```
%matplotlib inline


import os
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms


device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")
```

```
Using cpu device
```

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits


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

```
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)
  )
)
```

```
X = torch.rand(1, 28, 28, device=device)
logits = model(X)
pred_probab = nn.Softmax(dim=1)(logits)
y_pred = pred_probab.argmax(1)
print(f"Predicted class: {y_pred}")
```

```
Predicted class: tensor([8])
```

```
input_image = torch.rand(3,28,28)
print(input_image.size())
```

```
torch.Size([3, 28, 28])
```

```
flatten = nn.Flatten()
flat_image = flatten(input_image)
print(flat_image.size())
```

```
torch.Size([3, 784])
```

```
layer1 = nn.Linear(in_features=28*28, out_features=20)
hidden1 = layer1(flat_image)
```

```
print(hidden1.size())
```

```
torch.Size([3, 20])
```

```
print(f"Before ReLU: {hidden1}\n\n")
hidden1 = nn.ReLU()(hidden1)
print(f"After ReLU: {hidden1}")
```

```
Before ReLU: tensor([[-0.3362, -0.2820,  0.1018,  0.1758, -0.3598, -0.1138,  0.5141, -0.1963,
         -0.1018,  0.1037, -0.1464,  0.0781,  0.5090,  0.6621,  0.2732,  0.0755,
         -0.0545,  0.1207,  0.1699, -0.1700],
        [-0.2119,  0.1427,  0.2940,  0.0260, -0.2607, -0.3363,  0.3615, -0.1519,
          0.1163,  0.5876, -0.4799, -0.0172,  0.2416,  0.3134,  0.0739,  0.3432,
         -0.0935,  0.0860,  0.1761,  0.0128],
        [-0.3495, -0.0891, -0.2611, -0.1765,  0.1090,  0.0046, -0.0695,  0.2460,
          0.1014,  0.5830, -0.1319,  0.4513,  0.4053,  0.3169,  0.0466,  0.4235,
         -0.4497,  0.2829,  0.0742, -0.1336]], grad_fn=<AddmmBackward0>)


After ReLU: tensor([[0.0000, 0.0000, 0.1018, 0.1758, 0.0000, 0.0000, 0.5141, 0.0000, 0.0000,
         0.1037, 0.0000, 0.0781, 0.5090, 0.6621, 0.2732, 0.0755, 0.0000, 0.1207,
         0.1699, 0.0000],
        [0.0000, 0.1427, 0.2940, 0.0260, 0.0000, 0.0000, 0.3615, 0.0000, 0.1163,
         0.5876, 0.0000, 0.0000, 0.2416, 0.3134, 0.0739, 0.3432, 0.0000, 0.0860,
         0.1761, 0.0128],
        [0.0000, 0.0000, 0.0000, 0.0000, 0.1090, 0.0046, 0.0000, 0.2460, 0.1014,
         0.5830, 0.0000, 0.4513, 0.4053, 0.3169, 0.0466, 0.4235, 0.0000, 0.2829,
         0.0742, 0.0000]], grad_fn=<ReluBackward0>)
```

```
seq_modules = nn.Sequential(
    flatten,
    layer1,
    nn.ReLU(),
    nn.Linear(20, 10)
)
input_image = torch.rand(3,28,28)
logits = seq_modules(input_image)
```

```
softmax = nn.Softmax(dim=1)
pred_probab = softmax(logits)
```

```
print(f"Model structure: {model}\n\n")

for name, param in model.named_parameters():
    print(f"Layer: {name} | Size: {param.size()} | Values : {param[:2]} \n")
```

```
Model structure: 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)
  )
)


Layer: linear_relu_stack.0.weight | Size: torch.Size([512, 784]) | Values : tensor([[-0.0128,  0.0231, -0.0146,  ..., -0.0224,  0.0032,
        [ 0.0074,  0.0245, -0.0008,  ..., -0.0154,  0.0305,  0.0044]],
       grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.0.bias | Size: torch.Size([512]) | Values : tensor([0.0247, 0.0353], grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.2.weight | Size: torch.Size([512, 512]) | Values : tensor([[ 0.0242, -0.0226,  0.0034,  ..., -0.0416, -0.0334,
        [ 0.0249,  0.0086, -0.0189,  ...,  0.0117, -0.0311, -0.0285]],
       grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.2.bias | Size: torch.Size([512]) | Values : tensor([0.0197, 0.0191], grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.4.weight | Size: torch.Size([10, 512]) | Values : tensor([[ 0.0401, -0.0307,  0.0164,  ...,  0.0173, -0.0238, -
        [-0.0168,  0.0243,  0.0371,  ...,  0.0160,  0.0272,  0.0248]],
       grad_fn=<SliceBackward0>)

Layer: linear_relu_stack.4.bias | Size: torch.Size([10]) | Values : tensor([0.0219, 0.0290], grad_fn=<SliceBackward0>)
```