Week 2

ML/DL Basic 안태영

Lecture 06: Softmax Regression

Multinomial Classification

- Binary classfication 모델을 여러 개 만들어서 여러가지 군으로 분리하는 것
- 2차원 행렬을 연산하여 진행한다
- ex) 3개의 label을 분류한다 했을 때는
  $w_{11}x_1 + w_{12}x_2 + w_{13}x_3, w_{21}x_1 + w_{22}x_2 + w_{23}x_3, w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \cdots$ 이런식으로 나열해야 한다 하지만
  행렬 연산을 진행 했을땐, $\begin{bmatrix} w_{11}w_{12}w_{13} \\ w_{21}w_{22}w_{23} \\ w_{31}w_{32}w_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ 로 표현 가능함

Softmax

- 어떤 입력 값에 대해서 각각의 원소에 대한 확률을 나타내주는 형태의 sigmoid 대체 함수
- 값들은 0과 1사이이다
- 각각의 원소의 합들이 1로 나타내어 진다
- $S(y_i) = \frac{e^{y_i}}{\sum e^{y_j}}$

One-Hot Encoding

- Softmax 함수를 거쳐서 나온 값중 가장 큰 값의 확률을 1로 바꾸는 과정
- tensorflow에서는 argmax라는 함수가 담당한다

Lecture 06: Softmax Regression

> **Cross Entropy**
>
> - Multi label classification에서 cost
> - -$\sum_i L_i \log S_i = \sum_i L_i * (-\log S_i)$

> **Cost Function**
>
> - $Loss = \frac{1}{N}\sum_i D(S(wx_i + b), L_i)$
> - 여기서 $S(wx_i + b)$는 y 값, L_i는 확률 값이다

> **Gradient Descent**
>
> - 이번엔 각각의 weight 벡터에 대한 gradient의 편미분을 말하는 것이다

Lab 06-1: Softmax Classification Eager

[26]

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

tf.random.set_seed(777)  # for reproducibility
```

Data를 벡터 형태로 담는다

nb_class

→ nb_class는 몇개의 sector로 분류할 것인지에 대한 변수로, 행의 개수를 정의하는 것이다

[5]

```
#Data
x_data = [[1, 2, 1, 1],
          [2, 1, 3, 2],
          [3, 1, 3, 4],
          [4, 1, 5, 5],
          [1, 7, 5, 5],
          [1, 2, 5, 6],
          [1, 6, 6, 6],
          [1, 7, 7, 7]]
y_data = [[0, 0, 1],
          [0, 0, 1],
          [0, 0, 1],
          [0, 1, 0],
          [0, 1, 0],
          [0, 1, 0],
          [1, 0, 0],
          [1, 0, 0]]

#convert into numpy and float format
x_data = np.asarray(x_data, dtype=np.float32)
y_data = np.asarray(y_data, dtype=np.float32)

#nb_classes
nb_classes = 3
print(x_data.shape)
print(y_data.shape)
```

```
(8, 4)
(8, 3)
```

Lab 06-1: Softmax Classification Eager

Hyopothesis에 들어갈 weight값과 bias 설정

[6]
```
W = tf.Variable(tf.random.normal((4, nb_classes)), name='weight')
b = tf.Variable(tf.random.normal((nb_classes,)), name='bias')
variables = [W, b]

print(W,b)
```

```
<tf.Variable 'weight:0' shape=(4, 3) dtype=float32, numpy=
array([[ 0.7706481 ,  0.37335402, -0.05576323],
       [ 0.00358377, -0.5898363 ,  1.5702795 ],
       [ 0.2460895 , -0.09918973,  1.4418385 ],
       [ 0.3200988 ,  0.526784  , -0.7703731 ]], dtype=float32)> <tf.Variable 'bias:0' shape=(3,) dtype=float32, numpy=array([-1.3080608 , -0.13253094,  0.5513761 ], dtype=float32)>
```

Hypothesis

[7]
```
def hypothesis(X):
    return tf.nn.softmax(tf.matmul(X, W) + b)

print(hypothesis(x_data))
```

```
tf.Tensor(
[[1.36571955e-02 7.90162291e-03 9.78441179e-01]
 [3.92597765e-02 1.70347411e-02 9.43705440e-01]
 [3.80385250e-01 1.67723149e-01 4.51891541e-01]
 [3.23390484e-01 5.90759404e-02 6.17533624e-01]
 [3.62997366e-06 6.20727221e-08 9.99996245e-01]
 [2.62520202e-02 1.07279625e-02 9.63019967e-01]
 [1.56525093e-05 4.21802724e-07 9.99983847e-01]
 [2.94076904e-06 3.81133241e-08 9.99996960e-01]], shape=(8, 3), dtype=float32)
```

Softmax funciton

[8]
```
sample_db = [[8,2,1,4]]
sample_db = np.asarray(sample_db, dtype=np.float32)


print(hypothesis(sample_db))
```

Lab 06-1: Softmax Classification Eager

Cost function

```
[9]
def cost_fn(X, Y):
    logits = hypothesis(X)
    cost = -tf.reduce_sum(Y * tf.math.log(logits), axis=1)
    cost_mean = tf.reduce_mean(cost)

    return cost_mean

print(cost_fn(x_data, y_data))
```

```
tf.Tensor(6.07932, shape=(), dtype=float32)
```

Gradient Tapee

```
[10]
x = tf.constant(3.0)
with tf.GradientTape() as g:
    g.watch(x)
    y = x * x # x^2
dy_dx = g.gradient(y, x) # Will compute to 6.0
print(dy_dx)
```

```
tf.Tensor(6.0, shape=(), dtype=float32)
```

Lab 06-1: Softmax Classification Eager

Model fitting

```
[12]
def fit(X, Y, epochs=2000, verbose=100):
    optimizer =  tf.keras.optimizers.SGD(learning_rate=0.1)

    for i in range(epochs):
        grads = grad_fn(X, Y)
        optimizer.apply_gradients(zip(grads, variables))
        if (i==0) | ((i+1)%verbose==0):
            print('Loss at epoch %d: %f' %(i+1, cost_fn(X, Y).numpy()))

fit(x_data, y_data)
```

```
Loss at epoch 1: 2.849417
Loss at epoch 100: 0.684151
Loss at epoch 200: 0.613813
Loss at epoch 300: 0.558204
Loss at epoch 400: 0.508306
Loss at epoch 500: 0.461058
Loss at epoch 600: 0.415072
Loss at epoch 700: 0.369636
Loss at epoch 800: 0.324533
Loss at epoch 900: 0.280721
Loss at epoch 1000: 0.246752
Loss at epoch 1100: 0.232798
Loss at epoch 1200: 0.221645
Loss at epoch 1300: 0.211476
Loss at epoch 1400: 0.202164
Loss at epoch 1500: 0.193606
Loss at epoch 1600: 0.185714
Loss at epoch 1700: 0.178415
Loss at epoch 1800: 0.171645
Loss at epoch 1900: 0.165351
Loss at epoch 2000: 0.159483
```

Lab 06-1: Softmax Classification Eager

Argmax를 이용한 정확도 측정

```
sample_data = [[2,1,3,2]] # answer_label [[0,0,1]]
sample_data = np.asarray(sample_data, dtype=np.float32)

a = hypothesis(sample_data)

print(a)
print(tf.argmax(a, 1)) #index: 2

b = hypothesis(x_data)
print(b)
print(tf.argmax(b, 1))
print(tf.argmax(y_data, 1)) # matches with y_data
```

```
tf.Tensor([[0.00112886 0.08154673 0.9173244 ]], shape=(1, 3), dtype=float32)
tf.Tensor([2], shape=(1,), dtype=int64)
tf.Tensor(
[[2.1976039e-06 1.2331199e-03 9.9876475e-01]
 [1.1288594e-03 8.1546724e-02 9.1732436e-01]
 [2.2205660e-07 1.6418649e-01 8.3581328e-01]
 [6.3921934e-06 8.5045439e-01 1.4953916e-01]
 [2.6150793e-01 7.2644734e-01 1.2044546e-02]
 [1.3783264e-01 8.6213988e-01 2.7417602e-05]
 [7.4242103e-01 2.5754192e-01 3.6978636e-05]
 [9.2197543e-01 7.8024052e-02 6.0005920e-07]], shape=(8, 3), dtype=float32)
tf.Tensor([2 2 2 1 1 1 0 0], shape=(8,), dtype=int64)
tf.Tensor([2 2 2 1 1 1 0 0], shape=(8,), dtype=int64)
```

Lab 06-2: Softmax Zoo Classification-Eager

> **tf.onehot()**
> → 내가 원하는 행의 개수 만큼 행렬을 변환해주는 method
> → 3차원으로 반환
> **tf.reshape()**
> → 원하는 형태의 행렬로 재배열 해준다
> → 앞서 3차원으로 반환 되었지만 2차원 형태로 model fitting을 해야하기 때문에 reshape를 사용한다

```
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, -1]

nb_classes = 7  # 0 ~ 6

# Make Y data as onehot shape
#2차원에서 3차원으로 변환
#tf.one_hot()을 쓰면 3차원으로 반환을 해주기 때문에
Y_one_hot = tf.one_hot(y_data.astype(np.int32), nb_classes)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])
print(x_data.shape, Y_one_hot.shape)
```

```
(101, 16) (101, 7)
```

Lab 06-2: Softmax Zoo Classification-Eager

tf.onehot()
→ 내가 원하는 행의 개수 만큼 행렬을 변환해주는 method
→ 3차원으로 반환
tf.reshape()
→ 원하는 형태의 행렬로 재배열 해준다
→ 앞서 3차원으로 반환 되었지만 2차원 형태로 model fitting을 해야하기 때문에 reshape를 사용한다

```
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, -1]

nb_classes = 7  # 0 ~ 6

# Make Y data as onehot shape
#2차원에서 3차원으로 변환
#tf.one_hot()을 쓰면 3차원으로 반환을 해주기 때문에
Y_one_hot = tf.one_hot(y_data.astype(np.int32), nb_classes)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])
print(x_data.shape, Y_one_hot.shape)
```

```
(101, 16) (101, 7)
```

Lab 06-2: Softmax Zoo Classification-Eager

> **Weight와 Bias**
> → 전과 같음
> **Logit Function, Hypothesis**
> → 나중에 정확도 관련 값을 구할때 필요하기 때문에 따로 정의한다
> **Cross Entropy**
> → tf.keras.losses.categorical_crossentropy() 이용
> **tf.argmax()**
> → 가장 큰값을 가지는 index를 리턴해준다
> **Prediction**
> → Accuracy를 알려주는 함수

Lab 06-2: Softmax Zoo Classification-Eager

```
#Weight and bias setting
W = tf.Variable(tf.random.normal((16, nb_classes)), name='weight')
b = tf.Variable(tf.random.normal((nb_classes,)), name='bias')
variables = [W, b]

# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)

#####logit과 hypothesis를 다르게 함
def logit_fn(X):
    return tf.matmul(X, W) + b

def hypothesis(X):
    return tf.nn.softmax(logit_fn(X))

def cost_fn(X, Y):
    logits = logit_fn(X)
    cost_i = tf.keras.losses.categorical_crossentropy(y_true=Y, y_pred=logits,
                                                      from_logits=True)
    cost = tf.reduce_mean(cost_i)
    return cost

#이전과 동일
def grad_fn(X, Y):
    with tf.GradientTape() as tape:
        loss = cost_fn(X, Y)
        grads = tape.gradient(loss, variables)
        return grads

#정확도를 나타내주는것이 추가됨
#tf.argmax 알아보기
def prediction(X, Y):
    pred = tf.argmax(hypothesis(X), 1)
    correct_prediction = tf.equal(pred, tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    return accuracy
```

Lab 06-2: Softmax Zoo Classification-Eager

[16]
```python
def fit(X, Y, epochs=1000, verbose=100):
    optimizer =  tf.keras.optimizers.SGD(learning_rate=0.1)

    for i in range(epochs):
        grads = grad_fn(X, Y)
        optimizer.apply_gradients(zip(grads, variables))
        if (i==0) | ((i+1)%verbose==0):
#             print('Loss at epoch %d: %f' %(i+1, cost_fn(X, Y).numpy()))
            acc = prediction(X, Y).numpy()
            loss = cost_fn(X, Y).numpy()
            print('Steps: {} Loss: {}, Acc: {}'.format(i+1, loss, acc))

fit(x_data, Y_one_hot)
```

```
Steps: 1 Loss: 5.610101222991943, Acc: 0.1683168262243271
Steps: 100 Loss: 0.6019306182861328, Acc: 0.8415841460227966
Steps: 200 Loss: 0.38190874457359314, Acc: 0.9108911156654358
Steps: 300 Loss: 0.2877010405063629, Acc: 0.9405940771102905
Steps: 400 Loss: 0.23101474344730377, Acc: 0.9504950642585754
Steps: 500 Loss: 0.19246312975883484, Acc: 0.9603960514068604
Steps: 600 Loss: 0.16442003846168518, Acc: 0.9603960514068604
Steps: 700 Loss: 0.14311201870441437, Acc: 0.9603960514068604
Steps: 800 Loss: 0.12642519176006317, Acc: 0.9603960514068604
Steps: 900 Loss: 0.11306187510490417, Acc: 0.9900990128517151
Steps: 1000 Loss: 0.10216663777828217, Acc: 0.9900990128517151
```

Lab 07: Learning Rate & Evaluation

> **Learning Rate**
> → Learnging Rate 값이 크면 Overshootiing 현상이 생긴다. 즉, 다음 weight 값이 최소 점보다 더 멀리 나아가 발산한다.
> → Laerning Rate 값이 작으면 시간이 오래 걸려 Overfitting이나 발산형태로 나아간다.

Lab 07: Learning Rate & Evaluation

> **Exponential Decay**
>
> - $\alpha = \alpha_0 e^{-kt}$

[21]

```
"""
0: Exponential Decay
1: Inverse Time Decay
2: Cosine Daecay
3: Piecewise Decay

"""

EPOCHS = 1001

for step in range(EPOCHS):
    for features, labels  in iter(dataset):
        features = tf.cast(features, tf.float32)
        labels = tf.cast(labels, tf.float32)
        grads = grad(softmax_fn(features), features, labels)
        optimizer = learningRate(0)

        optimizer.apply_gradients(grads_and_vars=zip(grads,[W,b]))
        if step % 100 == 0:
            print("Iter: {}, Loss: {:.4f}".format(step, loss_fn(softmax_fn(features),features,labels)))
x_test = tf.cast(x_test, tf.float32)
y_test = tf.cast(y_test, tf.float32)
test_acc = accuracy_fn(softmax_fn(x_test),y_test)
print("Testset Accuracy: {:.4f}".format(test_acc))
```

```
Iter: 0, Loss: 12.4497
Iter: 100, Loss: 0.6930
Iter: 200, Loss: 0.5960
Iter: 300, Loss: 0.5394
Iter: 400, Loss: 0.4986
Iter: 500, Loss: 0.4665
Iter: 600, Loss: 0.4401
Iter: 700, Loss: 0.4177
Iter: 800, Loss: 0.3984
Iter: 900, Loss: 0.3814
Iter: 1000, Loss: 0.3664
Testset Accuracy: 1.0000
```

Lab 07: Learning Rate & Evaluation

> **Inverse Time Decay**
> $\rightarrow \alpha = \frac{\alpha_0}{1+kt}$

[22]

```
"""
0: Exponential Decay
1: Inverse Time Decay
2: Cosine Daecay
3: Piecewise Decay

"""

EPOCHS = 1001

for step in range(EPOCHS):
    for features, labels  in iter(dataset):
        features = tf.cast(features, tf.float32)
        labels = tf.cast(labels, tf.float32)
        grads = grad(softmax_fn(features), features, labels)
        optimizer = learningRate(1)

        optimizer.apply_gradients(grads_and_vars=zip(grads,[W,b]))
        if step % 100 == 0:
            print("Iter: {}, Loss: {:.4f}".format(step, loss_fn(softmax_fn(features),features,labels)))
x_test = tf.cast(x_test, tf.float32)
y_test = tf.cast(y_test, tf.float32)
test_acc = accuracy_fn(softmax_fn(x_test),y_test)
print("Testset Accuracy: {:.4f}".format(test_acc))
```

```
Iter: 0, Loss: 0.3662
Iter: 100, Loss: 0.3528
Iter: 200, Loss: 0.3407
Iter: 300, Loss: 0.3296
Iter: 400, Loss: 0.3195
Iter: 500, Loss: 0.3102
Iter: 600, Loss: 0.3016
Iter: 700, Loss: 0.2936
Iter: 800, Loss: 0.2861
Iter: 900, Loss: 0.2792
Iter: 1000, Loss: 0.2726
Testset Accuracy: 1.0000
```

Lab 07: Learning Rate & Evaluation

Cosine Annealing

- $\alpha = \alpha^i_{min} + \frac{1}{2}(\alpha^i_{max} - \alpha^i_{min})(1 + \cos(\frac{T_{current}}{T_i}\pi))$
- $\alpha_{min}, \alpha_{max}$: 학습전 설정된 learning rate의 최대 최소값
- $T_{current}$: 현재 Epoch
- T_i: Cosine Annealing을 실행하는 주기

```
"""
0: Exponential Decay
1: Inverse Time Decay
2: Cosine Daecay
3: Piecewise Decay

"""

EPOCHS = 1001

for step in range(EPOCHS):
    for features, labels  in iter(dataset):
        features = tf.cast(features, tf.float32)
        labels = tf.cast(labels, tf.float32)
        grads = grad(softmax_fn(features), features, labels)
        optimizer = learningRate(2)

        optimizer.apply_gradients(grads_and_vars=zip(grads,[W,b]))
        if step % 100 == 0:
            print("Iter: {}, Loss: {:.4f}".format(step, loss_fn(softmax_fn(features),features,labels)))
x_test = tf.cast(x_test, tf.float32)
y_test = tf.cast(y_test, tf.float32)
test_acc = accuracy_fn(softmax_fn(x_test),y_test)
print("Testset Accuracy: {:.4f}".format(test_acc))
```

```
Iter: 0, Loss: 0.2725
Iter: 100, Loss: 0.2664
Iter: 200, Loss: 0.2605
Iter: 300, Loss: 0.2550
Iter: 400, Loss: 0.2497
Iter: 500, Loss: 0.2447
Iter: 600, Loss: 0.2399
Iter: 700, Loss: 0.2354
Iter: 800, Loss: 0.2310
Iter: 900, Loss: 0.2268
Iter: 1000, Loss: 0.2228
Testset Accuracy: 1.0000
```

Lab 07: Learning Rate & Evaluation

Piecewise Annealing

- 특정 Epoch에 도달할 때 특정 값을 learning rate로 바꾼다
- **keras.optimizers.schedules.PiecewiseConstantDecay(boundaries, values)**
- boudary와 value는 순서가 있는 객체로 선언해야함

24]

```
"""
0: Exponential Decay
1: Inverse Time Decay
2: Cosine Daecay
3: Piecewise Decay

"""

EPOCHS = 1001

for step in range(EPOCHS):
    for features, labels  in iter(dataset):
        features = tf.cast(features, tf.float32)
        labels = tf.cast(labels, tf.float32)
        grads = grad(softmax_fn(features), features, labels)
        optimizer = learningRate(3)

        optimizer.apply_gradients(grads_and_vars=zip(grads,[W,b]))
        if step % 100 == 0:
            print("Iter: {}, Loss: {:.4f}".format(step, loss_fn(softmax_fn(features),features,labels)))
x_test = tf.cast(x_test, tf.float32)
y_test = tf.cast(y_test, tf.float32)
test_acc = accuracy_fn(softmax_fn(x_test),y_test)
print("Testset Accuracy: {:.4f}".format(test_acc))
```

```
Iter: 0, Loss: 0.2224
Iter: 100, Loss: 3.9286
Iter: 200, Loss: 7.1143
Iter: 300, Loss: 5.5011
Iter: 400, Loss: 7.2813
Iter: 500, Loss: 4.4398
Iter: 600, Loss: 6.1004
Iter: 700, Loss: 1.2818
Iter: 800, Loss: 2.3291
Iter: 900, Loss: 0.1078
Iter: 1000, Loss: 0.0225
Testset Accuracy: 1.0000
```

Lab 07-2: linear regression(without min/max)

> **Normalization**
> → 데이터의 값을 0과1 사이로 만들어주는 과정
> $x_{new} = \frac{x-\mu}{\sigma}$
> **Standardization**
> → 평균과 얼마나 떨어져 있는지에 대해서 평균에 대해 정규화 하는 과정
> $x_{new} = \frac{x-x_{min}}{x_{max}-x_{min}}$

0초 [27]

```
def normalization(data):
    numerator = data - np.min(data, 0)
    denominator = np.max(data, 0) - np.min(data, 0)
    return numerator / denominator

def standardization(data):
    numerator = data - np.mean(data)
    denominator = sqrt(np.sum(data - np.mean(data))^2/np.count(data))
```

Lab 07-3: Overfitting

Overfitting
→ model이 너무 train data에 취중해서 fitting 됨

Over fitting 줄이는 방법

- train data를 많이 받는다
- feature 수를 줄인다 (차원 축소)
- feature 수를 늘린다 (hypothesis의 식을 더 많이)

Data Set and Validation
→ train data와 test data의 비율을 잘 조정해서 data를 구성해야 한다

Fine tuning
→ model을 Learning 하는 과정에서 모델의 특정 분류 방법을 고치거나 기존 모델에 또 다른 기법을 추가해서 fitting 하는 과정

Lab 07-3-2: Mnist

> **Mnist data set**
> → 0부터 9까지 모아 놓은 손글씨 data set

[28]
```
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

> **Model**

[30]
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

Model

[30]

```python
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

Optimizer, Cross Entropy

[31]

```python
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Model fitting

[32]

```python
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.2164 - accuracy: 0.9358
Epoch 2/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0971 - accuracy: 0.9701
Epoch 3/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0702 - accuracy: 0.9777
Epoch 4/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0523 - accuracy: 0.9830
Epoch 5/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0443 - accuracy: 0.9850
<keras.src.callbacks.History at 0x7fccbe823d00>
```

Evaluation

[33]

```python
model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.0773 - accuracy: 0.9780
[0.07733535766601562, 0.9779999852180481]
```

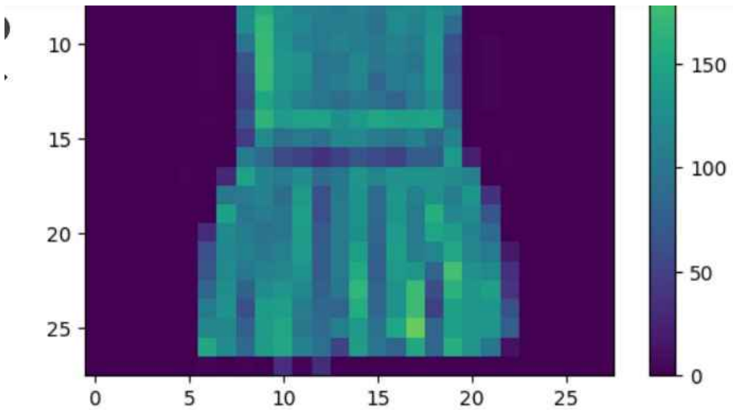
Fasion mnist

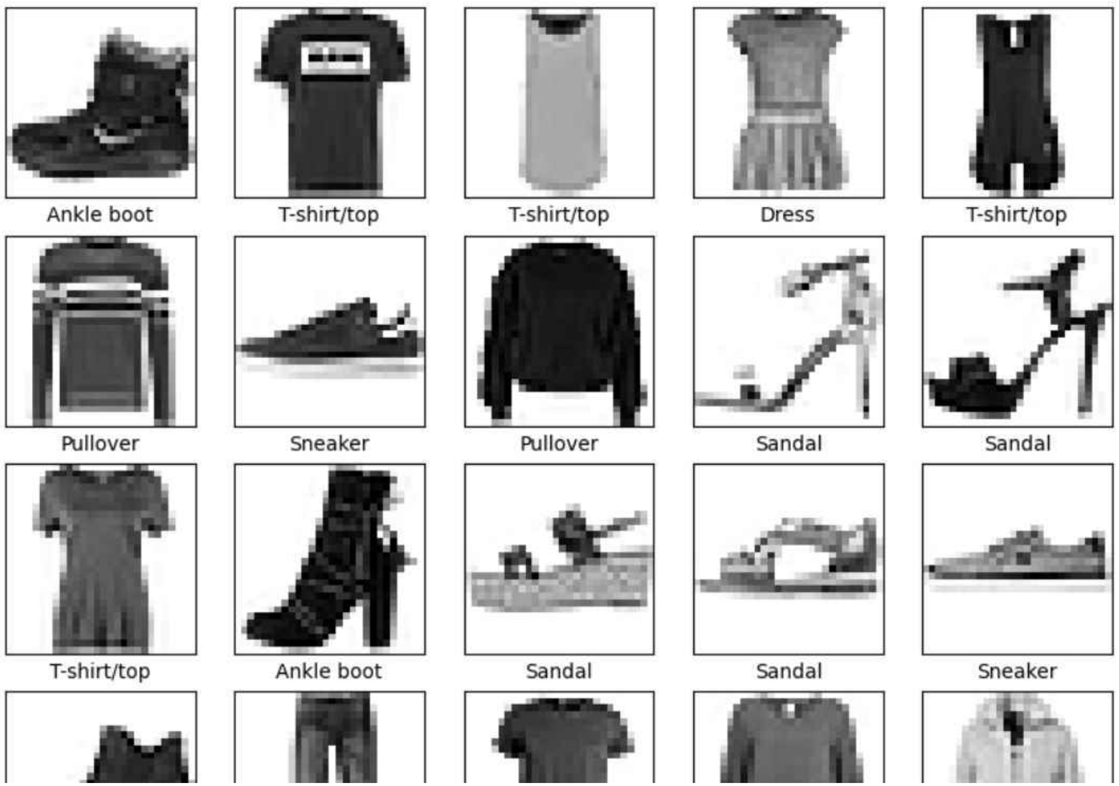
→ 10가지의 옷의 labe을 가진 data

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

```
plt.figure()
plt.imshow(train_images[3])
plt.colorbar()
plt.grid(False)
train_images = train_images / 255.0
test_images = test_images / 255.0

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
```


10
15
20
25
0
5
10
15
20
25
150
100
50
0


Ankle boot
T-shirt/top
T-shirt/top
Dress
T-shirt/top
Pullover
Sneaker
Pullover
Sandal
Sandal
T-shirt/top
Ankle boot
Sandal
Sandal
Sneaker