使用预训练的模型做预测

除了用测试集验证模型效果外，同时还查看模型中各层的权重、偏置，预测数据时候各神经元的激活值

In [1]:
```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt
```

载入训练数据和测试数据，用于预测和验证效果

In [2]:
```
# Download training data from open datasets.
training_data = datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)

# Download test data from open datasets.
test_data = datasets.MNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
```

载入预训练的模型 Loading Models

The process for loading a model includes re-creating the model structure and loading the state dictionary into it.

In [3]:
```
# Get cpu, gpu or mps device for training.
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")

# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 64),
            nn.ReLU(),
```

```
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, 10)
        )

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

model = NeuralNetwork().to(device)
model.load_state_dict(torch.load("model_predict.pth"))
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=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=10, bias=True)
  )
)
```

查看模型中各层的权重和偏置

In [4]:
```
# 获取第一层隐藏的权重和偏置
first_hidden_layer_weights = model.linear_relu_stack[0].weight.data
first_hidden_layer_bias = model.linear_relu_stack[0].bias.data

# 获取第二隐藏层（实际上是第二个线性层，因为ReLU不是参数层）的权重和偏置
second_hidden_layer_weights = model.linear_relu_stack[2].weight.data
second_hidden_layer_bias = model.linear_relu_stack[2].bias.data

# 获取输出层的权重和偏置
output_layer_weights = model.linear_relu_stack[4].weight.data
output_layer_bias = model.linear_relu_stack[4].bias.data

# 打印权重和偏置
print("第一隐藏层权重: ", first_hidden_layer_weights)
print("第一隐藏层偏置: ", first_hidden_layer_bias)

print("第二隐藏层权重: ", second_hidden_layer_weights)
print("第二隐藏层偏置: ", second_hidden_layer_bias)

print("输出层权重: ", output_layer_weights)
print("输出层偏置: ", output_layer_bias)
```

```
第一隐藏层权重:  tensor([[-0.0195,  0.0220, -0.0221,  ..., -0.0306,  0.0055,
0.0120],
        [-0.0280, -0.0122,  0.0229,  ..., -0.0307,  0.0062, -0.0096],
        [-0.0044, -0.0342,  0.0217,  ..., -0.0033, -0.0328,  0.0178],
        ...,
        [ 0.0209, -0.0292,  0.0077,  ..., -0.0334,  0.0252,  0.0041],
        [ 0.0046,  0.0154,  0.0139,  ...,  0.0330,  0.0071, -0.0024],
        [-0.0209,  0.0154, -0.0035,  ..., -0.0239, -0.0010, -0.0057]])
第一隐藏层偏置:  tensor([ 0.0296,  0.0425,  0.0893,  0.0680,  0.0476, -0.010
2, -0.0141,  0.0039,
         0.1179, -0.0051, -0.0063,  0.1402,  0.0305,  0.0803, -0.1300, -0.
0338,
         0.0416,  0.0299, -0.0043,  0.0756,  0.0449,  0.0931, -0.0292,  0.
1133,
         0.0456,  0.0377, -0.0042,  0.0035,  0.0477,  0.0054, -0.0119, -0.
0474,
         0.1227,  0.1388, -0.0281,  0.0813,  0.0178,  0.0552,  0.0490,  0.
0072,
        -0.0164,  0.0316, -0.0669,  0.1000,  0.0881,  0.0919, -0.0064, -0.
0218,
         0.0813,  0.0588,  0.0158, -0.0199,  0.0130, -0.0466,  0.0100, -0.
1247,
         0.1114,  0.1044, -0.0122, -0.0034,  0.0785, -0.0152, -0.0402, -0.
0415])
第二隐藏层权重:  tensor([[-0.0563, -0.0641, -0.0325,  ...,  0.0980,  0.1307,
-0.0628],
        [ 0.0474, -0.0067, -0.0182,  ...,  0.0804,  0.0941,  0.0752],
        [ 0.1841,  0.1053, -0.0109,  ..., -0.0493, -0.1440, -0.0285],
        ...,
        [ 0.2437,  0.3966, -0.3448,  ..., -0.1102, -0.0115,  0.0628],
        [-0.1370,  0.0867,  0.1750,  ...,  0.0465,  0.1278, -0.1213],
        [ 0.0573, -0.0602, -0.1162,  ..., -0.0504, -0.0274,  0.1009]])
第二隐藏层偏置:  tensor([ 0.0613, -0.0556,  0.2170, -0.0331,  0.0087,  0.003
8, -0.0242,  0.1194,
        -0.1149,  0.0857,  0.2528, -0.0365, -0.1136,  0.2322, -0.1338,  0.
0782,
        -0.0475,  0.1012,  0.2293, -0.0125,  0.1650,  0.0996,  0.1835, -0.
0772,
        -0.0919, -0.0349, -0.0830,  0.1514,  0.1181, -0.0754,  0.0235,  0.
1931,
        -0.0602, -0.0858,  0.0199,  0.1739, -0.0891, -0.0564, -0.0505,  0.
0408,
        -0.0315,  0.2024, -0.1083,  0.1851,  0.1893, -0.0402,  0.1810,  0.
1339,
        -0.0885, -0.0338,  0.0215, -0.0398, -0.0293, -0.1736,  0.1572,  0.
0027,
         0.0381,  0.0937,  0.0077, -0.0526,  0.0223, -0.2086,  0.0360, -0.
0940])
输出层权重:  tensor([[-3.2534e-03,  4.4505e-02, -1.4822e-01,  2.4159e-01, -
1.2649e-01,
         -2.6954e-02, -1.2039e-01,  2.4011e-01, -5.0230e-02,  4.4404e-01,
         -1.7705e-01,  9.6470e-04, -4.4233e-02, -1.6926e-02, -1.5379e-01,
          6.0385e-01, -1.8699e-01, -1.2985e-01, -2.3855e-01, -9.9534e-02,
          1.7219e-01,  6.8281e-02,  2.4858e-01, -1.0956e-01, -2.5428e-01,
         -1.2136e-01, -7.9913e-02, -8.0732e-02, -6.8158e-02,  8.3840e-02,
          1.1004e-02, -2.2385e-01, -3.3946e-02, -2.3403e-02, -1.4913e-01,
          1.8774e-01,  1.4440e-01, -1.5498e-01, -3.0777e-02, -5.3725e-01,
         -1.9087e-01, -2.3805e-01,  8.8907e-02, -1.4613e-01, -4.6100e-01,
         -1.0540e-02, -3.6497e-01, -3.3924e-01, -1.3423e-01, -1.1152e-01,
          8.3122e-02,  4.0271e-02, -1.7198e-01,  3.0428e-01,  2.8901e-01,
```

```
 -4.5310e-02,  1.0313e-01, -3.8627e-01,  3.7437e-02, -9.2067e-02,
  1.7127e-02,  4.5757e-02,  3.2498e-01,  1.9280e-01],
[ 2.0171e-01, -6.3532e-02, -2.7162e-01, -3.2064e-01, -6.6036e-02,
  4.4056e-02,  8.5121e-02, -3.7619e-01,  2.3227e-01, -8.1479e-02,
  3.3319e-01,  6.3291e-02, -6.0949e-02, -3.2472e-01,  5.9351e-02,
 -3.6042e-01, -3.5502e-02,  4.1703e-01,  1.8142e-01,  9.3860e-02,
 -1.4318e-01, -3.2246e-01, -1.9541e-01, -2.7804e-02,  1.1065e-01,
 -4.5328e-01, -9.8576e-02,  1.1114e-01, -1.3846e-01, -9.9673e-02,
 -8.2081e-02, -1.2492e-01, -6.3587e-02, -4.6172e-02, -1.6278e-01,
 -3.0023e-01, -2.2194e-01,  1.4163e-01,  2.6828e-01, -2.3857e-01,
  2.9525e-01,  6.7499e-02, -2.0892e-01,  3.4872e-01,  4.1419e-01,
 -1.0368e-01,  3.6993e-01,  1.8787e-01, -8.5072e-02,  2.2975e-01,
 -1.1250e-01,  2.5252e-01, -2.3388e-01, -1.2828e-01, -2.4120e-01,
  3.6731e-02,  1.6173e-01,  2.3248e-01,  1.0876e-01,  1.1014e-01,
 -3.6151e-02, -3.0127e-01,  1.6899e-01, -6.7070e-03],
[ 3.8320e-01, -1.0321e-01,  2.8060e-01, -4.1505e-01, -2.5645e-01,
  4.5169e-03, -4.6993e-02,  1.1033e-01,  3.6727e-01,  5.7644e-01,
  3.6159e-01, -1.6587e-01,  5.8019e-02,  1.3508e-01,  8.3435e-02,
  1.4098e-01,  2.8439e-01, -9.1504e-02, -2.7552e-02, -4.0286e-02,
  3.0576e-02,  5.0453e-01, -3.4170e-01,  5.9626e-02,  1.8237e-01,
  2.3294e-01, -9.3465e-02, -1.2417e-01, -3.5868e-01,  1.8332e-01,
  2.0099e-02,  3.7433e-01,  2.5299e-01, -1.0416e-01,  9.4507e-02,
  1.1438e-01,  3.3790e-02, -2.1978e-01, -2.2887e-01, -4.5903e-01,
 -3.4145e-01,  2.1213e-01,  2.2357e-01, -2.3670e-01,  1.7929e-01,
  7.4415e-02,  2.1814e-01, -7.1893e-03, -5.6927e-02,  2.1083e-01,
  1.7630e-01, -3.8970e-01, -1.9292e-01,  1.4985e-01, -4.3684e-01,
  3.7205e-02, -4.4985e-03,  1.1139e-01,  6.8341e-02,  2.6335e-02,
 -7.9717e-02,  5.3038e-02, -1.7264e-02,  1.0311e-01],
[-3.3298e-02,  3.6847e-02,  1.2413e-01,  5.6815e-02,  2.1413e-01,
 -8.0569e-02,  9.4141e-02, -2.1005e-01, -1.9460e-01, -1.9385e-01,
  1.7768e-01, -7.4433e-03, -9.8034e-02, -2.5197e-01,  2.2168e-01,
 -2.6661e-01,  6.5067e-01, -2.6710e-01,  2.7202e-01,  1.1038e-01,
 -3.5876e-01,  4.0973e-01,  3.9825e-01,  5.7796e-02,  1.0287e-01,
 -2.8792e-02,  3.0798e-02, -4.8476e-01,  1.6726e-01,  5.2262e-02,
 -1.2328e-03,  2.4525e-01,  1.6591e-01,  1.1681e-02,  5.2996e-01,
 -3.5220e-01, -2.0780e-01, -1.4655e-01, -9.0882e-02, -1.5820e-01,
  3.8702e-02, -1.5174e-01, -1.3915e-01, -3.5281e-01,  9.7171e-02,
  1.0420e-01, -2.5475e-01,  4.0237e-01, -7.3397e-05,  1.8596e-01,
 -1.0872e-01, -2.2287e-02,  1.9498e-02,  3.3368e-01, -1.4127e-01,
 -4.9131e-02, -7.1912e-02,  7.5584e-02, -1.1097e-01,  8.2389e-02,
 -7.9110e-02, -5.4226e-02,  3.0383e-01,  9.5168e-02],
[-2.7174e-01, -6.0858e-02, -1.1668e-01,  1.4214e-01,  4.9544e-01,
 -1.8088e-01, -6.4259e-02,  8.6089e-02, -5.5138e-02, -3.8636e-01,
 -1.6012e-01, -2.8535e-01,  4.1197e-02,  4.2602e-01,  1.1938e-01,
 -1.9848e-01, -9.9495e-02, -3.1655e-01,  1.8054e-01, -1.5455e-01,
 -4.2432e-02, -9.9304e-02, -2.6316e-01, -4.2428e-02,  8.3318e-02,
  2.9243e-01, -5.5171e-02,  2.5630e-01, -7.3522e-02,  5.5028e-02,
  6.8864e-02,  4.6163e-02,  1.3852e-01,  1.1987e-01, -1.8589e-01,
  4.1227e-01,  2.6068e-01, -4.3629e-02,  3.2813e-02,  3.7530e-01,
 -5.5932e-02,  4.4997e-01, -9.6690e-02,  2.1775e-01, -2.9025e-01,
  6.4579e-02, -1.0133e-01, -4.2879e-02,  2.3668e-02, -2.6720e-01,
  8.9956e-02,  4.4819e-01, -1.2212e-01, -3.9132e-01,  3.3778e-01,
 -9.1861e-02, -1.6772e-02, -3.0941e-01,  1.1886e-01,  1.6580e-02,
 -1.7925e-01, -4.1634e-01, -4.4012e-01, -1.2461e-01],
[-2.1715e-01,  1.2612e-01, -3.6664e-01,  1.5954e-01, -8.6591e-02,
  6.8688e-02, -3.8124e-02, -7.5023e-02, -3.2290e-01, -2.9748e-01,
  3.0393e-01, -1.3169e-01, -1.0475e-01, -3.0851e-02, -5.7899e-01,
  4.5265e-02,  1.6344e-01, -1.8032e-03,  1.6184e-01,  1.7447e-01,
  3.5641e-01,  8.2133e-02,  5.2564e-01,  2.3814e-02, -1.4072e-01,
 -1.5218e-01, -3.8827e-02,  2.2737e-01,  3.0797e-01,  6.1185e-02,
```

```
         6.5692e-02,  4.1137e-02, -3.0609e-02, -6.0616e-02, -1.1429e-01,
        -1.5159e-03, -3.5504e-01,  2.5456e-02,  2.2572e-01,  5.3370e-01,
        -2.7497e-01,  4.1086e-03,  4.5417e-02,  3.4347e-01,  1.7628e-01,
         7.8889e-02,  2.6389e-01, -2.5010e-01,  9.3335e-03,  8.0410e-02,
         4.3514e-03, -5.4185e-01,  2.4519e-01, -2.2344e-01,  3.8413e-01,
         3.2326e-01,  4.9847e-01,  1.7476e-02, -9.2366e-02, -1.7134e-02,
         2.7122e-01, -3.3946e-01,  2.9873e-02, -2.9647e-01],
       [ 6.0249e-02,  3.2855e-02,  5.2725e-02, -5.2597e-02, -1.2173e-01,
        -6.2448e-02, -2.9586e-02,  5.7109e-01,  2.1707e-02, -3.9100e-02,
        -1.8319e-01, -2.2985e-01,  7.5416e-02, -9.6401e-02,  8.1204e-02,
        -2.8611e-01, -2.3584e-01,  4.3281e-01, -3.4452e-01, -1.5902e-02,
        -1.8226e-01, -2.6503e-02, -3.4252e-01, -1.0255e-01, -3.0912e-01,
         7.3996e-03,  2.9054e-02,  5.4371e-01, -7.7536e-03, -9.8615e-02,
         2.9574e-02,  9.5454e-02,  7.9828e-03, -6.3567e-02, -2.7764e-01,
         6.4536e-01,  1.6058e-01,  4.9164e-02,  6.1685e-03, -2.2573e-01,
         2.8772e-01, -3.6104e-01, -3.1368e-01, -1.7835e-01, -4.9930e-01,
        -3.7238e-02,  1.9227e-01, -8.6436e-02,  9.3179e-02, -2.4657e-01,
        -3.6651e-02,  5.2397e-02, -3.5650e-02, -1.4690e-02,  1.2823e-01,
        -1.3939e-01, -8.6983e-02,  2.8421e-01,  5.5515e-03, -9.3632e-02,
         2.7255e-02, -3.7870e-01,  2.2368e-01, -1.3699e-01],
       [ 9.6791e-02, -2.1330e-02,  6.0455e-01, -3.3274e-02, -2.6985e-01,
         1.3725e-01,  3.3156e-02, -4.3107e-01,  2.7519e-01, -1.0872e-01,
        -6.6034e-02,  4.3771e-01,  5.2302e-02,  3.6360e-01,  2.4051e-01,
        -7.1833e-03, -1.9193e-02, -2.6637e-01, -1.3272e-01, -1.4235e-01,
        -5.2381e-02, -1.8105e-03, -4.5024e-02, -3.3265e-02, -1.1282e-01,
        -4.6341e-01, -9.9563e-02, -5.3478e-01,  6.9335e-02, -3.0484e-02,
         1.1696e-01,  1.4089e-01, -2.4266e-02,  3.1204e-02,  1.8141e-01,
         3.7548e-02,  3.7497e-01,  2.0594e-01,  2.2532e-02, -6.1480e-01,
         1.9315e-01,  3.8511e-01,  2.3545e-01,  2.5432e-01,  4.5443e-01,
         1.8848e-03, -4.9501e-03, -3.7095e-02,  5.5358e-02, -4.9929e-03,
        -2.9617e-02, -1.5727e-01,  8.3810e-03, -1.6816e-01,  3.3996e-02,
         1.8393e-01, -1.1376e-01,  1.8416e-01,  1.3123e-01, -6.6831e-02,
         6.3870e-03,  2.6138e-01, -6.0686e-01,  6.7701e-02],
       [ 8.8876e-02,  5.7941e-02, -3.5016e-01,  5.6710e-02, -3.1699e-02,
        -2.1507e-02,  3.1147e-02, -2.0155e-01,  1.9689e-01, -2.9615e-02,
        -2.0760e-01,  1.0260e-01, -1.1345e-01, -4.2585e-01,  5.8115e-02,
         1.0710e-01, -1.6641e-01, -9.2973e-03, -2.0675e-01, -1.3524e-01,
         1.6469e-01, -8.3255e-02, -1.0276e-01,  1.1316e-01,  1.3246e-01,
         3.9875e-01, -9.6859e-02,  1.3079e-01, -5.3444e-01, -2.2976e-01,
         4.5999e-02, -1.6478e-01, -2.6280e-01, -4.2939e-03, -1.9105e-01,
        -2.0289e-01,  1.0972e-01,  9.6983e-02, -1.1824e-01,  7.1798e-01,
         1.2519e-02, -1.2863e-02,  2.6970e-01, -3.6262e-01, -1.1144e-01,
         2.1813e-02,  5.1548e-02, -2.3445e-01,  3.7384e-02, -4.1903e-02,
         9.1773e-02,  1.5017e-01,  4.2428e-01,  1.4058e-01, -2.1198e-01,
        -4.0034e-01, -3.9371e-01, -7.9639e-02,  8.5924e-02, -5.7498e-02,
        -2.1454e-01,  4.0797e-01,  3.7963e-01,  1.7491e-01],
       [-4.0443e-01,  6.2075e-02, -2.4405e-02, -7.3128e-02,  1.5121e-02,
         2.8670e-01, -5.8839e-02,  3.8615e-02, -3.9365e-01, -1.1051e-01,
        -3.5494e-01,  3.2904e-01,  5.7171e-02,  4.5948e-01, -1.8030e-01,
        -2.1163e-01, -2.8878e-01, -5.1174e-02,  3.3773e-01,  1.6718e-01,
         1.4922e-01, -3.8947e-01,  9.2142e-02,  3.8681e-02,  1.2134e-01,
         4.3410e-01,  5.1291e-02,  2.7071e-01,  4.5115e-01, -2.6494e-02,
         1.8779e-02, -2.3331e-01, -3.8408e-01, -1.0890e-01, -8.0748e-02,
        -1.9154e-01, -1.3291e-01,  6.6808e-02,  1.6974e-01,  9.4202e-02,
         1.2804e-01, -3.3656e-01,  3.3414e-02, -2.5138e-01,  4.3936e-02,
        -1.3312e-02, -4.1785e-01,  1.2110e-01, -1.6208e-01, -1.7973e-01,
         5.4233e-02,  3.3847e-01,  8.0048e-02, -1.6679e-01,  8.8433e-03,
        -3.1725e-01, -1.8351e-01, -4.1031e-01,  1.3961e-02,  1.2478e-01,
         2.6132e-01,  5.3243e-01, -2.6031e-01,  3.0358e-01]])
输出层偏置:  tensor([-0.1774,  0.0530,  0.0688, -0.0728,  0.0332,  0.2327, -
```

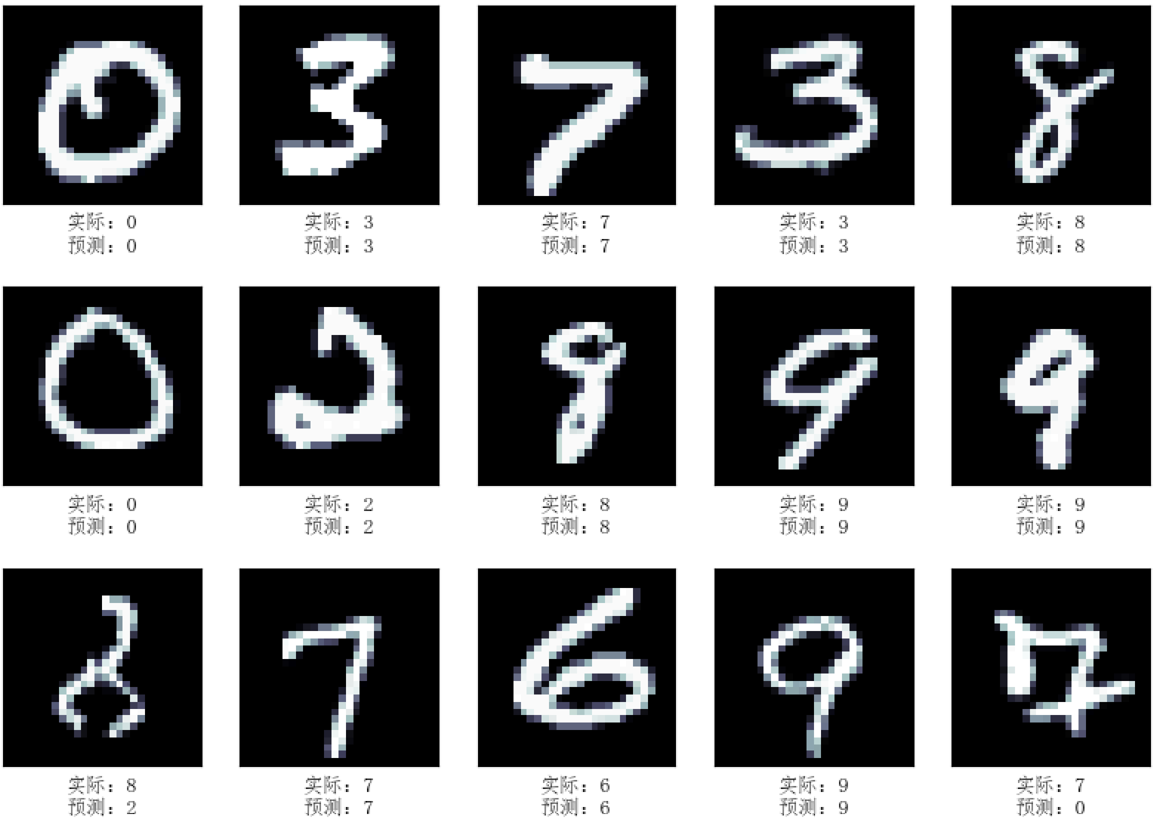
```
0.0102,  0.2033,
        -0.2708,  0.0733])
```

模型现在可以用来做预测，载入测试集中，随选3*5 个样本，观察一下预测值和实际值是否相符。

This model can now be used to make predictions.

In [5]:
```python
model.eval()

with torch.no_grad():
    fig, ax = plt.subplots(3, 5, figsize=(10, 7))  # 创建一个3行5列的画布
    for i, axi in enumerate(ax.flat):
        t = int(torch.randint(low=0, high=10000, size=(1, 1))[0][0])  # 生
        x, y = test_data[t][0], test_data[t][1]
        x = x.to(device)
        pred = model(x)
        pred = model(x)
        predicted, actual = pred[0].argmax(0), y
        axi.imshow(x.reshape(28, 28), cmap="bone")  # 绘制图像
        axi.set(xticks=[], yticks=[])
        axi.set_xlabel(f"实际: {actual}\n预测: {predicted}")
    plt.rcParams["font.sans-serif"] = "FangSong"
    plt.show()
```



用数据集中的第一个样本预测一下，看看预测后，各层各神经元的激活值

In [6]:
```python
x, y = test_data[0][0], test_data[0][1]
with torch.no_grad():
    x = x.to(device)
    pred = model(x)
```

```python
    predicted, actual = pred[0].argmax(0), y
    print(f'预测值: "{predicted}", 实际值: "{actual}"')
```

```
预测值: "7", 实际值: "7"
```

In [7]:

```python
activation_values = []
# 定义一个函数来获取并存储 激活值
def get_activations(model, x):
    for name, module in model.named_modules():
        # print(f"name: {name}, module: {module}")
        if isinstance(module, nn.Linear):  # 若模块是线性层（全连接层）
            x = module(x)
            activation_values.append(x.detach().numpy())  # 存储激活值
            x = torch.relu(x)  # 假设使用ReLU激活函数
# 调用函数进行前向传播并收集激活值
get_activations(model, x.reshape(-1))

# 输出隐藏层的激活值
for i, activation in enumerate(activation_values):
    print(f"第{i}层的激活值:")
    print(activation)
```

```
第0层的激活值:
[ 0.46497762  1.8669338  -0.9129058   1.2389984   2.2626479   0.07512591
 -0.38309968  0.54399186  1.4423015   0.536261    1.9764924   1.5057961
  0.23844558  0.38114756  0.40221643  0.46987668  0.27146715  1.274637
 -0.49883738  0.24865481  0.79552007  1.075719    0.7903518   1.2697514
  1.852175   -1.1983551  -0.16103308  1.8612244   0.05633884  0.5130293
  1.3756769   2.2858562   1.0699114   1.1052482   0.5681926   0.7645401
 -0.39304185 -0.6281291   0.8571925   1.2891697   0.22259882 -0.06213266
  1.3097606   0.4480669   1.6160623   0.85603905  2.4784517   1.2688293
  1.36515     0.3124885   1.6299437  -0.1479131   0.8402942   0.8894825
  2.7024019   1.1377822  -0.46651977  0.8129088   0.43705362 -0.15855323
  1.9431658  -0.24175374  0.06677328  0.03472799]
第1层的激活值:
[ 0.48188904 -1.0580878   3.2078369   2.501559    0.49949443  0.8256546
 -0.43552938 -1.1672809   0.11953129  2.1901674   1.0923786   3.5640512
 -0.8554372   1.7905514   1.5281311   3.458475    0.941286   -1.0150945
  1.8858632  -0.25153688  1.1031982   2.232267    3.8299809  -0.3758601
  1.4895217   0.6212081  -1.2349597  -2.008957    1.5738604   0.01556862
 -0.8178652  -0.6000197  -0.928039   -0.38120666  2.8808067  -0.16733965
  2.5220704   1.1508791  -0.23738976  0.1319632   0.75648177  3.0055974
  2.9861972   0.90724164  3.9145277  -0.6313552  -1.418997    1.6966313
  0.55710846  1.3501097  -0.21065494 -0.23896389  2.5194836   1.7318821
  1.7161682   1.4969379   0.24422005 -0.00675168  0.17934725 -1.5237654
  0.5739541   4.1640496   1.1814394   1.136941  ]
第2层的激活值:
[  0.28270984  -5.242114     3.0610437    4.1804614   -5.8833737
   0.4093417  -11.402834     9.624073    -0.79080784   1.6843071 ]
```

In [ ]: