본 영상 교재는
2025년도 과학기술정보통신부 및 정보통신기획평가원의
SW중심대학 사업의 지원을 받아 제작되었습니다.

한국항공대학교
Korea Aerospace University

KAU
AI융합대학

SW중심대학

PyTorch 코딩

Layer Sequence

Layer Sequence

01

Layer Sequence

- Layer Sequence
  - 직렬로 연결된 layer들을 하나의 인스턴스로 묶어주는 클래스
    - 성능 차이는 없지만 코딩이 비교적 편하며, 코딩 실수로 인한 에러 발생 확률을 크게 줄여줄 수 있음
  - 개별 layer를 인스턴스 변수로 선언하지 않고, 클래스 이름만으로 layer를 추가할 수 있음
    - ANN의 forward pass를 정의할 때 layer sequence를 통째로 사용할 수 있기 때문에 코딩이 매우 간단해짐
  - Parameter가 있든 없든 상관없이 모든 layer들을 반드시 명시해야 함
    - 코딩 중 layer를 빼먹을 가능성을 줄여줌

Layer Sequence

- Layer Sequence 실습
  - Cell 4-1 : ANN parameter 초기화 함수 정의하기

```
> class NeuralNetwork(nn.Module):
>     def parameter_initializer(self, layer):                    # ANN parameter를 초기화하는 함수를 정의
>         if hasattr(layer, 'weight') and hasattr(layer, 'bias'):
>             torch.nn.init.xavier_normal_(layer.weight)
>             torch.nn.init.zeros_(layer.bias)
```

Layer Sequence

- Layer Sequence 실습
  - Cell 4-2 : Layer sequence를 통해 layer 추가 및 parameter 초기화하기

```
> class NeuralNetwork(nn.Module):
>     ...
>
>     def __init__(self):
>         super().__init__()
>
>         self.linear_stack = nn.Sequential(              # 직렬로 연결된 layer들을 하나의 인스턴스로 묶을 수 있음
>             nn.Flatten(start_dim = 1),                  # 모든 layer와 연산을 명시해야 함
>
>             nn.Linear(28 * 28, 512),                    # 클래스 이름을 통해 layer을 추가
>             nn.ReLU(),
>
>             nn.Linear(512, 512),
>             nn.ReLU(),
>
>             nn.Linear(512, 10),
>             nn.Softmax(dim = 1)
>         )
>
>         for module in self.modules():
>             module.apply(self.parameter_initializer)    # apply()를 통해 각 구성요소에 특정 함수 적용
>
>     ...
```

Layer Sequence

- Layer Sequence 실습
  - Cell 4-3 : Layer sequence를 통해 ANN feed-forward 작업 정의하기

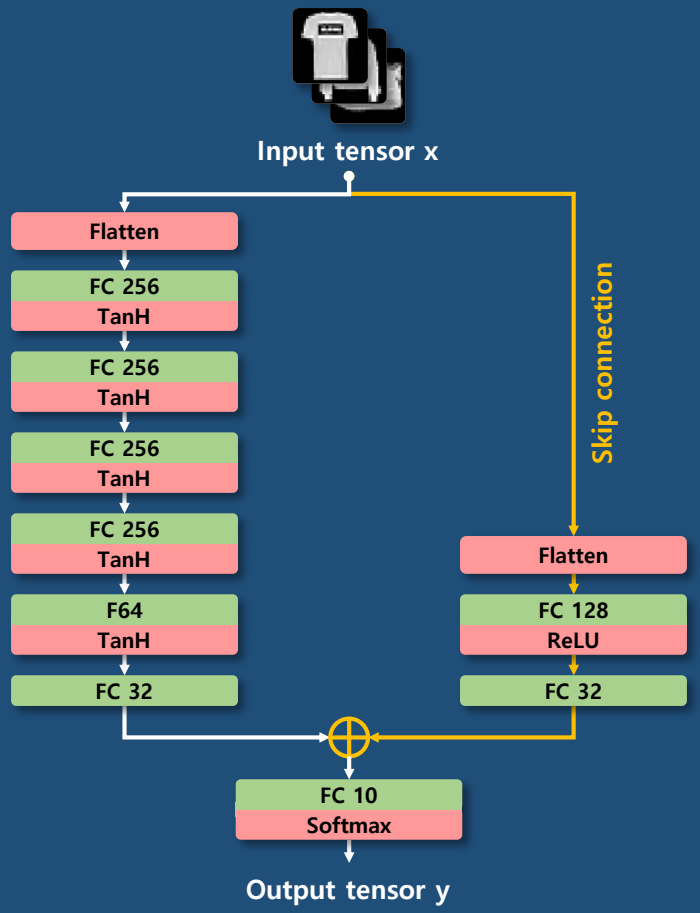
```
> class NeuralNetwork(nn.Module):
>     ...
>
>     def forward(self, x):
>         y = self.linear_stack(x) # 일반 layer와 같은 방법으로 layer sequence의 feed-forward 호출 가능
>
>         return y
```

02

Skip-connection

Skip-connection

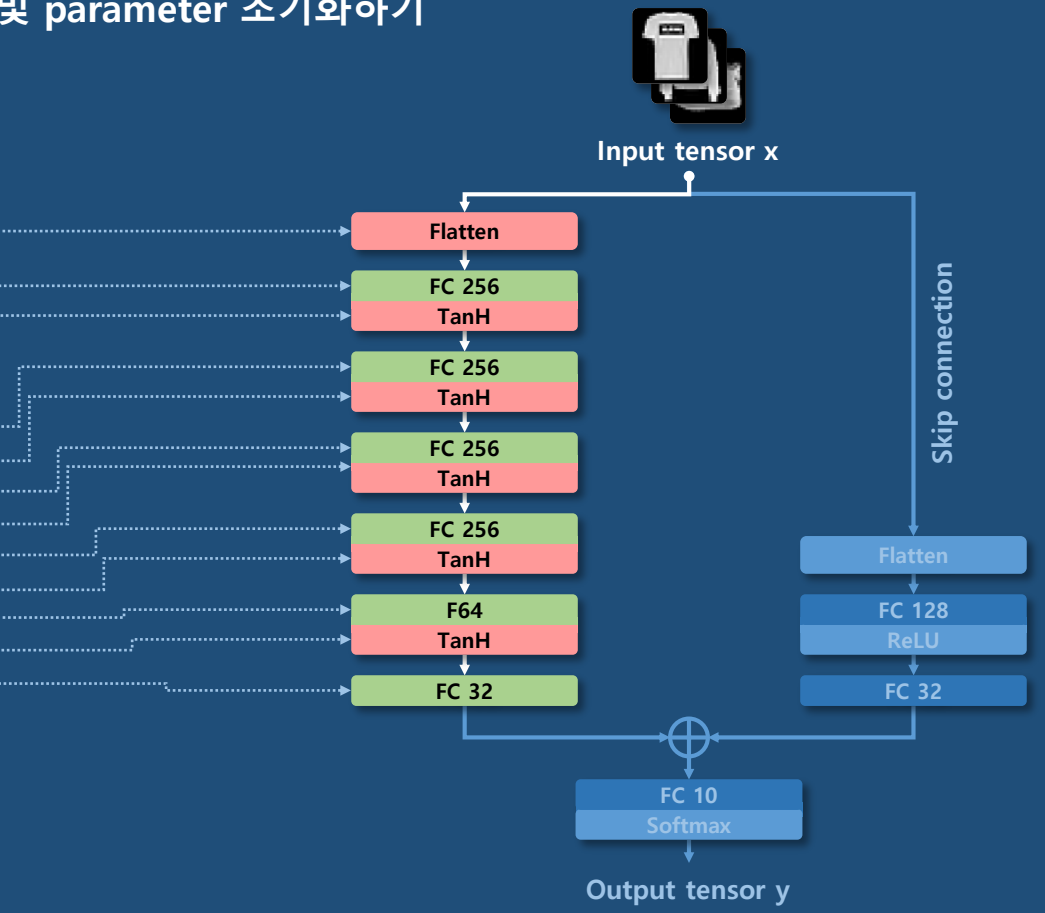
- Skip-connection 실습
  - 실습할 ANN architecture



Skip-connection

- Skip-connection 실습
  - Cell 4-1 : Layer sequence를 통해 layer 추가 및 parameter 초기화하기

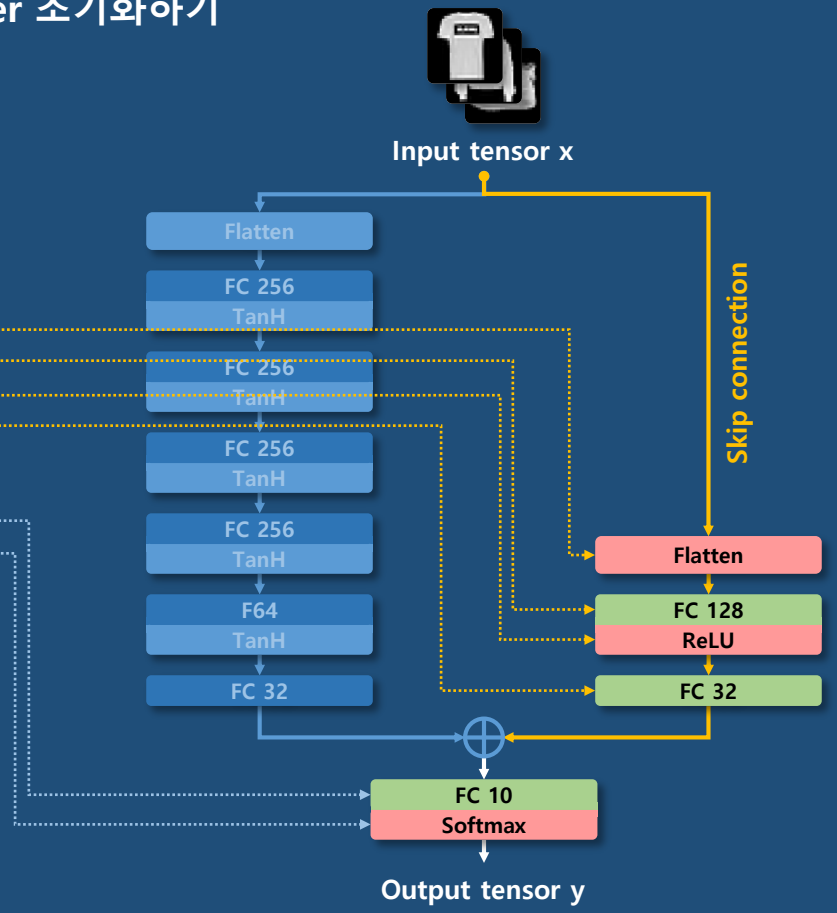
```
class NeuralNetwork(nn.Module):
    ...

    def __init__(self):
        super().__init__()

        self.linear_stack = nn.Sequential(
            nn.Flatten(start_dim = 1),
            nn.Linear(28 * 28, 256),
            nn.Tanh(),
            nn.Linear(256, 256),
            nn.Tanh(),
            nn.Linear(256, 256),
            nn.Tanh(),
            nn.Linear(256, 256),
            nn.Tanh(),
            nn.Linear(256, 64),
            nn.Tanh(),
            nn.Linear(64, 32),
        )

        ...
```



Skip-connection

- Skip-connection 실습
  - Cell 4-2 : Layer sequence를 통해 layer 추가 및 parameter 초기화하기

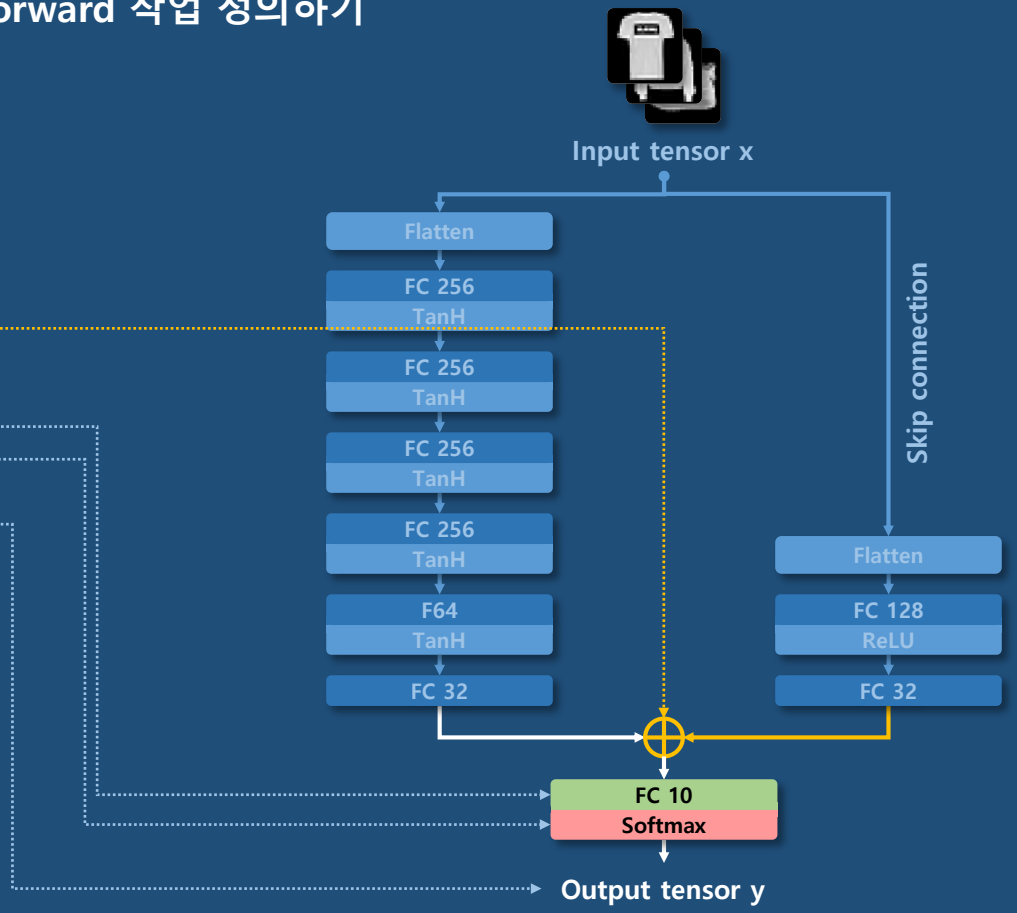
```
class NeuralNetwork(nn.Module):
    ...

    def __init__(self):
        ...

        self.skip_connection_stack = nn.Sequential(
            nn.Flatten(start_dim = 1),
            nn.Linear(28 * 28, 128),
            nn.ReLU(),
            nn.Linear(128, 32)
        )

        self.last_linear = nn.Linear(32, 10)
        self.softmax = nn.Softmax(dim = 1)
        # Parameter가 없는 layer도 인스턴스 변수로 선언 가능

        for module in self.modules():
            module.apply(self.parameter_initializer)

    ...
```



Skip-connection

- **Skip-connection 실습**
  - Cell 4-3 : Layer sequence를 통해 ANN feed-forward 작업 정의하기

```
> class NeuralNetwork(nn.Module):
>     ...
>
>     def forward(self, x):
>         x1 = self.linear_stack(x)
>         x2 = self.skip_connection_stack(x)
>
>         x = x1 + x2
>         # x1 + x2 대신 torch.add(x1, x2)를 사용해도 됨
>
>         x = self.last_linear(x)
>         y = self.softmax(x)
>
>         return y
```



Skip-connection

- Skip-connection 실습
  - Cell 4-4 : Learning rate 수정하기

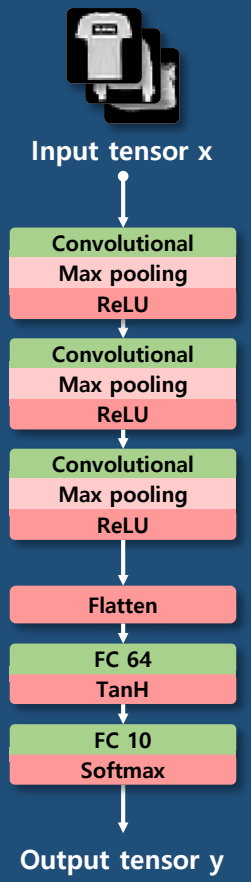
```
> model = NeuralNetwork().to(device)

> optimizer = torch.optim.SGD(
>     model.parameters(),
>     lr = 1.5e-3 # ANN architecture가 달라졌으므로 learning rate를 변경해야 할 수 있음
> )
```

03

CNN

CNN

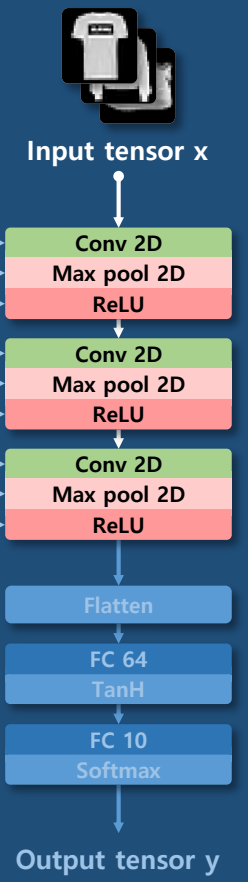
- CNN 실습
  - 실습할 ANN architecture



CNN

- CNN 실습
  - Cell 4-1 : Layer sequence를 통해 layer 추가 및 parameter 초기화하기

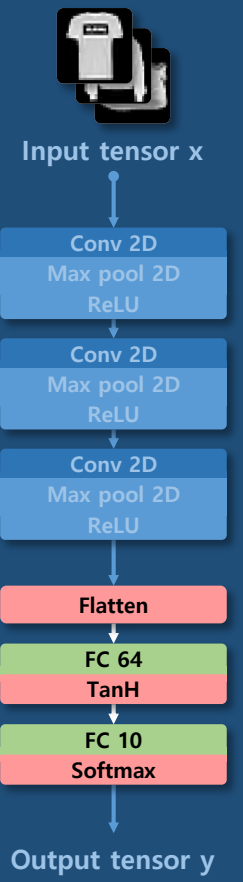
```
> class NeuralNetwork(nn.Module):
>     ...
>
>     def __init__(self):
>         super().__init__()
>
>         self.cnn_stack = nn.Sequential(
>             nn.Conv2d(in_channels = 1, out_channels = 4, kernel_size = 5, padding = 2, padding_mode = 'zeros'),
>             # 각 filter가 2D인 convoultional layer
>             nn.MaxPool2d(kernel_size = 2, stride = 2),
>             # 각 filter가 2D인 max pooling layer
>             nn.ReLU(),
>
>             nn.Conv2d(in_channels = 4, out_channels = 8, kernel_size = 3, padding = 1, padding_mode = 'zeros'),
>             nn.MaxPool2d(kernel_size = 2, stride = 2),
>             nn.ReLU(),
>
>             nn.Conv2d(in_channels = 8, out_channels = 16, kernel_size = 3, padding = 1, padding_mode = 'zeros'),
>             nn.MaxPool2d(kernel_size = 2, stride = 2),
>             nn.ReLU()
>         )
>
>         ...
```



CNN

- CNN 실습
  - Cell 4-2 : Layer sequence를 통해 layer 추가 및 parameter 초기화하기

```
class NeuralNetwork(nn.Module):
    ...

    def __init__(self):
        ...

        self.linear_stack = nn.Sequential(
            nn.Flatten(start_dim = 1),
            # CNN의 output은 [N, C, H, W]이므로, linear layer로 전달하기 위해 flatten 작업 수행

            nn.Linear(16 * 3 * 3, 64),
            # Linear layer는 input 데이터의 dimension을 알려주어야 하므로, CNN stack의 output dimension을 알아야 함
            nn.Tanh(),

            nn.Linear(64, 10),
            nn.Softmax(dim = 1)
        )

        for module in self.modules():
            module.apply(self.parameter_initializer)

    ...
```



CNN

- CNN 실습
  - Cell 4-3 : Layer sequence를 통해 ANN feed-forward 작업 정의하기

```
> class NeuralNetwork(nn.Module):
>     ...
>
>     def forward(self, x):
>         x = self.cnn_stack(x)
>         # print(f'Shape of CNN stack output: {x.shape}') # 확인용 임시 코드, cnn_stack의 output dimension을 확인할 수 있음
>
>         y = self.linear_stack(x)
>
>         return y
```

CNN

- CNN 실습
  - Cell 5 : Optimizer를 SGD에서 ADAM으로 변환하기

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

> optimizer = torch.optim.Adam(    # Training 효율을 높이기 위해 SGD optimizer 대신 ADAM optimizer를 사용
>     model.parameters(),
>     lr = 2e-3,                   # 고급 optimizer를 사용할 경우 training을 빠르게 하기 위해 learning rate를 높여도 training이 덜 불안정해짐
>     betas = (0.9, 0.999),        # Adam optimizer의 β1과 β2
>     weight_decay = 1e-4          # Weight decay 추가
> )
```

Have a nice day!

Fin.