax) for classification

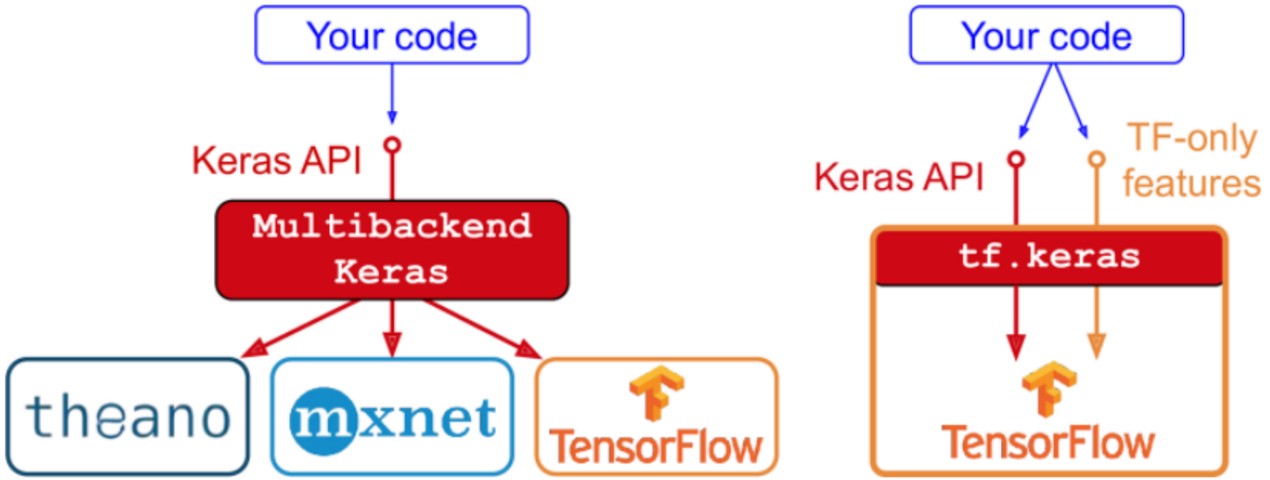
Keras

Keras :

- 2015. François Chollet 발표
- 체계적, 사용 편함, Backend 위에서 동작 : Tensorflow, CNTK, Theano
- 3가지 형태 프로그래밍 지원 : Sequential, Functional, Subclassing API

Tensorflow :

- 2.0부터 Keras 형태로 바뀜.
- tf.keras : Keras 지원 + tensorflow 고유함수 이용 가능 (data load, preprocessing등)



Sequential API 사용 예

다층퍼셉트론 : Fashion MNIST

Fashion MNIST. 28*28 uinit8 images. Total 70,000 ➔ Training 60,000 Test 10,000

```
import tensorflow as tf
from tensorflow import keras

fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] / 255.0
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])

model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd",
              metrics=["accuracy"])

model.fit(X_train, y_train, epochs=30, batch_size=32,
          validation_data=(X_valid, y_valid))
model.save("my_keras_model.h5")
    # model = keras.models.load_model("my_keras_model.h5")
```

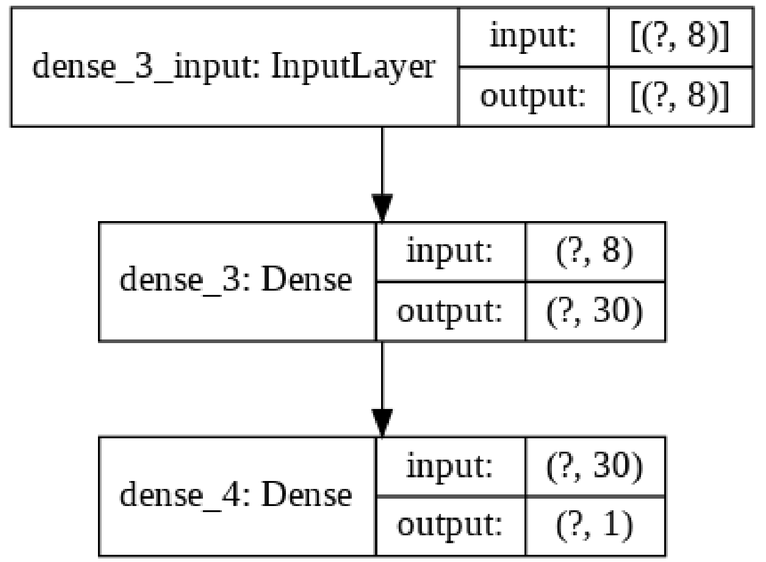
- Label이 0부터 9까지 숫자로 되어 있는 경우 : sparse_categorical_crossenstropy
- Label이 one-hot-encoding되어 10개의 0 혹은 1 숫자로 되어 있는 경우 : categorical_crossentropy
- 출력 노드가 1개인 경우 : binary_crossentropy

없어도 됨. Default=32

```
model.evaluate(X_test, y_test)
➔ 10000/10000 [==========] - 0s 29us/sample - loss: 0.3340 - accuracy: 0.8851

X_new = X_test[:3]
y_proba = model.predict(X_new)
y_proba.round(2)
➔ array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.03, 0.  , 0.01, 0.  , 0.96],
         [0.  , 0.  , 0.98, 0.  , 0.02, 0.  , 0.  , 0.  , 0.  , 0.  ],
         [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]], dtype=float32)
```

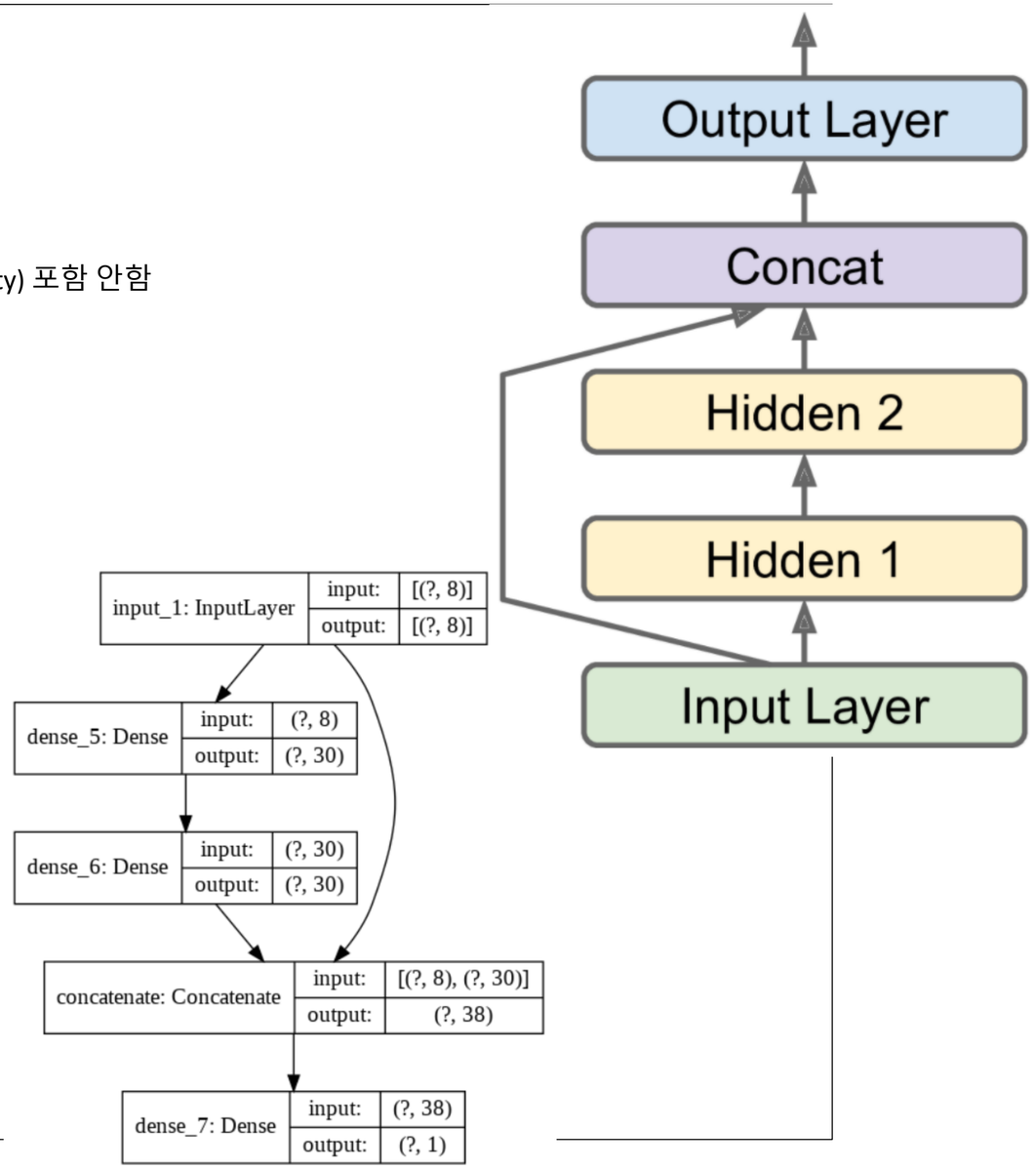
Sequential API 사용 예 다층퍼셉트론 : California Housing

```
import tensorflow as tf
from tensorflow import kearas
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()  # 입력 특징에 숫자형만 포함된 데이터.  범주형(Ocean Proximity) 포함 안함
X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data, housing.target)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full)

scaler = StandardScaler()                    # 입력 특징 scaling
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)

Model = keras.models.Sequential([    # 모델 : 히든레이어 1개
    keras.layers.Dense(30, activation="relu", input_shape=X_train.shape[1:]),
    keras.layers.Dense(1)
])
model.compile(loss="mean_squared_error", optimizer="sgd")
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
mse_test = model.evaluate(X_test, y_test)
X_new = X_test[:3] # pretend these are new instances
y_pred = model.predict(X_new)
```



Functional API 사용 예 Wide%Deep Net : California Housing

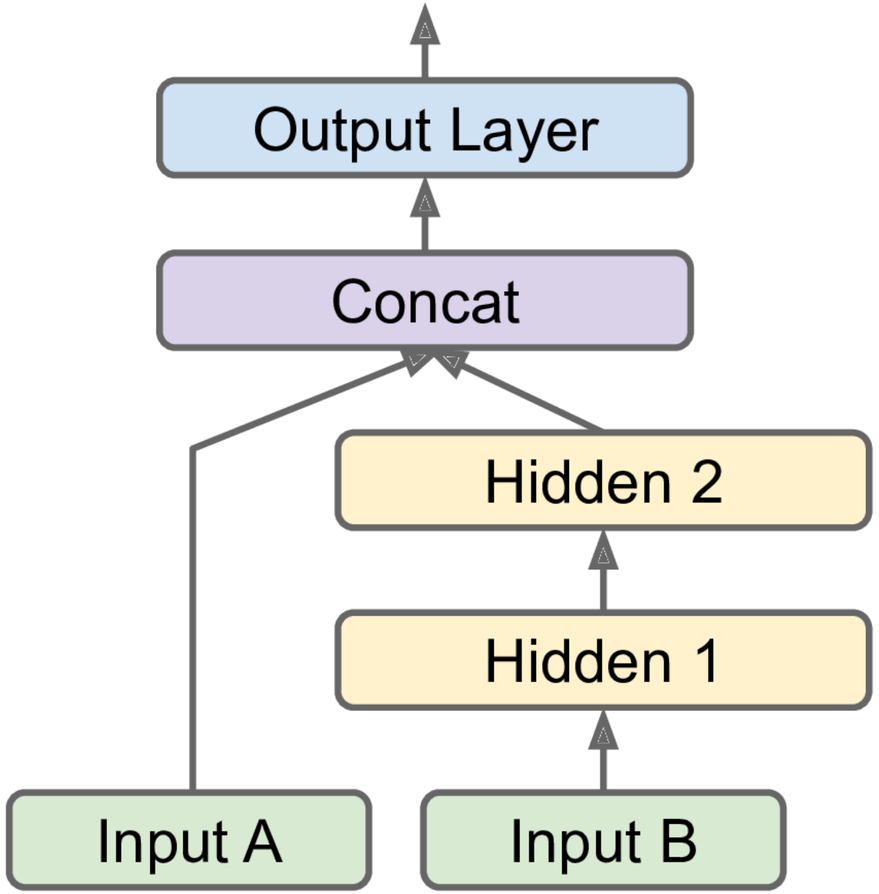
```
import tensorflow as tf
from tensorflow import kearas
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

housing = fetch_california_housing()  # 입력 특징에 숫자형만 포함된 데이터.  범주형(Ocean Proximity) 포함 안함
X_train_full, X_test, y_train_full, y_test = train_test_split(housing.data, housing.target)
X_train, X_valid, y_train, y_valid = train_test_split(X_train_full, y_train_full)

scaler = StandardScaler()                    # 입력 특징 scaling
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)

input_ = keras.layers.Input(shape=X_train.shape[1:])  # X_train.shaple=(11610, 8)
hidden1 = keras.layers.Dense(30, activation="relu")(input_)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input_], outputs=[output])

model.compile(loss="mean_squared_error", optimizer="sgd")
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
mse_test = model.evaluate(X_test, y_test)
X_new = X_test[:3] # pretend these are new instances
y_pred = model.predict(X_new)
```



Functional API 사용 예

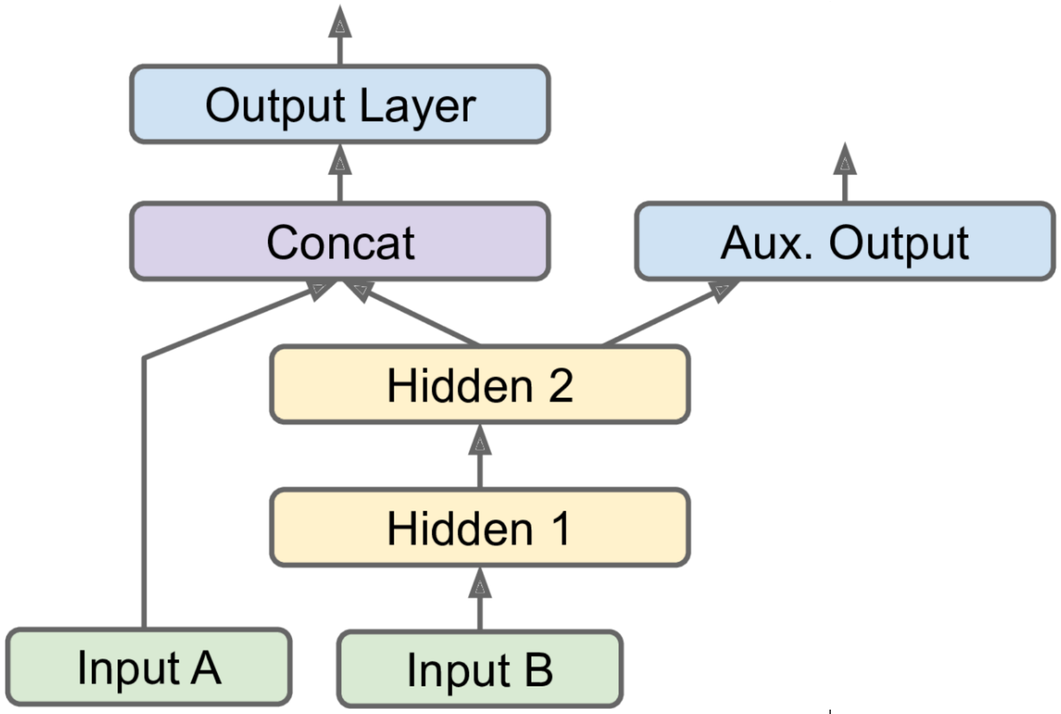
Wide%Deep Net2 : California Housing

```
import tensorflow as tf
from tensorflow import kearas
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

...

input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name="output")(concat)
model = keras.models.Model(inputs=[input_A, input_B], outputs=[output])

X_train_A, X_train_B = X_train[:, :5], X_train[:, 2:]
X_valid_A, X_valid_B = X_valid[:, :5], X_valid[:, 2:]
X_test_A, X_test_B = X_test[:, :5], X_test[:, 2:]
X_new_A, X_new_B = X_test_A[:3], X_test_B[:3]

model.compile(loss="mse", optimizer="sgd")
history = model.fit((X_train_A, X_train_B), y_train, epochs=20, validation_data=((X_valid_A, X_valid_B), y_valid))
mse_test = model.evaluate((X_test_A, X_test_B), y_test)
y_pred = model.predict((X_new_A, X_new_B))
```



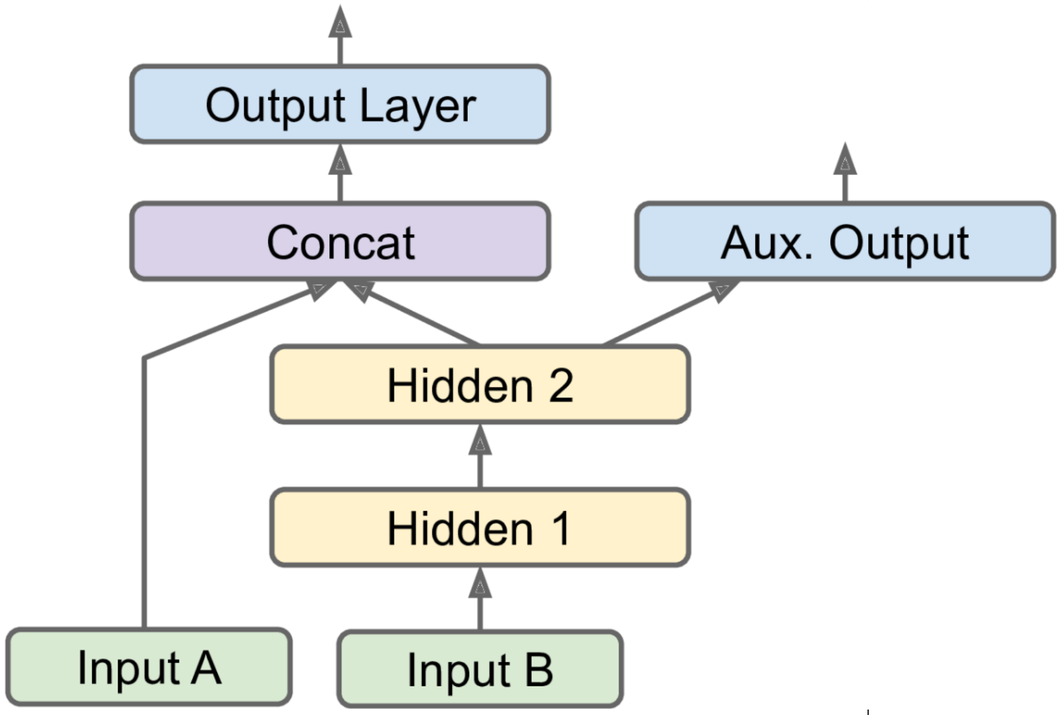
Functional API 사용 예 Wide%Deep Net3 : California Housing

```
import tensorflow as tf
from tensorflow import kearas
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


...
X_train_A, X_train_B = X_train[:, :5], X_train[:, 2:]
X_valid_A, X_valid_B = X_valid[:, :5], X_valid[:, 2:]
X_test_A, X_test_B = X_test[:, :5], X_test[:, 2:]
X_new_A, X_new_B = X_test_A[:3], X_test_B[:3]

input_A = keras.layers.Input(shape=[5], name="wide_input")
input_B = keras.layers.Input(shape=[6], name="deep_input")
hidden1 = keras.layers.Dense(30, activation="relu")(input_B)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name="main_output")(concat)
aux_output = keras.layers.Dense(1, name="aux_output")(hidden2)
model = keras.models.Model(inputs=[input_A, input_B], outputs=[output, aux_output])

model.compile(loss=["mse", "mse"], loss_weights=[0.9, 0.1], optimizer="sgd")
history = model.fit([X_train_A, X_train_B], [y_train, y_train], epochs=20, validation_data=([X_valid_A, X_valid_B], [y_valid, y_valid]))
total_loss, main_loss, aux_loss = model.evaluate([X_test_A, X_test_B], [y_test, y_test])
y_pred_main, y_pred_aux = model.predict([X_new_A, X_new_B])
```



Subclassing API 사용 예 Wide%Deep Net3 : California Housing

```
...

class WideAndDeepModel(keras.models.Model):
    def __init__(self, units=30, activation="relu", **kwargs):
        super().__init__(**kwargs)
        self.hidden1 = keras.layers.Dense(units, activation=activation)
        self.hidden2 = keras.layers.Dense(units, activation=activation)
        self.main_output = keras.layers.Dense(1)
        self.aux_output = keras.layers.Dense(1)

    def call(self, inputs):
        input_A, input_B = inputs
        hidden1 = self.hidden1(input_B)
        hidden2 = self.hidden2(hidden1)
        concat = keras.layers.concatenate([input_A, hidden2])
        main_output = self.main_output(concat)
        aux_output = self.aux_output(hidden2)
        return main_output, aux_output

model = WideAndDeepModel(30)

model.compile(loss=["mse", "mse"], loss_weights=[0.9, 0.1], optimizer="sgd")
history = model.fit([X_train_A, X_train_B], [y_train, y_train], epochs=20, validation_data=([X_valid_A, X_valid_B], [y_valid,
y_valid]))
total_loss, main_loss, aux_loss = model.evaluate([X_test_A, X_test_B], [y_test, y_test])
y_pred_main, y_pred_aux = model.predict([X_new_A, X_new_B])
```



동적(dynamic). Call()안에 조건문, 저수준 tf api 사용 가능.
모델을 저장할 수 없음 (층간 연결을 동적으로 변경할 수 있어 모델이 static이 아님)

콜백, 모델 저장

```
model = keras.models.Sequential([
    keras.layers.Dense(30, activation="relu", input_shape=[8]),
    keras.layers.Dense(30, activation="relu"),
    keras.layers.Dense(1)
])

model.compile(loss="mse", optimizer="sgd")

# set call back
checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.h5", save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)

# 학습할 때 call back 지정
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb, early_stopping_cb])

# save, load 원하면
#model.save("my_keras_model.h5")
#model = keras.models.load_model("my_keras_model.h5")
```

tensorboard 사용하기

[프로그램] 학습과정에서 정해진 폴더에 데이터(“summary”라고 부름) 저장

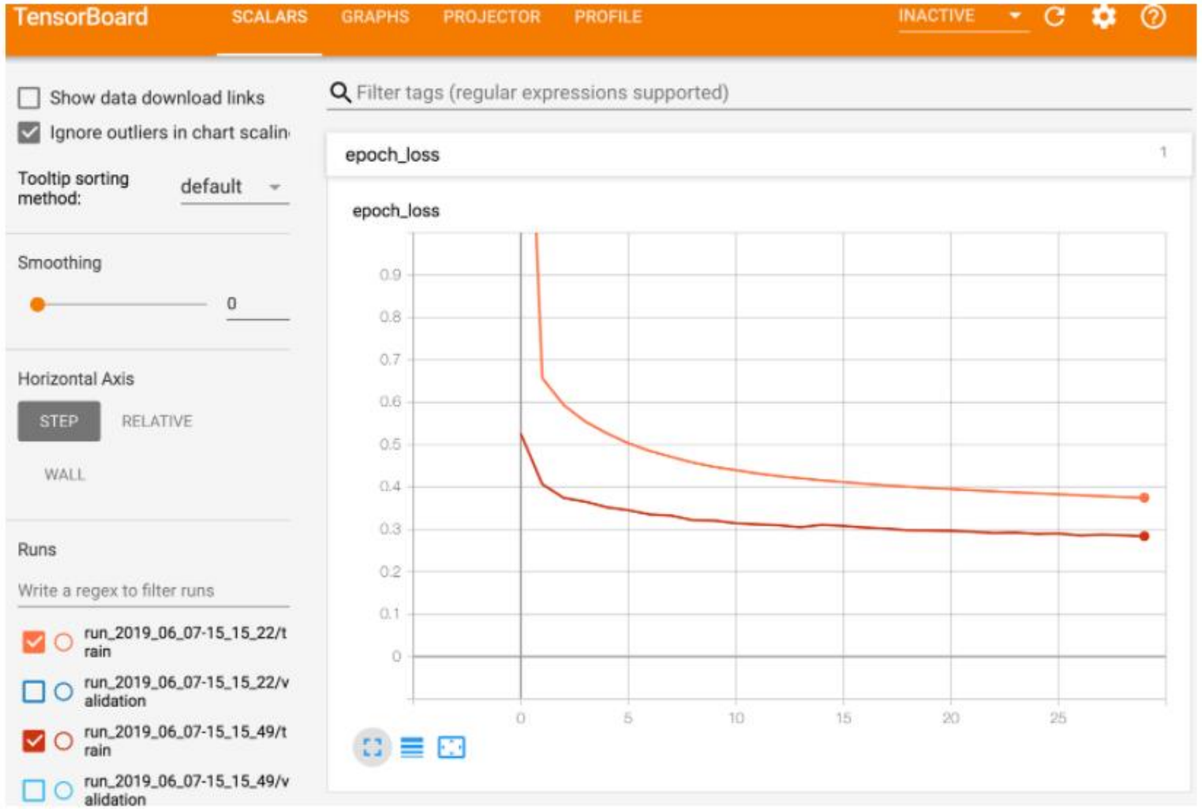
```
root_logdir = os.path.join(os.curdir, "my_logs")
def get_run_logdir():
    import time
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    return os.path.join(root_logdir, run_id)

tensorboard_cb = keras.callbacks.TensorBoard(get_run_logdir())
checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.h5",
                                                save_best_only=True)

history = model.fit(X_train, y_train, epochs=30,  validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb, tensorboard_cb])
```

[command line] tensorboad 실행 : 브라우저에서 그 폴더 정보를 시각화

```
tensorboard --logdir=./my_logs --port=6006
```

[[**web browser]** localhost:6006 하면 브라우저 화면에 표시됨

```
my_logs/
├── run_2019_06_07-15_15_22
│   ├── train
│   │   ├── events.out.tfevents.1559891732.mycomputer.local.38511.694049.v2
│   │   ├── events.out.tfevents.1559891732.mycomputer.local.profile-empty
│   │   └── plugins/profile/2019-06-07_15-15-32
│   │       └── local.trace
│   └── validation
│       └── events.out.tfevents.1559891733.mycomputer.local.38511.696430.v2
└── run_2019_06_07-15_15_49
    └── [...]
```



하이퍼파라메터 조정(hyperparameter Tuning)

- 네트워크 구조 (network topology)
- 레이어 수 (number of layers)
- 레이어당 뉴런 수(number of neurons per layer)
- 가중치 초기값 (weight initialization logic)
- 활성화 함수 (activation function)
- 규제 (regularization)
- 드랍아웃(drop out)
- 학습방법 (training method)

한 레이어에 뉴런 수를 많이 두는 것보다 여러 레이어에 뉴론 수를 적게 두는 것이 더 좋음. 일부 레이어는 미리 훈련한 것을 가져다 사용하고 나머지 레이어만 훈련하는 것도 가능(transfer learning)

은닉층에서는 ReLU, 출력층에서는 Softmax.
회기문제에는 출력층에 활성화함수 사용안힘

감사합니다