Mutliple Classification MNIST(GPU)

October 5, 2025

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

[2]:
```
# torch MNIST dataset got issue. Download manually ....

# train_data = datasets.MNIST(
#     root = "mnistdataset/",
#     train = True,
#     download = True,
#     # transform = transforms.ToTensor()
# )

# test_data = datasets.MNIST(
#     root = "mnistdataset/",
#     train = False,
#     download = True,
#     # transform = transforms.ToTensor()
# )
```

[3]:
```
train_data = datasets.ImageFolder(
    root = "archive/MNIST_dataset/train",
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Grayscale(num_output_channels=1)
    ])
)

test_data = datasets.ImageFolder(
    root = "archive/MNIST_dataset/test",
    transform = transforms.Compose([
        transforms.ToTensor(),
```

```
        transforms.Grayscale(num_output_channels=1)
    ])
)
```

[4]:
```
train_data,test_data
```

[4]:
```
(Dataset ImageFolder
     Number of datapoints: 56000
     Root location: archive/MNIST_dataset/train
     StandardTransform
 Transform: Compose(
                ToTensor()
                Grayscale(num_output_channels=1)
            ),
 Dataset ImageFolder
     Number of datapoints: 14000
     Root location: archive/MNIST_dataset/test
     StandardTransform
 Transform: Compose(
                ToTensor()
                Grayscale(num_output_channels=1)
            ))
```

[5]:
```
train_data[0]
```

[5]:
```
(tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0000, 0.0000],
          [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
           0.0000, 0.0000, 0.0039, 0.0274, 0.0274, 0.0274, 0.0274, 0.0274,
```

```
 0.0274, 0.0196, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0431,
 0.5372, 0.5372, 0.6078, 0.9960, 0.9960, 0.9960, 0.9960, 0.9960,
 0.9960, 0.8509, 0.3725, 0.0314, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.3490,
 0.9960, 0.9960, 0.9960, 0.9960, 0.9960, 0.9960, 0.9960, 0.9960,
 0.9960, 0.9960, 0.9960, 0.8391, 0.3725, 0.0745, 0.0078, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0784, 0.8195,
 0.9960, 0.9960, 0.8627, 0.4117, 0.4117, 0.4117, 0.4117, 0.4117,
 0.7960, 0.8979, 0.9332, 0.9960, 0.9960, 0.9960, 0.4000, 0.0078,
 0.0000, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.5372, 0.9960,
 0.9960, 0.9215, 0.1412, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.1333, 0.5646, 0.9215, 0.9960, 0.9960, 0.6274,
 0.0157, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.6078, 0.9960,
 0.9960, 0.6352, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1412, 0.8038, 0.9881, 0.9960,
 0.0196, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1098, 0.9960, 0.9960,
 0.9960, 0.7921, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.8470, 0.9960,
 0.4274, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.8470, 0.9960, 0.9960,
 0.9293, 0.1216, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.4392, 0.9960,
 0.9960, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.9999, 0.9960, 0.9960,
 0.8979, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7882, 0.9960,
 0.8901, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.9999, 0.9960, 0.9960,
 0.8979, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.4274, 0.9803, 0.9960,
 0.3882, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.9999, 0.9960, 0.9960,
 0.8979, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.1059, 0.9293, 0.9960, 0.9960,
 0.0196, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7646, 0.9960, 0.9960,
 0.8979, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
 0.0000, 0.0000, 0.0000, 0.0000, 0.5372, 0.9960, 0.9960, 0.8587,
 0.0157, 0.0000, 0.0000, 0.0000],
[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.6078, 0.9960, 0.9960,
```

```
0.9764, 0.3137, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.1176, 0.9176, 0.9960, 0.9960, 0.3764,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.3451, 0.9960, 0.9960,
          0.9960, 0.3882, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.1176, 0.5843, 0.9960, 0.9960, 0.8470, 0.0314,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0039, 0.1843, 0.9960,
          0.9960, 0.8038, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.2980, 0.8234, 0.9960, 0.9960, 0.8431, 0.1137, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0078, 0.4235,
          0.9960, 0.9215, 0.3333, 0.0000, 0.0000, 0.0000, 0.0000, 0.1059,
          0.5372, 0.9764, 0.9960, 0.9960, 0.8431, 0.1176, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0431,
          0.8274, 0.9960, 0.9842, 0.5764, 0.4117, 0.5254, 0.9058, 0.9293,
          0.9960, 0.9960, 0.9960, 0.8431, 0.1176, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.3294, 0.6117, 0.9960, 0.9960, 0.9960, 0.9960, 0.9960, 0.9960,
          0.9960, 0.9960, 0.4627, 0.0471, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0667, 0.5333, 0.8391, 0.9960, 0.9960, 0.8823, 0.5333,
          0.3921, 0.0471, 0.0078, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0157, 0.0196, 0.0196, 0.0157, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
          0.0000, 0.0000, 0.0000, 0.0000]]]),
 0)
```

[6]:

```
img, label = train_data[0]
train_data.classes
```

[6]:
```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

[7]:
```
class_names = train_data.classes
class_names[label]
```

[7]:
```
'0'
```

[8]:
```
img.shape
```

[8]:
```
torch.Size([1, 28, 28])
```

[9]:
```
BATCH_SIZE = 32

train_dataloader = DataLoader(train_data,batch_size = BATCH_SIZE,shuffle = True)
test_dataloader = DataLoader(test_data,batch_size = BATCH_SIZE,shuffle = False)
```

[10]:
```
x_first_batch, y_first_batch = next(iter(train_dataloader))
x_first_batch.shape,y_first_batch.shape
```

[10]:
```
(torch.Size([32, 1, 28, 28]), torch.Size([32]))
```

[11]:
```
x_first_batch[0].shape
```

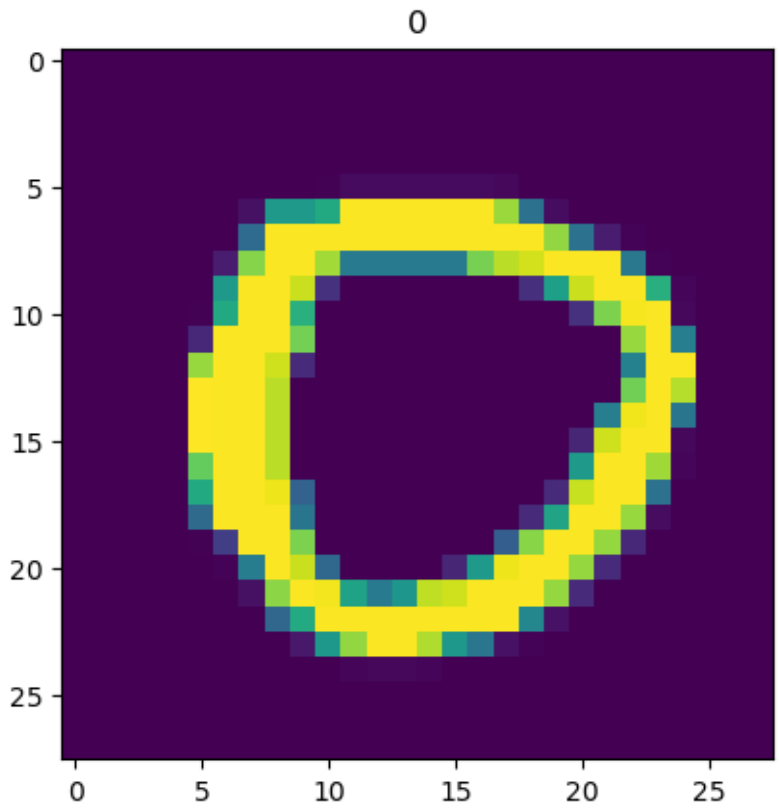
[11]:
```
torch.Size([1, 28, 28])
```

[12]:
```
# img = x_first_batch[0] / 2 + 0.5   # from [-1,1] back to [0,1]
img = img.permute(1,2,0)
img.shape
```

[12]:
```
torch.Size([28, 28, 1])
```

[13]:
```
plt.imshow(img) #reshape to torch.Size([32, 32, 3])
plt.title(class_names[label])
```

[13]:
```
Text(0.5, 1.0, '0')
```

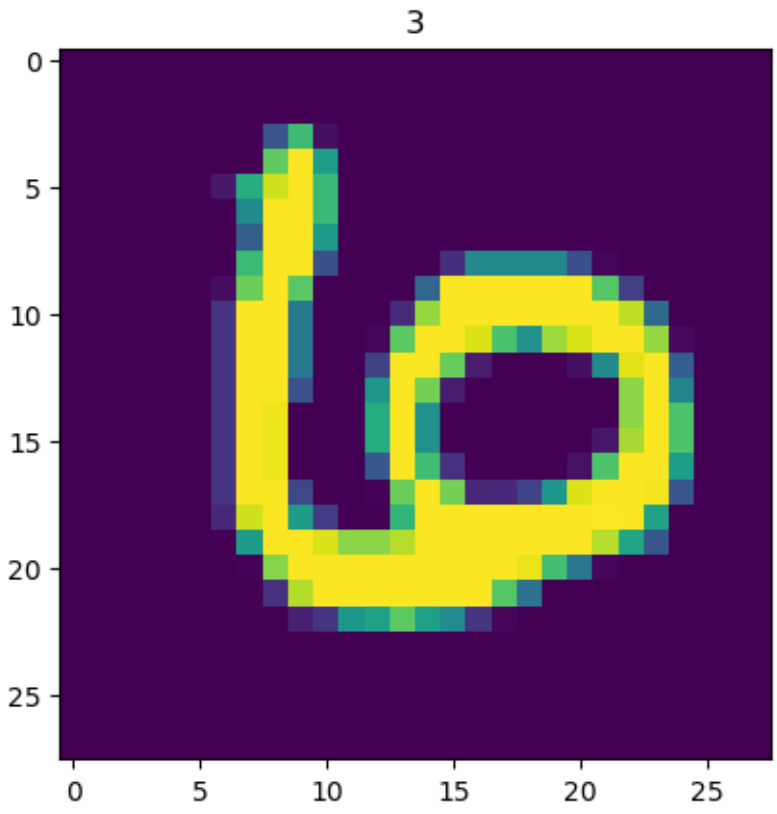


0.1 Random Pick

[14]:
```
random_img = random.randint(0, len(x_first_batch))
rand_img, rand_label = x_first_batch[random_img], y_first_batch[random_img]

plt.imshow(rand_img.permute(1,2,0))
plt.title(class_names[y_first_batch[rand_label]])
```

[14]:
```
Text(0.5, 1.0, '3')
```



0.2 Flatten

[15]:
```
#reference: https://docs.pytorch.org/docs/stable/generated/torch.nn.Flatten.
↪html#torch.nn.Flatten
```

[16]:
```
x_first_batch.shape
```

[16]:
```
torch.Size([32, 1, 28, 28])
```

[17]:
```
x_first_batch[0].shape
```

[17]:
```
torch.Size([1, 28, 28])
```

[18]:
```
f = nn.Flatten(start_dim=0, end_dim=-1) # Default is nn.Flatten(start_dim=1,␣
↪end_dim=-1)
f(x_first_batch[0]).shape
```

[18]:
```
torch.Size([784])
```

0.3 Model

[19]:
```
class ImageClassificationModel(nn.Module):
    def __init__(self,input_shape,output_shape):
        super().__init__()
        self.layer_stack = nn.Sequential(
            nn.Flatten(start_dim = 1, end_dim = -1),  # x_first_batch.shape = torch.Size([32, 3, 32, 32]) should pick(3,32,32). Thus, change start_dim to 1
            nn.Linear(in_features = input_shape, out_features = output_shape),
            nn.Softmax(dim = 1) # model(x_first_batch).shape = torch.Size([32,10]), should pick 10. Thus, change to dim = 1
        )

    def forward(self, x):
        return self.layer_stack(x)
```

[20]:
```
torch.manual_seed(87)
model = ImageClassificationModel(784,10)
model(x_first_batch)
```

[20]:
```
tensor([[0.0755, 0.0989, 0.1432, 0.0917, 0.1034, 0.1240, 0.0762, 0.0920, 0.1017,
         0.0934],
        [0.1212, 0.0897, 0.1176, 0.0908, 0.1043, 0.1102, 0.0804, 0.1048, 0.1191,
         0.0617],
        [0.1102, 0.1156, 0.0914, 0.0934, 0.0834, 0.1135, 0.0876, 0.1236, 0.1075,
         0.0738],
        [0.0859, 0.1108, 0.1019, 0.0910, 0.1123, 0.1090, 0.0899, 0.1135, 0.0983,
         0.0876],
        [0.1232, 0.0848, 0.1070, 0.1023, 0.0905, 0.1348, 0.0838, 0.1003, 0.1117,
         0.0616],
        [0.0907, 0.1024, 0.0964, 0.1045, 0.0982, 0.1134, 0.0924, 0.1160, 0.0975,
         0.0884],
        [0.0789, 0.0871, 0.1231, 0.1106, 0.0996, 0.1519, 0.0684, 0.1029, 0.1169,
         0.0605],
        [0.0971, 0.1136, 0.1138, 0.1080, 0.1365, 0.0982, 0.0636, 0.1193, 0.0864,
         0.0632],
        [0.1030, 0.0809, 0.1198, 0.1220, 0.1136, 0.1101, 0.0951, 0.0903, 0.0974,
         0.0677],
        [0.0983, 0.0752, 0.1264, 0.1248, 0.1039, 0.1020, 0.0989, 0.0937, 0.1022,
         0.0747],
        [0.0921, 0.0890, 0.1203, 0.1100, 0.1224, 0.1288, 0.0747, 0.0862, 0.1003,
         0.0762],
        [0.1321, 0.0967, 0.1166, 0.0994, 0.0845, 0.0948, 0.0928, 0.1113, 0.1051,
         0.0668],
        [0.1031, 0.1265, 0.1222, 0.0931, 0.1071, 0.0954, 0.0875, 0.0860, 0.0960,
         0.0832],
        [0.0895, 0.0979, 0.1182, 0.0962, 0.1128, 0.1060, 0.0844, 0.1145, 0.0921,
```

```
         0.0885],
        [0.0770, 0.0872, 0.0938, 0.1571, 0.1332, 0.1230, 0.0705, 0.1132, 0.0813,
         0.0637],
        [0.0863, 0.0878, 0.1042, 0.1198, 0.0830, 0.1558, 0.0704, 0.1312, 0.1049,
         0.0566],
        [0.0948, 0.0661, 0.1218, 0.1136, 0.0815, 0.1158, 0.0876, 0.1474, 0.1063,
         0.0652],
        [0.0709, 0.1070, 0.1213, 0.1135, 0.1128, 0.1211, 0.0853, 0.1089, 0.0834,
         0.0759],
        [0.1093, 0.0746, 0.1289, 0.1002, 0.1171, 0.0910, 0.1085, 0.0906, 0.1005,
         0.0792],
        [0.0990, 0.1170, 0.1038, 0.1066, 0.1061, 0.0994, 0.0728, 0.0961, 0.1171,
         0.0821],
        [0.1137, 0.1092, 0.1024, 0.0954, 0.0916, 0.1143, 0.0916, 0.1029, 0.1122,
         0.0667],
        [0.0979, 0.0865, 0.0955, 0.1356, 0.1212, 0.1223, 0.0643, 0.1112, 0.1011,
         0.0644],
        [0.0890, 0.0815, 0.1047, 0.1256, 0.1234, 0.1307, 0.0814, 0.1166, 0.0844,
         0.0627],
        [0.1011, 0.1119, 0.0983, 0.1007, 0.1006, 0.1136, 0.0876, 0.1003, 0.1008,
         0.0852],
        [0.1160, 0.0666, 0.1184, 0.1067, 0.1358, 0.0934, 0.1081, 0.0828, 0.0852,
         0.0869],
        [0.1004, 0.0946, 0.1192, 0.0958, 0.1006, 0.1269, 0.0980, 0.0966, 0.0986,
         0.0691],
        [0.1240, 0.0928, 0.1103, 0.0976, 0.0910, 0.0984, 0.0982, 0.1031, 0.1040,
         0.0806],
        [0.1200, 0.0864, 0.1229, 0.0841, 0.1092, 0.1001, 0.0890, 0.1209, 0.0860,
         0.0816],
        [0.0878, 0.0759, 0.1069, 0.1563, 0.1120, 0.1087, 0.0909, 0.1190, 0.0773,
         0.0653],
        [0.0857, 0.1150, 0.1062, 0.0935, 0.1134, 0.0931, 0.1026, 0.0974, 0.1009,
         0.0923],
        [0.1314, 0.1103, 0.1088, 0.0779, 0.1198, 0.0933, 0.0843, 0.1016, 0.0959,
         0.0766],
        [0.0805, 0.1076, 0.1037, 0.1102, 0.1047, 0.1115, 0.0985, 0.1082, 0.0950,
         0.0801]], grad_fn=<SoftmaxBackward0>)
```

[21]:
```
# softmax
# reference: https://docs.pytorch.org/docs/stable/generated/torch.nn.Softmax.
  ↪html#torch.nn.Softmax
```

[22]:
```
y_pred = model(x_first_batch)
y_pred.argmax(dim = 1)
```

[22]:
```
tensor([2, 0, 7, 7, 5, 7, 5, 4, 3, 2, 5, 0, 1, 2, 3, 5, 7, 2, 2, 8, 5, 3, 5, 5,
        4, 5, 0, 2, 3, 1, 0, 5])
```

[23]:
```
y_pred.shape
```

[23]:
```
torch.Size([32, 10])
```

[24]:
```
sm = nn.Softmax(dim = 1)
sm(y_pred).sum(dim = 1) #check softmax
```

[24]:
```
tensor([1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000,
        1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000,
        1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000,
        1.0000, 1.0000, 1.0000, 1.0000, 1.0000], grad_fn=<SumBackward1>)
```

0.4 Cost function

[25]:
```
# reference: https://docs.pytorch.org/docs/stable/generated/torch.nn.
 ↪CrossEntropyLoss.html#torch.nn.CrossEntropyLoss
```

[26]:
```
# model without softmax because nn.CrossEntropyLoss() have included
class ImageClassificationModel2 (nn.Module):
    def __init__(self,input_shape,output_shape):
        super().__init__()
        self.layer_stack = nn.Sequential(
            nn.Flatten(start_dim = 1, end_dim = -1),  # x_first_batch.shape =␣
 ↪torch.Size([32, 3, 32, 32]) should pick(3,32,32). Thus, change start_dim to 1
            nn.Linear(in_features = input_shape, out_features = output_shape)
            # ,nn.Softmax(dim = 1) # model(x_first_batch).shape = torch.
 ↪Size([32, 10]), should pick 10. Thus, change to dim = 1
        )

    def forward(self, x):
        return self.layer_stack(x)
```

[27]:
```
x_first_batch, y_first_batch = next(iter(train_dataloader))
x_first_batch.shape,y_first_batch.shape
```

[27]:
```
(torch.Size([32, 1, 28, 28]), torch.Size([32]))
```

[28]:
```
torch.manual_seed(87)
model2 = ImageClassificationModel2(784,10)

cost_fn = nn.CrossEntropyLoss()
y_pred = model2(x_first_batch)
cost = cost_fn(y_pred, y_first_batch)
print(model2.state_dict()) #before
print(cost) #before
```

```
optimizer = torch.optim.SGD(params = model2.parameters(), lr = 0.01, momentum=0.
 ↪9, weight_decay=1e-4)
optimizer.zero_grad()
cost.backward() # gradient descent
optimizer.step() # Once the gradients are ready, this tells the optimizer (e.g.
 ↪, SGD, Adam) to update the parameters.

y_pred = model2(x_first_batch)
cost = cost_fn(y_pred, y_first_batch)
print(model2.state_dict()) #after
print(cost) #after
```

```
OrderedDict([('layer_stack.1.weight', tensor([[-0.0355,  0.0281, -0.0032,  …,
-0.0253,  0.0329, -0.0172],
        [-0.0108, -0.0178,  0.0132,  …,  0.0324, -0.0354,  0.0180],
        [-0.0197,  0.0092,  0.0163,  …, -0.0204,  0.0296, -0.0310],
        …,
        [ 0.0022,  0.0250, -0.0206,  …,  0.0313,  0.0052, -0.0215],
        [ 0.0283,  0.0323,  0.0323,  …,  0.0146, -0.0040,  0.0086],
        [ 0.0243,  0.0077, -0.0030,  …,  0.0065,  0.0205,  0.0202]])),
('layer_stack.1.bias', tensor([ 0.0099,  0.0330,  0.0156, -0.0259,  0.0210,
-0.0344, -0.0287,  0.0027,
        -0.0189,  0.0245]))])
tensor(2.3080, grad_fn=<NllLossBackward0>)
OrderedDict([('layer_stack.1.weight', tensor([[-0.0355,  0.0281, -0.0032,  …,
-0.0253,  0.0329, -0.0172],
        [-0.0108, -0.0178,  0.0132,  …,  0.0324, -0.0354,  0.0180],
        [-0.0197,  0.0092,  0.0163,  …, -0.0204,  0.0296, -0.0310],
        …,
        [ 0.0022,  0.0250, -0.0206,  …,  0.0313,  0.0052, -0.0215],
        [ 0.0283,  0.0323,  0.0323,  …,  0.0146, -0.0040,  0.0086],
        [ 0.0243,  0.0077, -0.0030,  …,  0.0065,  0.0205,  0.0202]])),
('layer_stack.1.bias', tensor([ 0.0089,  0.0327,  0.0154, -0.0253,  0.0208,
-0.0343, -0.0296,  0.0025,
        -0.0181,  0.0256]))])
tensor(2.2704, grad_fn=<NllLossBackward0>)
```

[29]:
```
device = "cuda" if torch.cuda.is_available() else "cpu"
model2.to(device)
```

[29]:
```
ImageClassificationModel2(
  (layer_stack): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=784, out_features=10, bias=True)
  )
)
```

[30]:
```
epochs = 3


for epoch in range(epochs):
    print(f"Epoch: {epoch} \n -----")

    train_cost = 0
    train_acc = 0
    for batch, (x, y) in enumerate(train_dataloader):

        x = x.to(device)
        y = y.to(device)

        model2.train()
        y_pred =  model2(x)
        cost = cost_fn(y_pred,y)
        train_acc += (y_pred.argmax(dim=1)==y).sum() / len(y) * 100

        train_cost += cost

        optimizer = torch.optim.SGD(params = model2.parameters(), lr = 0.01,␣
 ↪momentum=0.9, weight_decay=1e-4)
        optimizer.zero_grad()
        cost.backward()
        optimizer.step()

        if batch % 500 == 0:
            print(f"num. {batch * len(x)} / {len(train_data)} dataset")

    train_cost /= len(train_dataloader)
    train_acc /= len(train_dataloader)

    test_cost = 0
    test_acc = 0
    model2.eval()
    with torch.inference_mode():
        for x, y in test_dataloader:
            x = x.to(device)
            y = y.to(device)
            test_pred =  model2(x)

            test_cost += cost_fn(test_pred,y)
            test_acc += (test_pred.argmax(dim=1)==y).sum() / len(y) * 100

        test_cost /= len(test_dataloader)
        test_acc /= len(test_dataloader)
```

```
    print(f"\nTrain Cost: {train_cost:.4f}, {train_acc:.2f}")
    print(f"Test Cost: {test_cost:.4f}, {test_acc:.2f} \n")
```

```
Epoch: 0
 -----
num. 0 / 56000 dataset
num. 16000 / 56000 dataset
num. 32000 / 56000 dataset
num. 48000 / 56000 dataset

Train Cost: 0.7798, 83.26
Test Cost: 0.4995, 87.79

Epoch: 1
 -----
num. 0 / 56000 dataset
num. 16000 / 56000 dataset
num. 32000 / 56000 dataset
num. 48000 / 56000 dataset

Train Cost: 0.4596, 88.15
Test Cost: 0.4159, 89.28

Epoch: 2
 -----
num. 0 / 56000 dataset
num. 16000 / 56000 dataset
num. 32000 / 56000 dataset
num. 48000 / 56000 dataset

Train Cost: 0.4062, 89.09
Test Cost: 0.3815, 89.82
```

0.5 Save

[7]:
```
#save
# torch.save(model2.state_dict(), "image_classification_weights.pth")
```