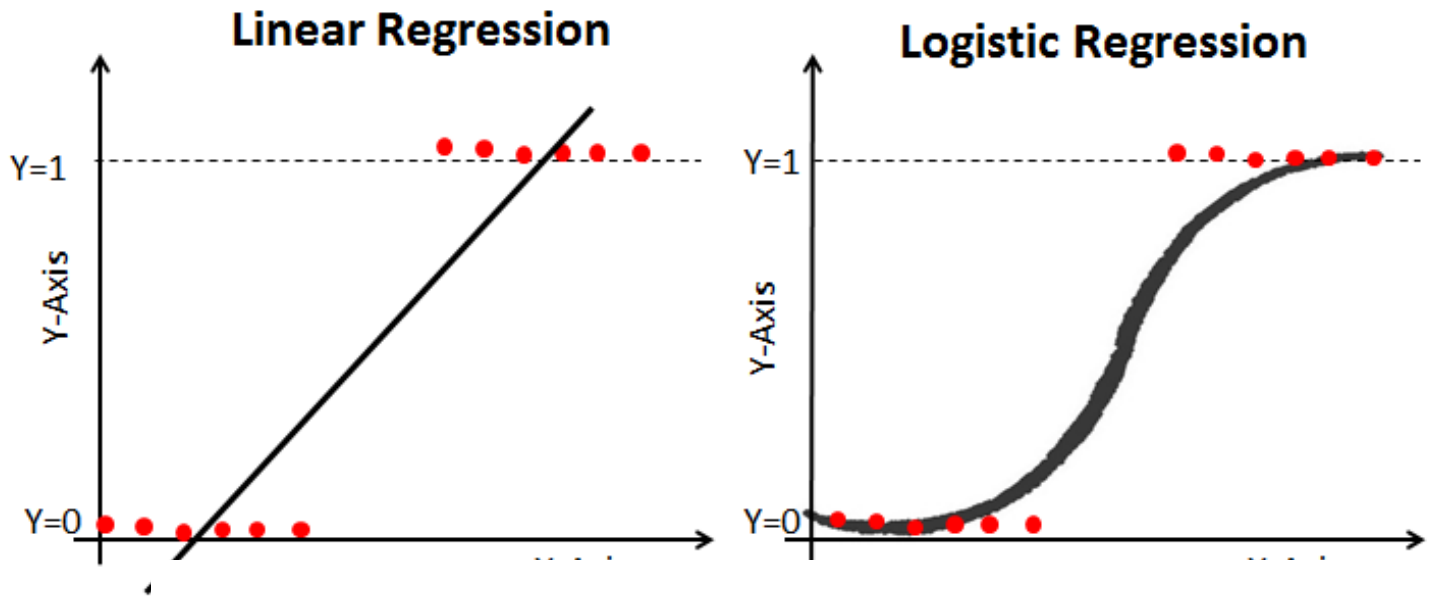
Keras를 이용한 순차 모델

명지대학교
MYONGJI UNIVERSITY

Linear Regression vs Logistic Regression

Linear Regression

Logitstic Regression



Linear Hypothesis

$$H(x) = Wx + b$$

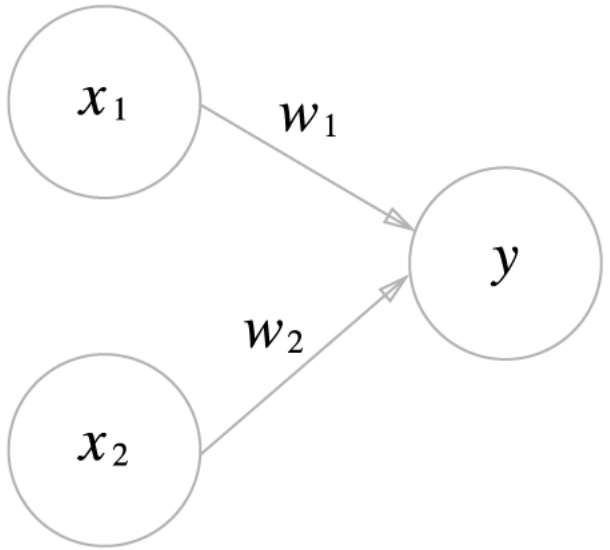
Logistic Hypothesis

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

sigmoid(z)

Perceptron

- 뉴런이 활성화: 뉴런에서 보내온 신호의 총합이 정해진 한계를 넘어서면 1을 출력
- 임계값: 정해진 한계. 세타 기호로 표현



$$y = 1(w_1x_1 + w_2x_2 > theta)$$

$$y = 0(w_1x_1 + w_2x_2 <= theta)$$

Tensorflow vs keras

TensorFlow 1.x는 이제

- 옛날 코드 ! (Legacy)
- 구글은 보안과 관련된 업데이트만 제공할 예정입니다
- 앞으로 몇 년 더 유지될지는

TensorFlow 2.0

- **이전과 마찬가지로 구글이 만든 오픈 소스 프레임워크**
- **머신러닝과 딥러닝을 위한 라이브러리**
- **Apache 라이선스**
- **TensorFlow 1.x 에 존재한 중복된 API 제거 , 빠른 프로토타입 및 쉬운 디버깅이 가능**
- **하위 호환성을 제공합니다**
  - tf.compat.v1 모듈 제공 tf.contrib 미포함
  - tf_upgrade_v2 라는 변환 스크립트를 제공
- **tensorflow.js v1.0 을 제공합니다**
- **분산 데이터를 위한 TensorFlow Federated 를 제공합니다**
- **ragged tensors 를 지원합니다**
  - tf.ragged.constant ([[3, 1, 4, 1], [], [5, 9, 2], [6],
- **확률적 추론 및 통계분석을 위한 TensorFlow Probability 를 제공합니다**

Keras Vs Tensorflow 2.0

- 근데 TensorFlow 2.0 과 Keras 는 무슨 관계야
- Keras 로 개발하는데 TensorFlow 2.0?
- 신경망을 학습시키는데 keras 패키지를 사용해야하나요
- 아니면tf.keras 를 사용해야하나요
- 어떤TensorFlow 2.0 기능에 관심을 가져야 하나요

TensorFlow와 Keras 의 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 좋았습니다. 그러나, 이런 백엔드 (backend)가 필요했습니다.

2015년 11월 9일 TensorFlow 첫 공개

천천히, 조금씩 Keras는 백엔드로 TensorFlow를 지원하기 시작합니다

2016년 9월 20일 Keras 1.1.0 공개

기본 백엔드가 TensorFlow로 변경됩니다.

서브 모듈에 tf.keras가 포함됩니다.
즉, TensorFlow와 Keras와의 통합이 시작됩니다.
이제 Keras는 tf.keras는 별도의 패키지!?

2018년 8월 9일 TensorFlow 1.10.0 공개

2019년 6월 구글이 TensorFlow 2.0 개발을 알림

Keras가 TensorFlow의 공식 고수준 API라고 선언합니다!?

Francois Chollet 가 말씀하시길

1. Keras와 tf.keras가 이제 같습니다.
2. 이제 앞으로는 멀티 백엔드 지원 안 합니다.
3. 앞으로 TensorFlow 2.0과 tf.keras를 사용하세요.
4. Keras는 버그 정도만 수정할 예정!

2019년 9월 18일 Keras 2.3.0 공개

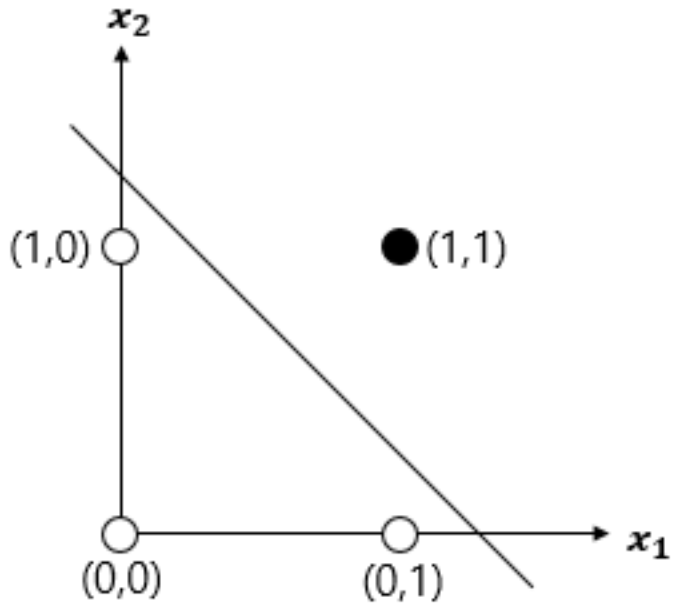
출처:https://tykimos.github.io/warehouse/2019-12-17_TensorFlow_2_0_NN_CV.pdf

경사하강법 & 역전파 (Backpropagation)

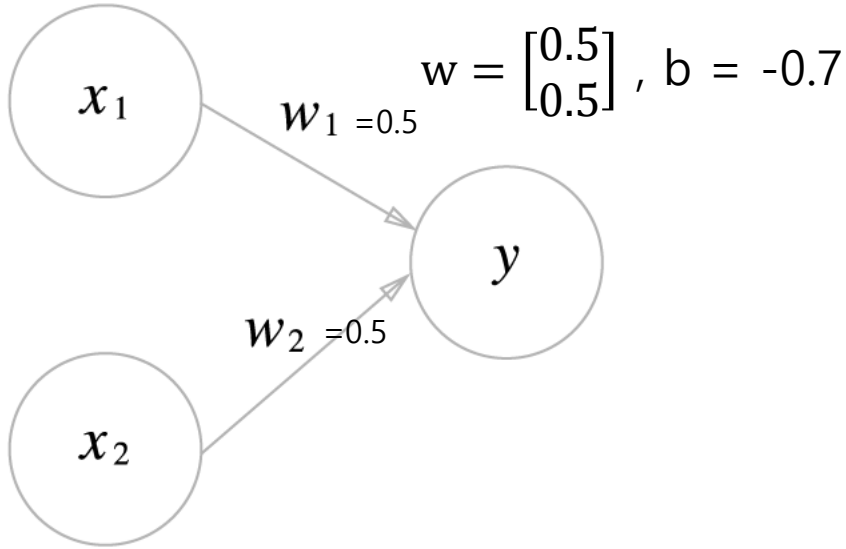
명지대학교
MYONGJI UNIVERSITY

단순한 논리 회로

AND 게이트



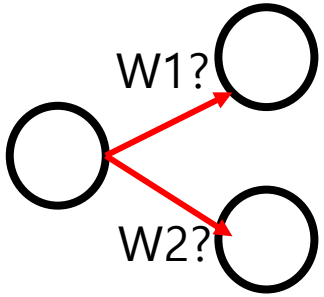
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



Numpy로 표현

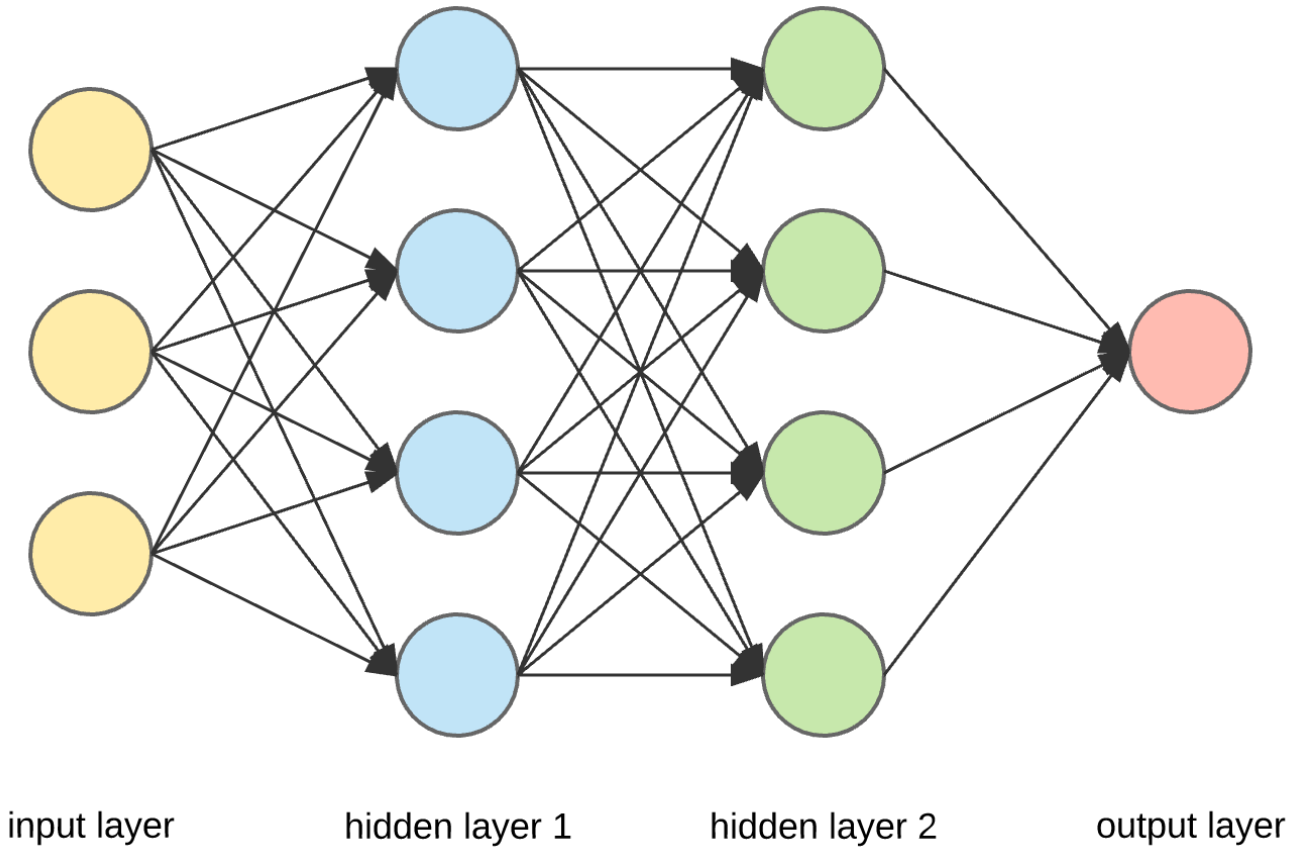
x1	x2	x1*w1 + x2*w2+b1		X3
0	0	0* 0.5 + 0 * 0.5 - 0.7 =-0.7	<0	0
1	0	1* 0.5 + 0 * 0.5 - 0.7 =-0.2	<0	0
0	1	0* 0.5 + 1 * 0.5 - 0.7 =-0.2	<0	0
1	1	1* 0.5 + 1 * 0.5 - 0.7 =1.7	>0	1

Keras를 이용한 예측




```
import keras
import numpy as np
from keras.layers import Dense
from keras.models import Sequential

w = 10
b = 2

x_train = np.array([[0],[1]])
y_train = x_train * w + b

x_test = np.array([[2],[2]])
y_test = x_test * w + b

model = Sequential() # 순차 모델 만들기
# 모델은 (*, 1) 형태의 배열을 인풋으로 받고
# (*, 2) 형태의 배열을 출력합니다
model.add(Dense(2,input_shape=(1,)))

# 첫 번째 레이어 이후에는,
# 인풋의 크기를 특정하지 않아도 됩니다:
model.summary()

model.compile(loss='mean_squared_error',optimizer='SGD')
model_fit = model.fit(x_train,y_train,batch_size=3,epochs=100,verbose=2)
#callbacks=[keras.callbacks.TensorBoard(log_dir='./graph', histogram_freq=1)]
# fit함수의 첫번째 인자는 x 입력값
# fit함수의 두번째 인자는 y 입력값
# fit함수의 세번째 인자는 학습시킬때의 묶음 샘플수
# fit함수의 두번째 인자는 학습의 횟수
# fit함수의 두번째 인자는 verbose 학습 진행과정을 보여줄

# (*, 2) 형태의 배열을 출력합니다
y_hat = model.predict(x_test)
print(y_hat)
print("Y-test:",y_test)
```

다층 퍼셉트론(MLP)



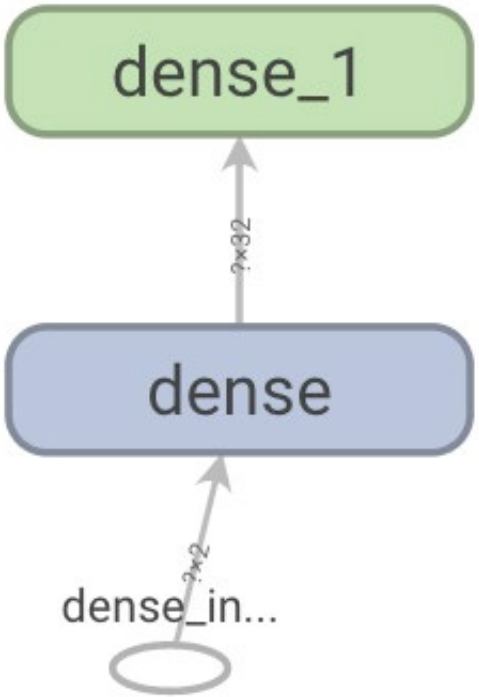
https://miro.medium.com/proxy/1*Gh5PS4R_A5drl5ebd_gNrg@2x.png

Keras를 이용한 XOR – 1layer

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Activation, Dense

training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")

model = Sequential()
model.add(Dense(32, input_dim=2, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

model.fit(training_data, target_data, epochs=1000, verbose=2)

print( model.predict(training_data))
```

Keras를 이용한 XOR – 1 layer

```
1/1 - 0s - loss: 0.1116 - binary_accuracy: 1.0000
Epoch 998/1000
1/1 - 0s - loss: 0.1112 - binary_accuracy: 1.0000
Epoch 999/1000
1/1 - 0s - loss: 0.1108 - binary_accuracy: 1.0000
Epoch 1000/1000
1/1 - 0s - loss: 0.1103 - binary_accuracy: 1.0000
```

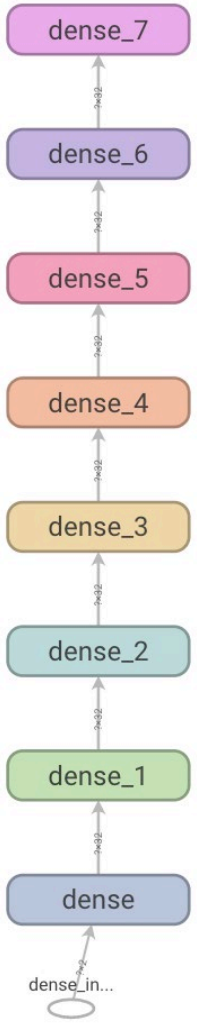


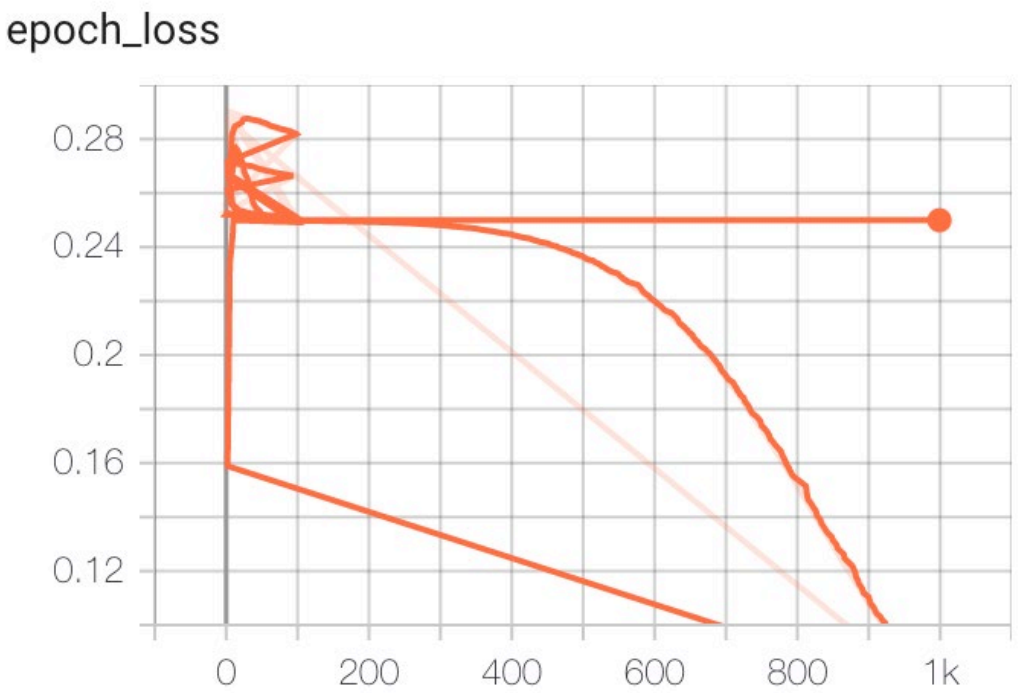
Keras를 이용한 XOR – 8 layers

```
import numpy as np
import tensorflow as tf
from keras.models import Sequential
from keras.layers.core import Activation, Dense

training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")

model = Sequential()
model.add(Dense(32, input_dim=2, activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(32, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

model.fit(training_data, target_data, epochs=1000, verbose=2,
          callbacks=[tf.keras.callbacks.TensorBoard(log_dir='./graph', histogram_freq=1)])

print( model.predict(training_data))
```

Keras를 이용한 XOR – 8 layers

```
Epoch 997/1000
1/1 - 0s - loss: 0.2499 - binary_accuracy: 0.7500
Epoch 998/1000
1/1 - 0s - loss: 0.2499 - binary_accuracy: 0.7500
Epoch 999/1000
1/1 - 0s - loss: 0.2499 - binary_accuracy: 0.7500
Epoch 1000/1000
1/1 - 0s - loss: 0.2499 - binary_accuracy: 0.7500
```



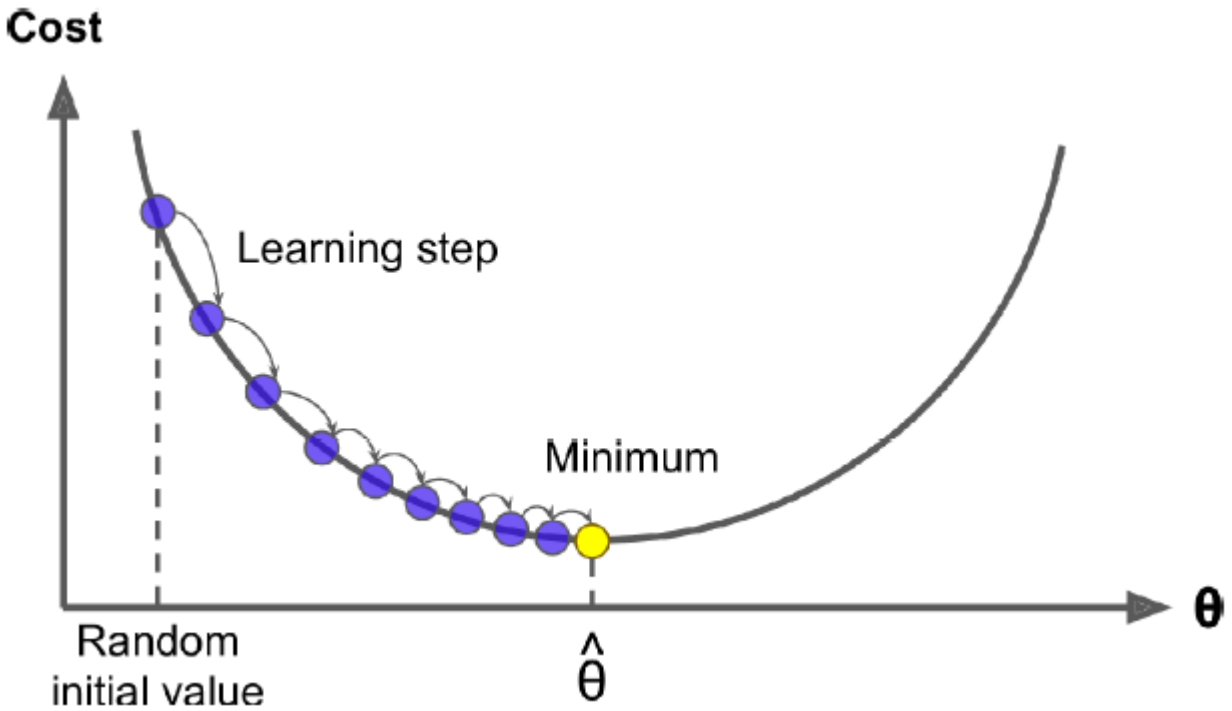


Tensorboard

```
python3 -m tensorboard.main  --logdir=./

tensorboard  --logdir='./graph/'
```

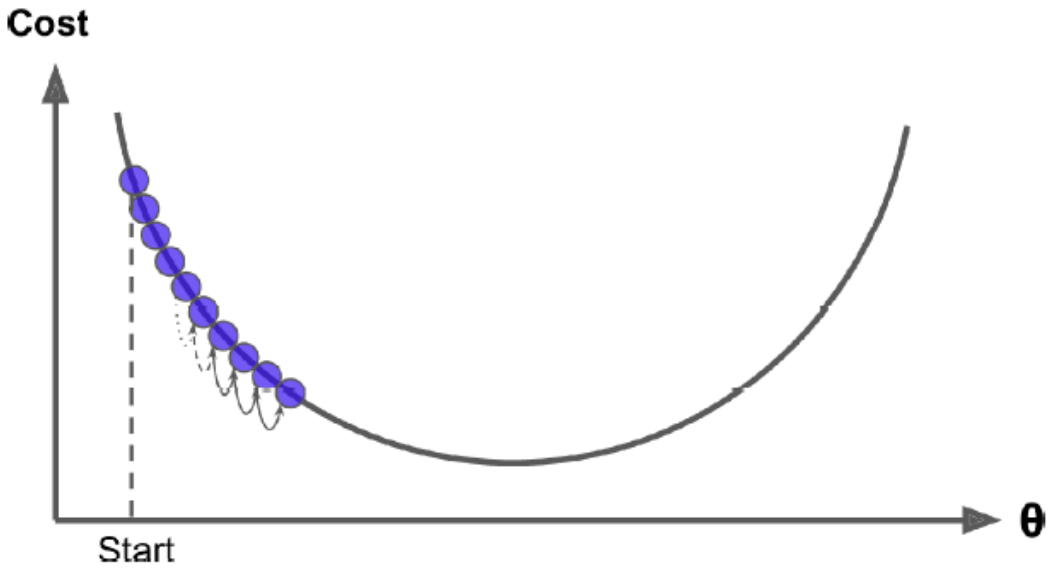
경사 하강법

- 비용함수를 최소화 하기 위해서 반복해서 파라미터를 조정해 가는 것
- θ를 임의로 시작해서 (무작위 초기화) 조금씩 비용이 감소되는 방향으로 진행하여 알고리즘이 최솟값에 수행 할때까지 점진적으로 향상

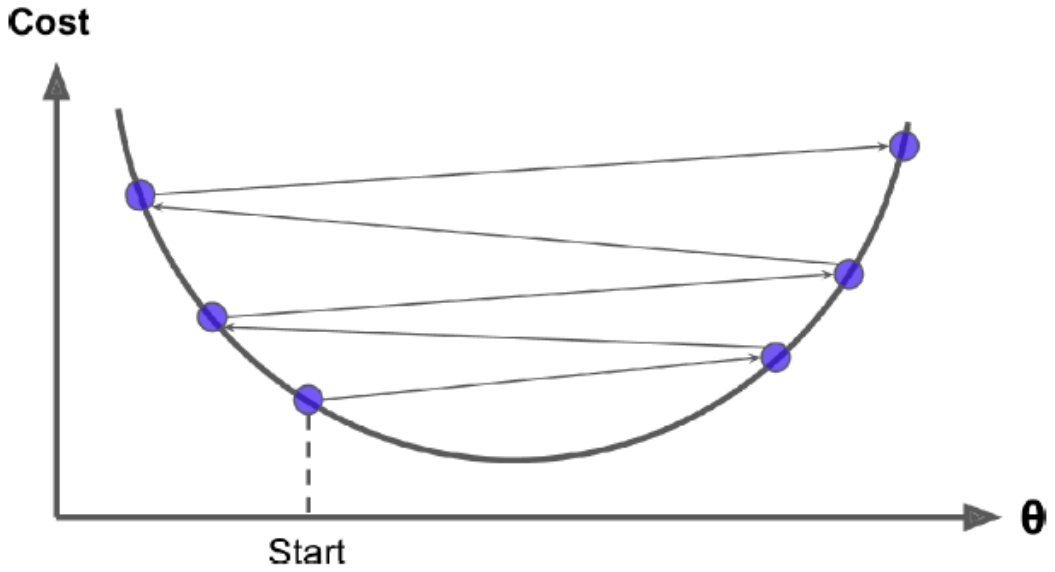


경사 하강법

중요한 파라미터는 스텝의 크기로 학습률 결정



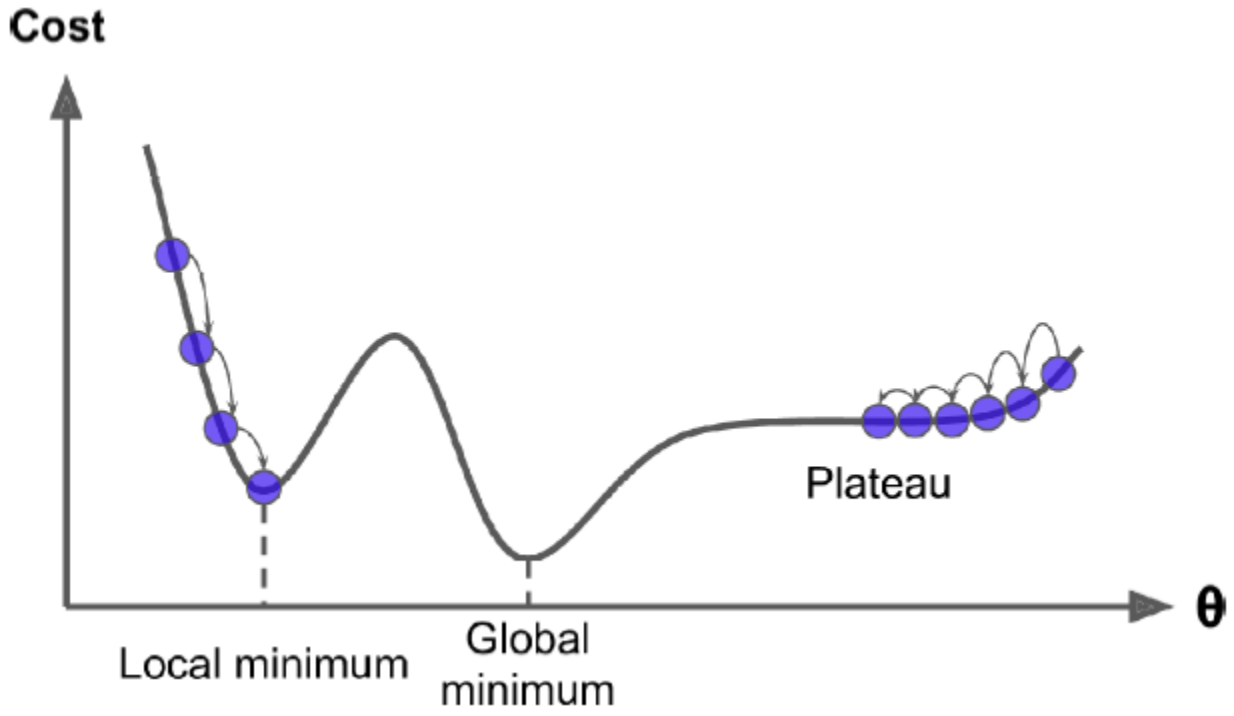
학습률이 너무 작으면 알고리즘이 수렴하기 때문에 반복 진행



학습률이 너무 크면 반대편으로 건너 발산

경사 하강법

경사 하강법의 문제점



경사 하강법

배치 경사 하강법

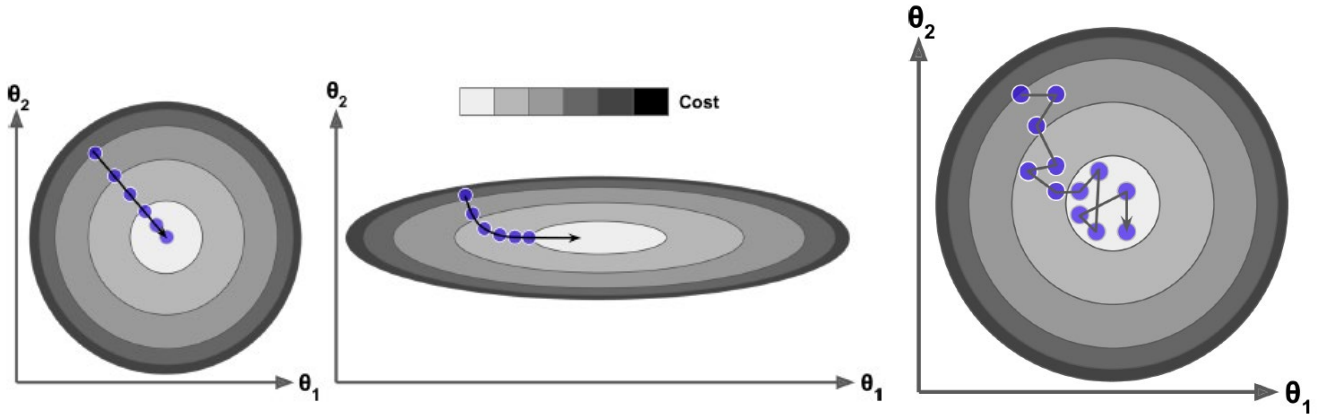
- 전체 훈련세트를 사용해서 그레이언트 계산

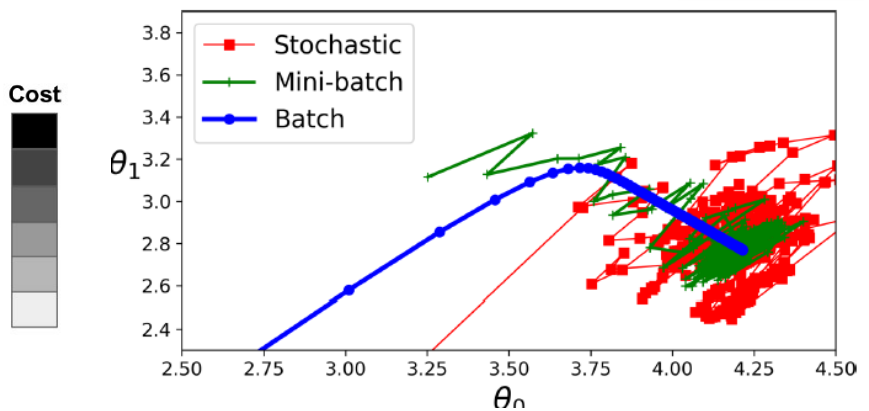
확률적 경사 하강법

- 매 스텝에서 한 개의 샘플으 무작위로 선택하여 그래디언트 계산

미니배치 경사 하강법

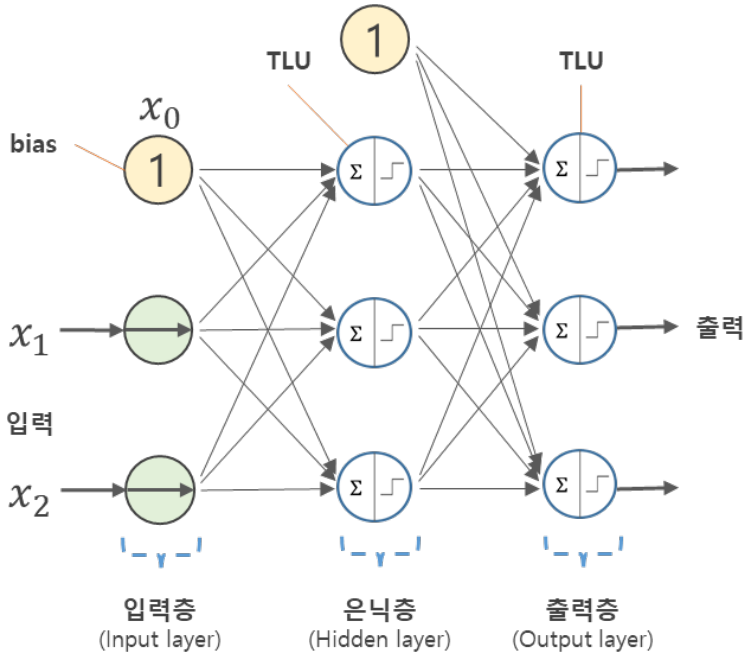
- 미니 배치라고 부르는 임의의 작은 샘플에 세트에 대해 그레디언트 계산





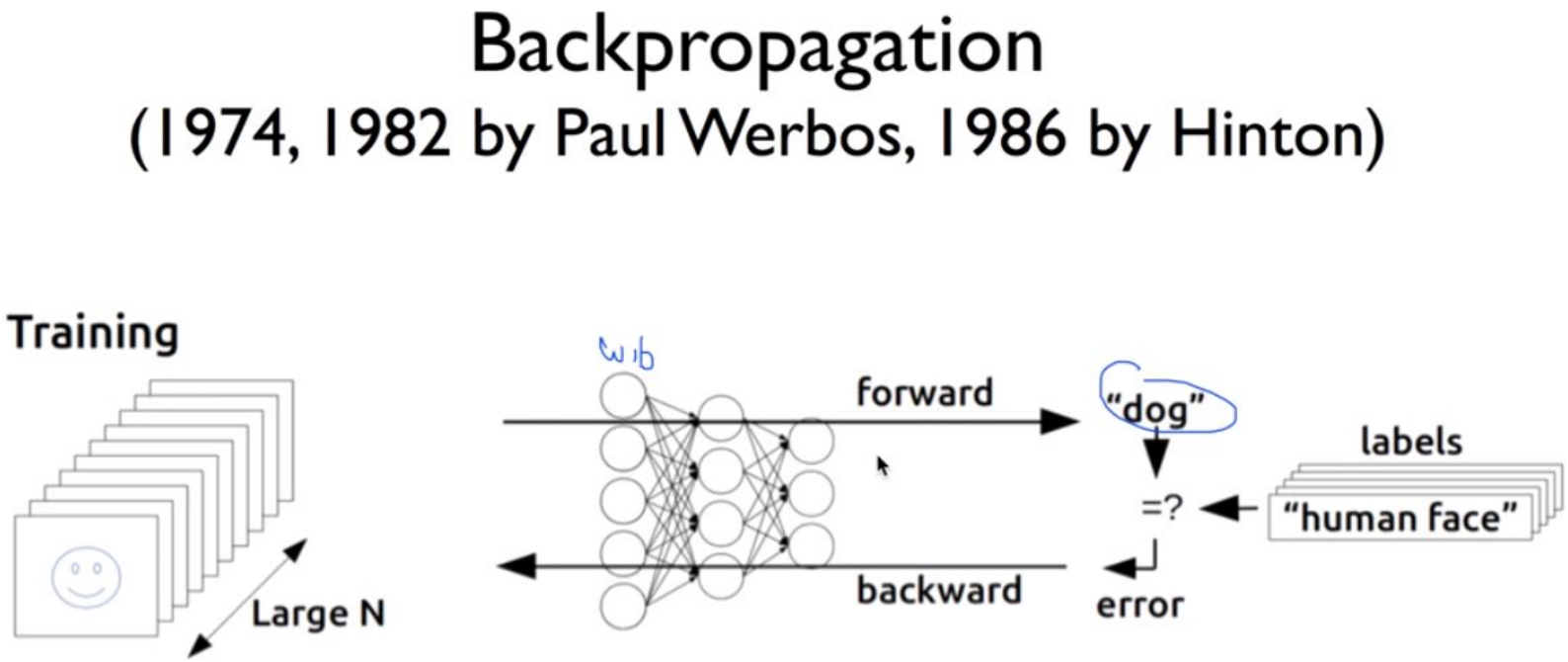
다층 퍼셉트론(MLP)

- 입력층, **은닉층**(hidden layer)이라 부르는 하나 이상의 TLU 층과 마지막으로 **출력층**(output layer)으로 구성
- 인공 신경망의 은닉층이 2개 이상일 때, **심층 신경망**(**DNN**, Deep Neural Network)라 하고 이를 학습하여 모델을 만드는 것을 우리가 익히 들어온 **딥러닝**(Deep-Learning)



역전파(backpropogation)

- 그래디언트를 자동으로 계산하는 경사 하강법
- 네트워크를 정방향 역방향 통과하는 것만으로 이 역전파 알고리즘은 모델 파라미터에 대한 네트워크 오차의 그레디언트를 계산



https://smartstuartkim.files.wordpress.com/2019/02/backpropagation.png

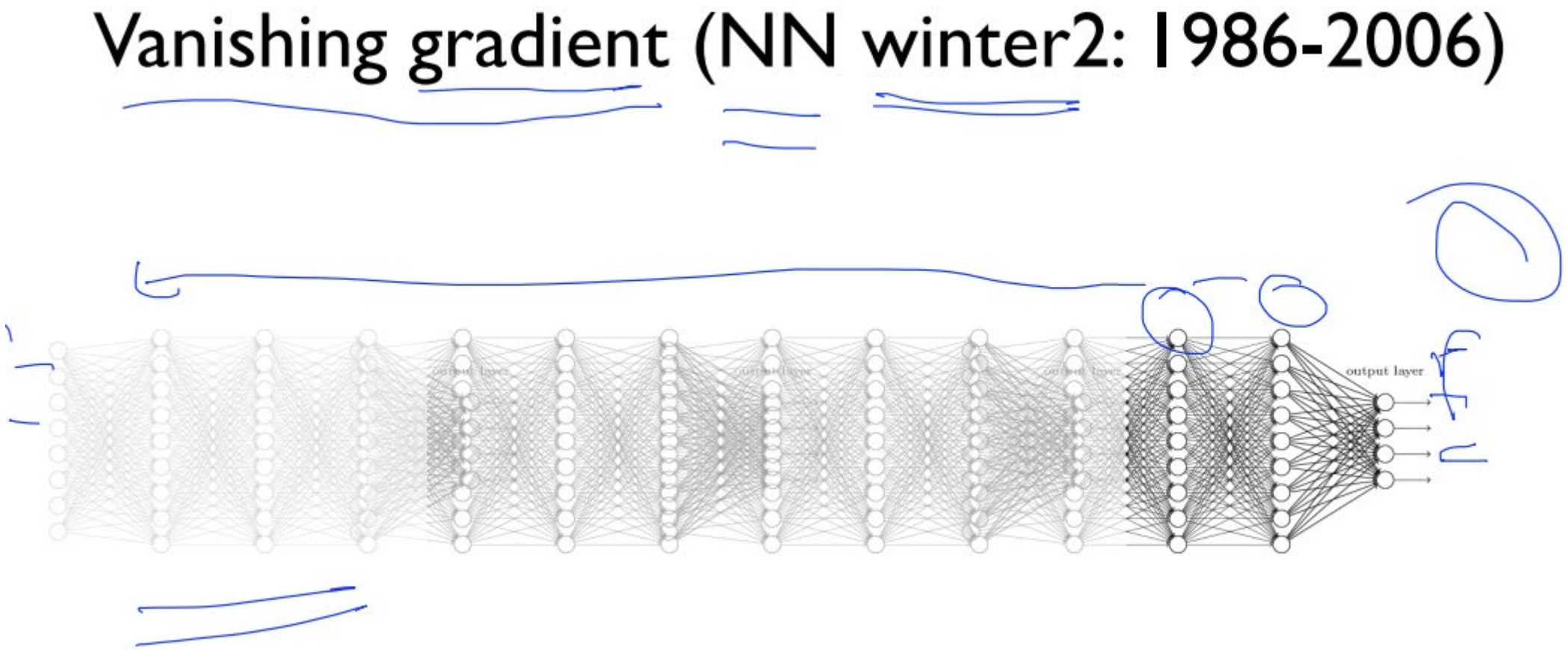
역전파(backpropogation)

- 층이 깊어질 수록 증가하는 가중치 매개변수의 수로 인해 다층 퍼셉트론을 학습시키기에는 오랜 시간이 걸리는 문제가 발생

역전파(backpropogation)

- 먼저, 각 학습 데이터 샘플을 네트워크에 입력으로 넣어주고 출력층까지 각 층의 뉴런 마다 출력을 계산한다. 이를 **순전파**(forward propagation)이라고 한다.
- 그 다음 네트워크의 마지막 출력층에 대한 결과(예측값)와 실제값과의 차이, 즉 오차(error)를 계산하는데, 손실함수(loss function)를 이용하여 계산한다.
- 그리고 이 오차를 역방향으로 흘러 보내면서, 각 출력 뉴런의 오차에 마지막 입력 뉴런이 얼마나 기여했는지 측정한다. 이말을 쉽게 설명하면, **각 뉴런의 입력값에 대한 손실함수의 편미분, 그래디언트(gradient)을 계산**하는 것을 말한다.
- 3번과 같은 방법을 입력층에 도달할 때까지 계속 반복해서 역방향으로 흘러 보낸다.
- 마지막으로, 계산한 그래디언트를 네트워크의 모든 가중치 매개변수에 반영해주는 **경사 하강법 단계**를 수행한다.

역전파(backpropogation)



활성화 함수

명지대학교
MYONGJI UNIVERSITY

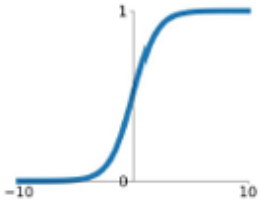
활성화 함수 (activation function)

- 역전파 알고리즘이 잘 동작하기 위해서 다층 퍼셉트론(MLP)의 구조에 변화를 주었는데, 그것이 바로 활성화 함수 부분에서 계단 함수를 시그모이드 함수(로지스틱 함수)로 바꿔준 것
- 가중치 매개변수를 조정 해주기 위해 그래디언트, 편미분을 계산하게 되는데, 계단 함수는 0을 기준으로 기울기가 없는 직선이므로 그래디언트를 계산하는 것이 의미가 없기 때문

활성화 함수 (activation function)

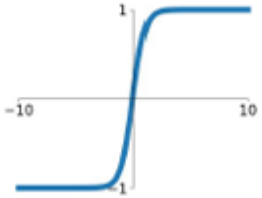
Sigmoid

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



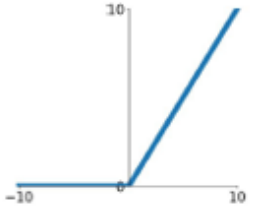
tanh

$$\tanh(x)$$



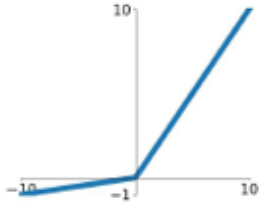
ReLU

$$\max(0, x)$$



Leaky ReLU

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

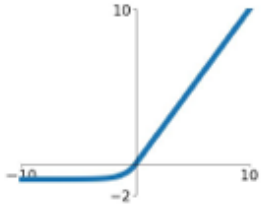


Maxout

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

ELU

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




```
def sigmoid(z):
        return 1 / (1 + np.exp(-z))
def relu(z):
        return np.maximum(0, z)
```

활성화 함수 (activation funct

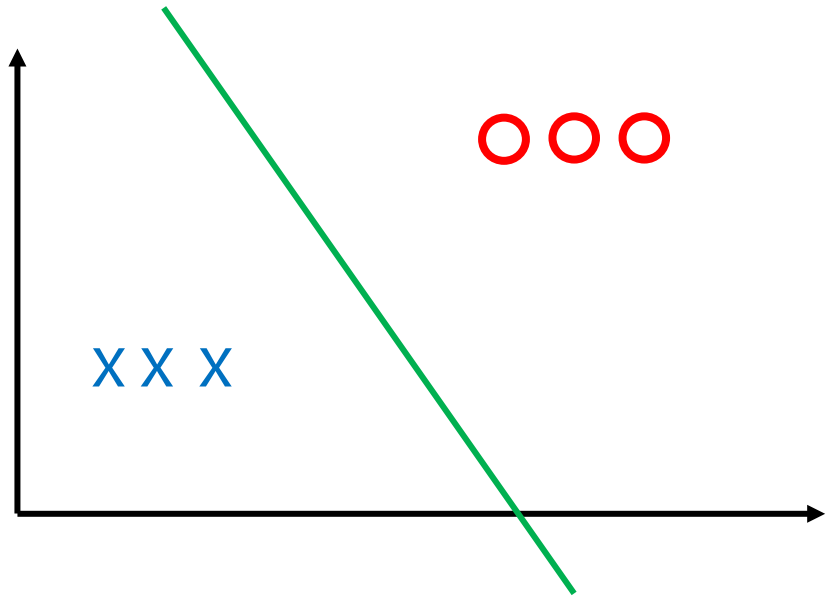
```
def heaviside(z):
    return (z >= 0).astype(z.dtype)

def mlp_xor(x1, x2, activation=heaviside):
    return activation(-activation(x1 + x2 - 1.5) + activat
ion(x1 + x2 - 0.5) - 0.5)
```

```
x1s = np.linspace(-0.2, 1.2, 100)
x2s = np.linspace(-0.2, 1.2, 100)
x1, x2 = np.meshgrid(x1s, x2s)

z1 = mlp_xor(x1, x2, activation=heaviside)
z2 = mlp_xor(x1, x2, activation=sigmoid)

plt.figure(figsize=(10,4))

plt.subplot(121)
plt.contourf(x1, x2, z1)
plt.plot([0, 1], [0, 1], "gs", markersize=20)
plt.plot([0, 1], [1, 0], "y^", markersize=20)
plt.title("Activation function: heaviside", fontsize=14)
plt.grid(True)

plt.subplot(122)
plt.contourf(x1, x2, z2)
plt.plot([0, 1], [0, 1], "gs", markersize=20)
plt.plot([0, 1], [1, 0], "y^", markersize=20)
plt.title("Activation function: sigmoid", fontsize=14)
plt.grid(True)
plt.show()
```

Multinomial Classification

명지대학교
MYONGJI UNIVERSITY

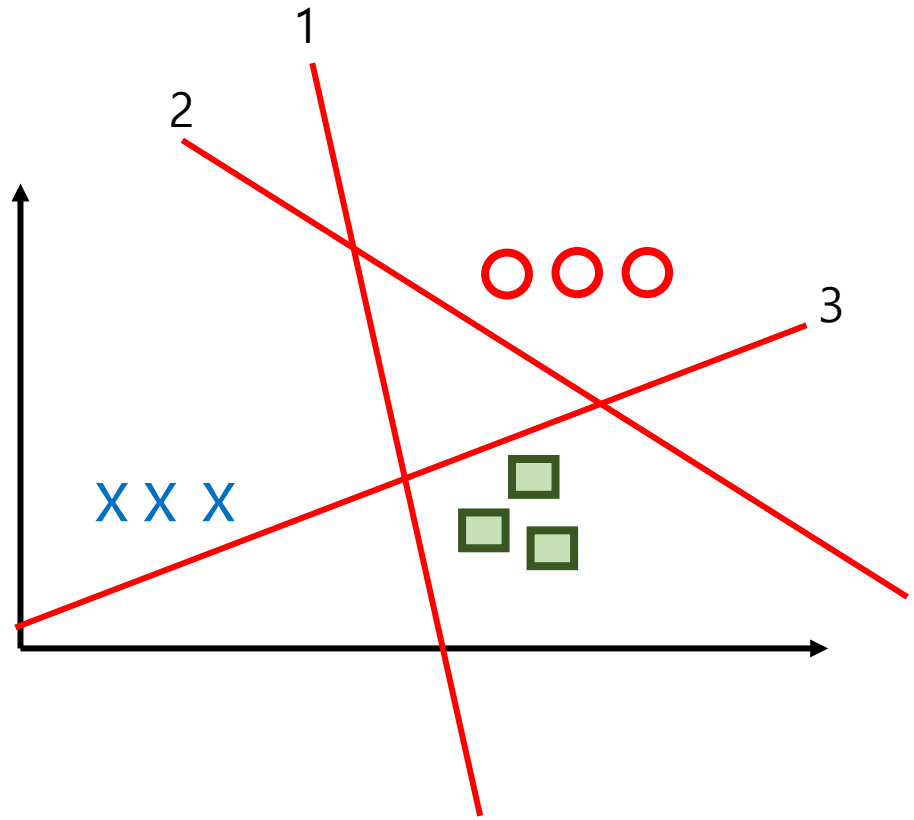
Logistics regression



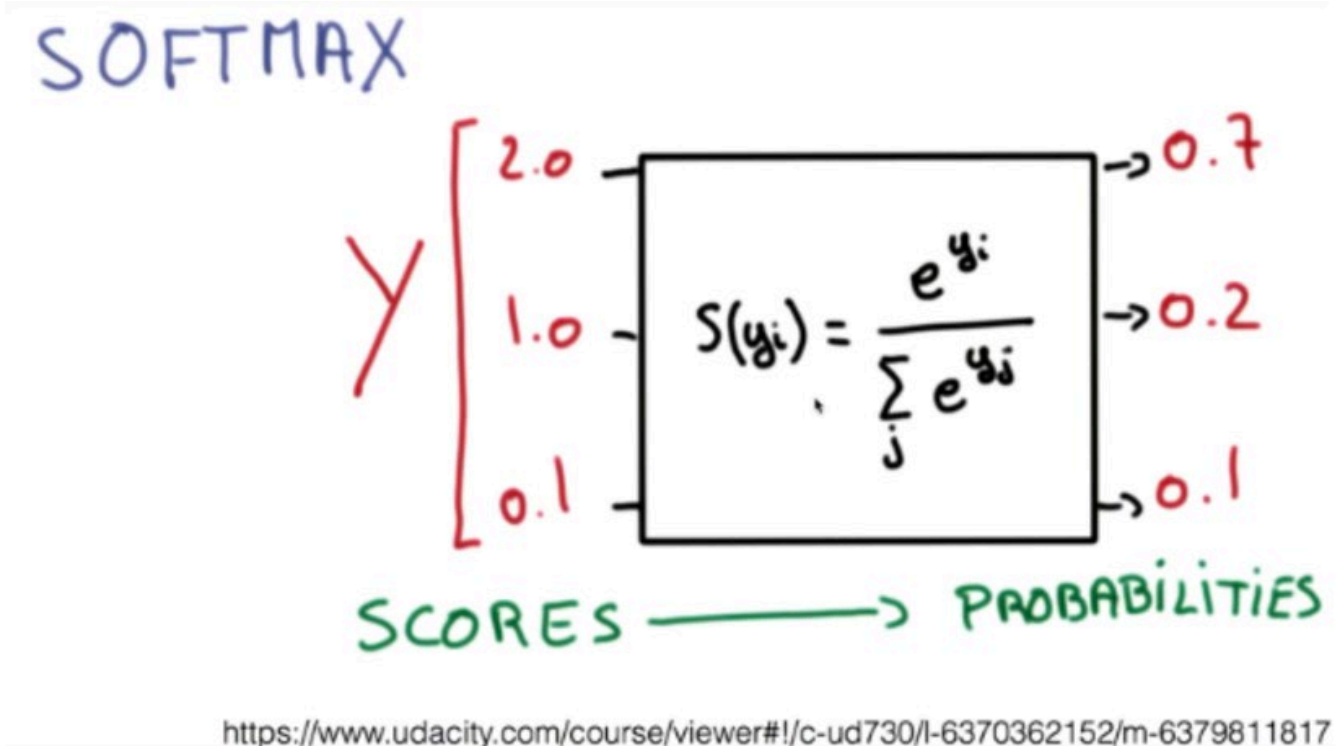
입력-> 가중치 곱하기->액티브 함수->hat of Y

Multinomial classification

- 다중 분류를 해결하기 위한 모델을 제안
  - 참고 : Logistic Regression(Logistic Regression 혹은 Linear_Regression은 연산결과 0~1사이의 확율값으로 표현하고 둘중 하나로 결론



Keras를 이용한 softmax



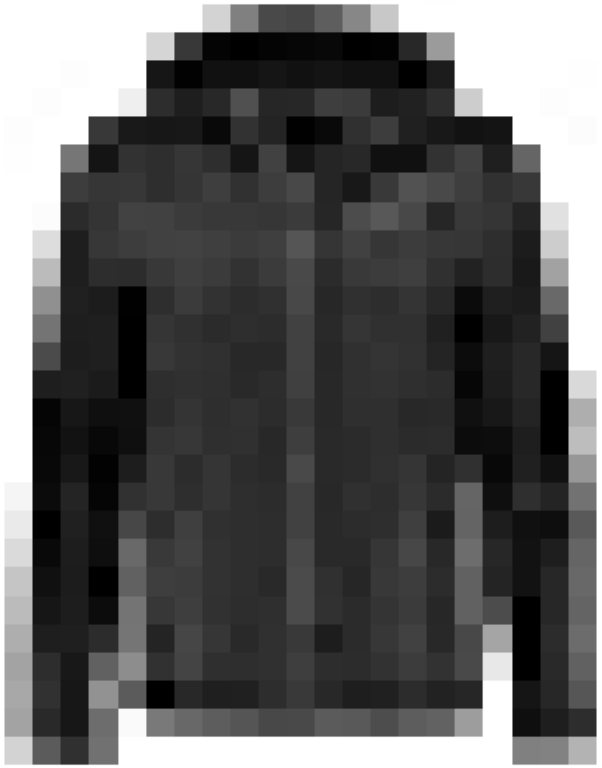
https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

```
keras.layers.Dense(10, activation="softmax")
```

Keras를 이용한 fashion mnist

명지대학교
MYONGJI UNIVERSITY

Keras를 이용한 fashion mnist



```
import sys
assert sys.version_info >= (3, 5)
from tensorflow import keras
# 사이킷런 ≥0.20 필수
import sklearn
assert sklearn.__version__ >= "0.20"

# 텐서플로 ≥2.0 필수
import tensorflow as tf
assert tf.__version__ >= "2.0"

# 공통 모듈 임포트
import numpy as np
import os
import matplotlib as mpl
import matplotlib.pyplot as plt

# 먼저 MNIST 데이터셋을 로드하겠습니다.
# 케라스는 `keras.datasets`에 널리 사용하는 데이터셋을 로드하기 위한 함수를
니다.
# 이 데이터셋은 이미 훈련 세트와 테스트 세트로 나누어져 있습니다.
# 훈련 세트를 더 나누어 검증 세트를 만드는 것이 좋습니다:

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., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.

plt.imshow(X_train[0], cmap="binary")
plt.axis('off')
plt.show()
```

Keras를 이용한 fashion mnist


```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

n_rows = 4
n_cols = 10
plt.figure(figsize=(n_cols * 1.2, n_rows * 1.2))
for row in range(n_rows):
    for col in range(n_cols):
        index = n_cols * row + col
        plt.subplot(n_rows, n_cols, index + 1)
        plt.imshow(X_train[index], cmap="binary", interpolation="nearest")
        plt.axis('off')
        plt.title(class_names[y_train[index]], fontsize=12)
plt.subplots_adjust(wspace=0.2, hspace=0.5)
save_fig('fashion_mnist_plot', tight_layout=False)
plt.show()
```

Keras를 이용한 fashion mnist

```
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")
])
```

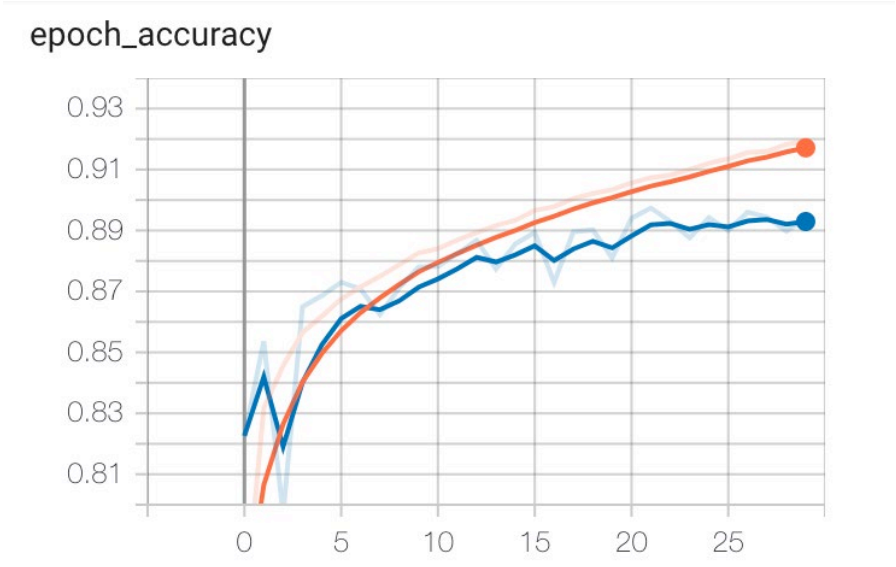
Keras를 이용한 fashion mnist

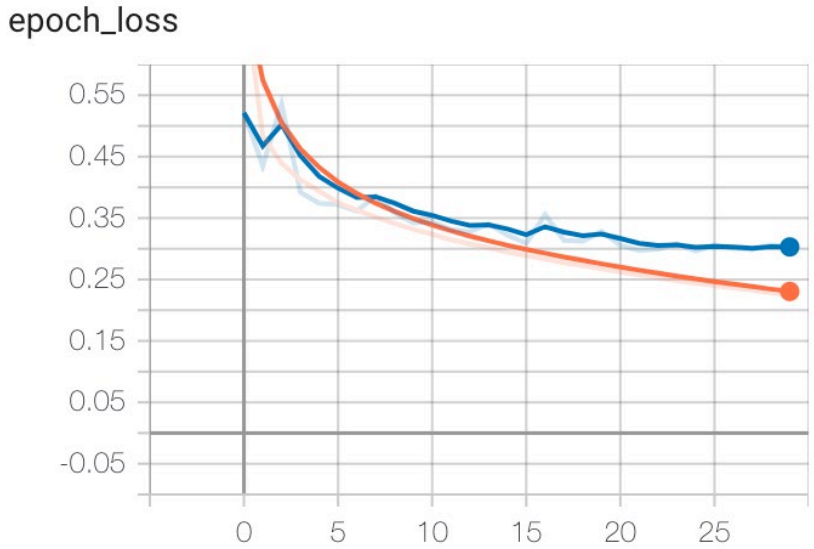
```
model.layers
model.summary()
hidden1 = model.layers[1]
hidden1.name
model.get_layer(hidden1.name) is hidden1
weights, biases = hidden1.get_weights()
weights
weights.shape
biases
biases.shape
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
import time
tb_hist = keras.callbacks.TensorBoard(log_dir='./graph', histogram_freq=0, write_graph=True, write_ima
ges=True)
start = time.time()
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid),callbacks=[tb_hist])
print("time :", time.time() - start)
```

Tensorboard

```
python3 -m tensorboard.main  --logdir=./

tensorboard  --logdir='./graph/'
```

Tensorboard





이미지 및 코드 출처
https://github.com/ExcelsiorCJH/Hands-On-ML/
https://github.com/rickiepark/handson-ml2

```
import numpy as np
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

print (keras.__version__)
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train_new, y_train_new = x_train[(y_train==0) | (y_train==1)], y_train[(y_train==0) | (y_train==1)]
x_test_new, y_test_new = x_test[(y_test==0) | (y_test==1)], y_test[(y_test==0) | (y_test==1)]
x_test_final = x_test_new.reshape((-1, 784))
x_train_final = x_train_new.reshape((-1, 784))
x_train_final = x_train_final / 255
x_test_final = x_test_final / 255
model = keras.models.Sequential()
model.add(Dense(1, input_shape=(784,), activation='sigmoid'))


model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['binary_accuracy'])
model.fit(
  x=x_train_final,
  y=y_train_new,
  shuffle=True,
  epochs=5,
  batch_size=16,
  callbacks=[tf.keras.callbacks.TensorBoard(log_dir='./graph', histogram_freq=1)])
```