In [1]:
```
# Google Colab에서 노트북을 실행하실 때에는
# https://tutorials.pytorch.kr/beginner/colab 를 참고하세요.
%matplotlib inline
```

파이토치(PyTorch) 기본 익히기 || **빠른 시작** || 텐서(Tensor) || Dataset과 Dataloader || 변형(Transform) || 신경망 모델 구성하기 || Autograd || 최적화(Optimization) || 모델 저장하고 불러오기

빠른 시작(Quickstart)

이번 장에서는 기계 학습의 일반적인 작업들을 위한 API를 통해 실행됩니다. 더 자세히 알아보려면 각 장(section)의 링크를 참고하세요.

데이터 작업하기

파이토치(PyTorch)에는 데이터 작업을 위한 기본 요소 두가지인 `torch.utils.data.DataLoader` 와 `torch.utils.data.Dataset` 가 있습니다. `Dataset` 은 샘플과 정답(label)을 저장하고, `DataLoader` 는 `Dataset` 을 순회 가능한 객체(iterable)로 감쌉니다.

In [33]:
```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
import numpy as np
```

PyTorch는 TorchText, TorchVision 및 TorchAudio 와 같이 도메인 특화 라이브러리를 데이터셋과 함께 제공하고 있습니다. 이 튜토리얼에서는 TorchVision 데이터셋을 사용하도록 하겠습니다.

`torchvision.datasets` 모듈은 CIFAR, COCO 등과 같은 다양한 실제 비전(vision) 데이터에 대한 `Dataset` \ (전체 목록은 여기)\ 을 포함하고 있습니다. 이 튜토리얼에서는 FasionMNIST 데이터셋을 사용합니다. 모든 TorchVision `Dataset` 은 샘플과 정답을 각각 변경하기 위한 `transform` 과 `target_transform` 의 두 인자를 포함합니다.

In [17]:
```
# 공개 데이터셋에서 학습 데이터를 내려받습니다.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# 공개 데이터셋에서 테스트 데이터를 내려받습니다.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

```
print(test_data[0]) # 이곳 수정
```

```
(tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0118, 0.0039, 0.0000, 0.0000, 0.027
5,
          0.0000, 0.1451, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.0078, 0.000
0,
```

```
          0.1059, 0.3294, 0.0431, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.4667, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.0000, 0.000
0,
          0.3451, 0.5608, 0.4314, 0.0000, 0.0000, 0.0000, 0.0000, 0.086
3,
          0.3647, 0.4157, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0157, 0.0000, 0.207
8,
          0.5059, 0.4706, 0.5765, 0.6863, 0.6157, 0.6510, 0.5294, 0.603
9,
          0.6588, 0.5490, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0078, 0.0000, 0.0431, 0.537
3,
          0.5098, 0.5020, 0.6275, 0.6902, 0.6235, 0.6549, 0.6980, 0.584
3,
          0.5922, 0.5647, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.000
0,
          0.0078, 0.0039, 0.0000, 0.0118, 0.0000, 0.0000, 0.4510, 0.447
1,
          0.4157, 0.5373, 0.6588, 0.6000, 0.6118, 0.6471, 0.6549, 0.560
8,
          0.6157, 0.6196, 0.0431, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0118, 0.0000, 0.0000, 0.3490, 0.5451, 0.352
9,
          0.3686, 0.6000, 0.5843, 0.5137, 0.5922, 0.6627, 0.6745, 0.560
8,
          0.6235, 0.6627, 0.1882, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0078, 0.015
7,
          0.0039, 0.0000, 0.0000, 0.0000, 0.3843, 0.5333, 0.4314, 0.427
5,
          0.4314, 0.6353, 0.5294, 0.5647, 0.5843, 0.6235, 0.6549, 0.564
7,
          0.6196, 0.6627, 0.4667, 0.0000],
         [0.0000, 0.0000, 0.0078, 0.0078, 0.0039, 0.0078, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.1020, 0.4235, 0.4588, 0.3882, 0.4353, 0.458
8,
          0.5333, 0.6118, 0.5255, 0.6039, 0.6039, 0.6118, 0.6275, 0.552
9,
          0.5765, 0.6118, 0.6980, 0.0000],
         [0.0118, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.082
4,
          0.2078, 0.3608, 0.4588, 0.4353, 0.4039, 0.4510, 0.5059, 0.525
5,
          0.5608, 0.6039, 0.6471, 0.6667, 0.6039, 0.5922, 0.6039, 0.560
8,
          0.5412, 0.5882, 0.6471, 0.1686],
         [0.0000, 0.0000, 0.0902, 0.2118, 0.2549, 0.2980, 0.3333, 0.462
```

```
7,
         0.5020, 0.4824, 0.4353, 0.4431, 0.4627, 0.4980, 0.4902, 0.545
1,
         0.5216, 0.5333, 0.6275, 0.5490, 0.6078, 0.6314, 0.5647, 0.607
8,
         0.6745, 0.6314, 0.7412, 0.2431],
        [0.0000, 0.2667, 0.3686, 0.3529, 0.4353, 0.4471, 0.4353, 0.447
1,
         0.4510, 0.4980, 0.5294, 0.5333, 0.5608, 0.4941, 0.4980, 0.592
2,
         0.6039, 0.5608, 0.5804, 0.4902, 0.6353, 0.6353, 0.5647, 0.541
2,
         0.6000, 0.6353, 0.7686, 0.2275],
        [0.2745, 0.6627, 0.5059, 0.4078, 0.3843, 0.3922, 0.3686, 0.380
4,
         0.3843, 0.4000, 0.4235, 0.4157, 0.4667, 0.4706, 0.5059, 0.584
3,
         0.6118, 0.6549, 0.7451, 0.7451, 0.7686, 0.7765, 0.7765, 0.733
3,
         0.7725, 0.7412, 0.7216, 0.1412],
        [0.0627, 0.4941, 0.6706, 0.7373, 0.7373, 0.7216, 0.6706, 0.600
0,
         0.5294, 0.4706, 0.4941, 0.4980, 0.5725, 0.7255, 0.7647, 0.819
6,
         0.8157, 1.0000, 0.8196, 0.6941, 0.9608, 0.9882, 0.9843, 0.984
3,
         0.9686, 0.8627, 0.8078, 0.1922],
        [0.0000, 0.0000, 0.0000, 0.0471, 0.2627, 0.4157, 0.6431, 0.725
5,
         0.7804, 0.8235, 0.8275, 0.8235, 0.8157, 0.7451, 0.5882, 0.321
6,
         0.0314, 0.0000, 0.0000, 0.0000, 0.6980, 0.8157, 0.7373, 0.686
3,
         0.6353, 0.6196, 0.5922, 0.0431],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
```

```
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000]]]), 9)
```

`Dataset` 을 `DataLoader` 의 인자로 전달합니다. 이는 데이터셋을 순회 가능한 객체(iterable)로 감싸고, 자동화된 배치(batch), 샘플링(sampling), 섞기(shuffle) 및 다중 프로세스로 데이터 불러오기(multiprocess data loading)를 지원합니다. 여기서는 배치 크기(batch size)를 64로 정의합니다. 즉, 데이터로더(dataloader) 객체의 각 요소는 64개의 특징(feature)과 정답(label)을 묶음(batch)으로 반환합니다.

In [26]:
```
batch_size = 64

# 데이터로더를 생성합니다.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

```
Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])
Shape of y: torch.Size([64]) torch.int64
```

In [27]:
```
X[0] # normalized 되어있는 FasionMNIST 데이터 # 이곳수정
```

```
Out[27]: tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000],
                  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000],
                  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000],
                  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000],
                  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000],
                  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000],
                  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000],
                  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0118, 0.0039, 0.0000, 0.0000, 0.027
         5,
                   0.0000, 0.1451, 0.0000, 0.0000],
                  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
         0,
                   0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.0078, 0.000
         0,
```

```
          0.1059, 0.3294, 0.0431, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.4667, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.0000, 0.000
0,
          0.3451, 0.5608, 0.4314, 0.0000, 0.0000, 0.0000, 0.0000, 0.086
3,
          0.3647, 0.4157, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0157, 0.0000, 0.207
8,
          0.5059, 0.4706, 0.5765, 0.6863, 0.6157, 0.6510, 0.5294, 0.603
9,
          0.6588, 0.5490, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0078, 0.0000, 0.0431, 0.537
3,
          0.5098, 0.5020, 0.6275, 0.6902, 0.6235, 0.6549, 0.6980, 0.584
3,
          0.5922, 0.5647, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.000
0,
          0.0078, 0.0039, 0.0000, 0.0118, 0.0000, 0.0000, 0.4510, 0.447
1,
          0.4157, 0.5373, 0.6588, 0.6000, 0.6118, 0.6471, 0.6549, 0.560
8,
          0.6157, 0.6196, 0.0431, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0118, 0.0000, 0.0000, 0.3490, 0.5451, 0.352
9,
          0.3686, 0.6000, 0.5843, 0.5137, 0.5922, 0.6627, 0.6745, 0.560
8,
          0.6235, 0.6627, 0.1882, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0078, 0.015
7,
          0.0039, 0.0000, 0.0000, 0.0000, 0.3843, 0.5333, 0.4314, 0.427
5,
          0.4314, 0.6353, 0.5294, 0.5647, 0.5843, 0.6235, 0.6549, 0.564
7,
          0.6196, 0.6627, 0.4667, 0.0000],
         [0.0000, 0.0000, 0.0078, 0.0078, 0.0039, 0.0078, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.1020, 0.4235, 0.4588, 0.3882, 0.4353, 0.458
8,
          0.5333, 0.6118, 0.5255, 0.6039, 0.6039, 0.6118, 0.6275, 0.552
9,
          0.5765, 0.6118, 0.6980, 0.0000],
         [0.0118, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.082
4,
          0.2078, 0.3608, 0.4588, 0.4353, 0.4039, 0.4510, 0.5059, 0.525
5,
          0.5608, 0.6039, 0.6471, 0.6667, 0.6039, 0.5922, 0.6039, 0.560
8,
          0.5412, 0.5882, 0.6471, 0.1686],
         [0.0000, 0.0000, 0.0902, 0.2118, 0.2549, 0.2980, 0.3333, 0.462
```

```
7,
         0.5020, 0.4824, 0.4353, 0.4431, 0.4627, 0.4980, 0.4902, 0.545
1,
         0.5216, 0.5333, 0.6275, 0.5490, 0.6078, 0.6314, 0.5647, 0.607
8,
         0.6745, 0.6314, 0.7412, 0.2431],
        [0.0000, 0.2667, 0.3686, 0.3529, 0.4353, 0.4471, 0.4353, 0.447
1,
         0.4510, 0.4980, 0.5294, 0.5333, 0.5608, 0.4941, 0.4980, 0.592
2,
         0.6039, 0.5608, 0.5804, 0.4902, 0.6353, 0.6353, 0.5647, 0.541
2,
         0.6000, 0.6353, 0.7686, 0.2275],
        [0.2745, 0.6627, 0.5059, 0.4078, 0.3843, 0.3922, 0.3686, 0.380
4,
         0.3843, 0.4000, 0.4235, 0.4157, 0.4667, 0.4706, 0.5059, 0.584
3,
         0.6118, 0.6549, 0.7451, 0.7451, 0.7686, 0.7765, 0.7765, 0.733
3,
         0.7725, 0.7412, 0.7216, 0.1412],
        [0.0627, 0.4941, 0.6706, 0.7373, 0.7373, 0.7216, 0.6706, 0.600
0,
         0.5294, 0.4706, 0.4941, 0.4980, 0.5725, 0.7255, 0.7647, 0.819
6,
         0.8157, 1.0000, 0.8196, 0.6941, 0.9608, 0.9882, 0.9843, 0.984
3,
         0.9686, 0.8627, 0.8078, 0.1922],
        [0.0000, 0.0000, 0.0000, 0.0471, 0.2627, 0.4157, 0.6431, 0.725
5,
         0.7804, 0.8235, 0.8275, 0.8235, 0.8157, 0.7451, 0.5882, 0.321
6,
         0.0314, 0.0000, 0.0000, 0.0000, 0.6980, 0.8157, 0.7373, 0.686
3,
         0.6353, 0.6196, 0.5922, 0.0431],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
         0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
```

```
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000]]])
```

In [28]:
```
X[0].shape# 이곳수정
```

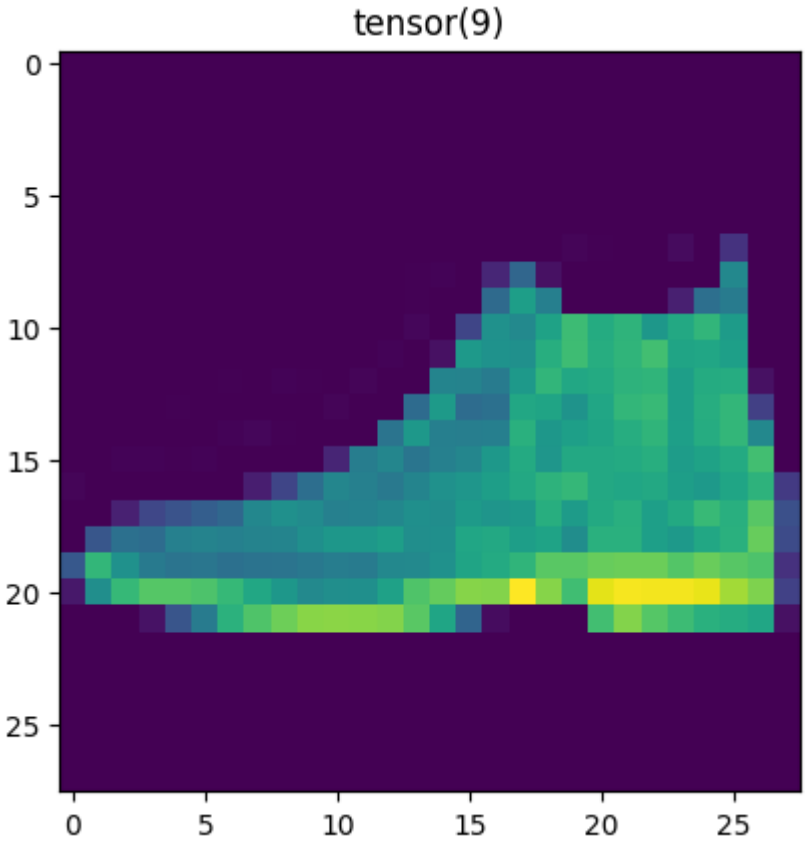
Out[28]:
```
torch.Size([1, 28, 28])
```

In [29]:
```
X_squeeze = X[0].squeeze() # 시각화 하기 편한형태로 변경 # 이곳 수정
print(X_squeeze.shape) # 형태를 보여줌
X_squeeze_numpy = X_squeeze.numpy() # tensor -> numpy
```

```
torch.Size([28, 28])
```

In [35]:
```
import matplotlib.pyplot as plt # 이곳 수정
plt.imshow(X_squeeze_numpy)
plt.title(y[0])
```

Out[35]:
```
Text(0.5, 1.0, 'tensor(9)')
```



In [34]:
```
np.unique(y.numpy())
```

Out[34]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

PyTorch에서 데이터를 불러오는 방법 을 자세히 알아보세요.

모델 만들기

PyTorch에서 신경망 모델은 nn.Module 을 상속받는 클래스(class)를 생성하여 정의합니다. `__init__` 함수에서 신경망의 계층(layer)들을 정의하고 `forward` 함수에서 신경망에 데이터를 어떻게 전달할지 지정합니다. 가능한 경우 GPU 또는 MPS로 신경망을 이동시켜 연산을 가속(accelerate)합니다.

In [38]:
```
# 학습에 사용할 CPU나 GPU, MPS 장치를 얻습니다.
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")

# 모델을 정의합니다.
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten() # image 를 fully connected layer에 넣기
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(in_features = 28*28,out_features= 512),
            nn.ReLU(), # 활성화 함수로 relu 사용
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10) # 출력값을 10개인 이유 : 10개 중 1개를 맞추는 분류도
        )
  # 신경망의 데이터를 어떻게 전달할지 정해줌
    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

```
Using cpu device
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

PyTorch에서 신경망을 정의하는 방법 을 자세히 알아보세요. -- 공식문서에서 확인해야할수도

모델 매개변수 최적화하기

모델을 학습하려면 손실 함수(loss function) 와 옵티마이저(optimizer) 가 필요합니다.

In [42]:
```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3) # SGD : Stochast
```

각 학습 단계(training loop)에서 모델은 (배치(batch)로 제공되는) 학습 데이터셋에 대한 예측을 수행하고, 예측 오류를 역전파하여 모델의 매개변수를 조정합니다.

In [43]:
```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # 예측 오류 계산
        pred = model(X)
        loss = loss_fn(pred, y) # 손실값 계산 # 주석추가

        # 역전파
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0: # 100번중에 한번만 확인 # 주석추가
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

모델이 학습하고 있는지를 확인하기 위해 테스트 데이터셋으로 모델의 성능을 확인합니다.

In [44]:
```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {t
```

학습 단계는 여러번의 반복 단계 (*에폭(epochs)*) 를 거쳐서 수행됩니다. 각 에폭에서는 모델은 더 나은 예측을 하기 위해 매개변수를 학습합니다. 각 에폭마다 모델의 정확도(accuracy)와 손실(loss)을 출력합니다; 에폭마다 정확도가 증가하고 손실이 감소하는 것을 보려고 합니다.

In [46]:
```
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
```

```
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

```
Epoch 1
-------------------------------
loss: 1.162744  [   64/60000]
loss: 1.171615  [ 6464/60000]
loss: 0.986147  [12864/60000]
loss: 1.127118  [19264/60000]
loss: 0.999958  [25664/60000]
loss: 1.029107  [32064/60000]
loss: 1.054237  [38464/60000]
loss: 0.996921  [44864/60000]
loss: 1.038713  [51264/60000]
loss: 0.977125  [57664/60000]
Test Error:
 Accuracy: 66.1%, Avg loss: 0.987355

Epoch 2
-------------------------------
loss: 1.042839  [   64/60000]
loss: 1.071808  [ 6464/60000]
loss: 0.872512  [12864/60000]
loss: 1.032909  [19264/60000]
loss: 0.910843  [25664/60000]
loss: 0.935367  [32064/60000]
loss: 0.974687  [38464/60000]
loss: 0.922625  [44864/60000]
loss: 0.959679  [51264/60000]
loss: 0.908466  [57664/60000]
Test Error:
 Accuracy: 67.5%, Avg loss: 0.913838

Epoch 3
-------------------------------
loss: 0.954968  [   64/60000]
loss: 1.002313  [ 6464/60000]
loss: 0.790906  [12864/60000]
loss: 0.965539  [19264/60000]
loss: 0.850577  [25664/60000]
loss: 0.866266  [32064/60000]
loss: 0.918963  [38464/60000]
loss: 0.872910  [44864/60000]
loss: 0.902839  [51264/60000]
loss: 0.858951  [57664/60000]
Test Error:
 Accuracy: 68.5%, Avg loss: 0.860808

Epoch 4
-------------------------------
loss: 0.887544  [   64/60000]
loss: 0.950459  [ 6464/60000]
loss: 0.730069  [12864/60000]
loss: 0.915199  [19264/60000]
loss: 0.807451  [25664/60000]
loss: 0.813708  [32064/60000]
loss: 0.876995  [38464/60000]
loss: 0.838226  [44864/60000]
loss: 0.860632  [51264/60000]
loss: 0.821353  [57664/60000]
Test Error:
 Accuracy: 69.5%, Avg loss: 0.820667
```

```
Epoch 5
-------------------------------
loss: 0.834031  [   64/60000]
loss: 0.909106  [ 6464/60000]
loss: 0.682788  [12864/60000]
loss: 0.876214  [19264/60000]
loss: 0.774821  [25664/60000]
loss: 0.772999  [32064/60000]
loss: 0.843194  [38464/60000]
loss: 0.812501  [44864/60000]
loss: 0.828210  [51264/60000]
loss: 0.791382  [57664/60000]
Test Error:
 Accuracy: 70.5%, Avg loss: 0.788893

Done!
```

모델을 학습하는 방법 을 자세히 알아보세요.

모델 저장하기

모델을 저장하는 일반적인 방법은 (모델의 매개변수들을 포함하여) 내부 상태 사전(internal state dictionary)을 직렬화(serialize)하는 것입니다.

In [47]:
```
torch.save(model.state_dict(), "model.pth")
print("Saved PyTorch Model State to model.pth")
```

```
Saved PyTorch Model State to model.pth
```

모델 불러오기

모델을 불러오는 과정에는 모델 구조를 다시 만들고 상태 사전을 모델에 불러오는 과정이 포함됩니다.

In [48]:
```
model = NeuralNetwork().to(device)
model.load_state_dict(torch.load("model.pth"))
```

Out[48]:
```
<All keys matched successfully>
```

이제 이 모델을 사용해서 예측을 할 수 있습니다.

In [52]:
```
classes = [
    "T-shirt/top",
    "Trouser",
    "Pullover",
    "Dress",
    "Coat",
    "Sandal",
    "Shirt",
    "Sneaker",
    "Bag",
    "Ankle boot",
]

model.eval()
```

```
x, y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    x = x.to(device)
    pred = model(x)
    predicted, actual = classes[pred[0].argmax(0)], classes[y]
    print(f'Predicted: "{predicted}", Actual: "{actual}"')
```

```
Predicted: "Ankle boot", Actual: "Ankle boot"
```

모델을 저장하고 불러오는 방법 을 자세히 알아보세요.