mlp_torch

February 11, 2025

[1]:
```
import os
import gzip
import numpy as np
import matplotlib.pyplot as plt
import copy
import torch
import torch.nn as nn
from torchvision import datasets,transforms
import torch.optim as optim
```

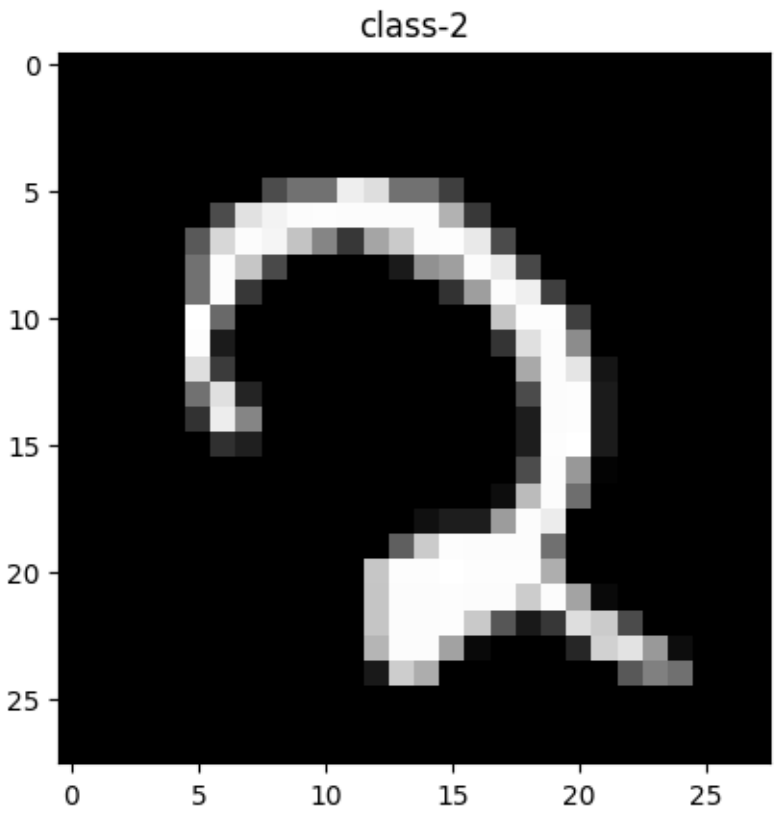
[2]:
```
device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

[3]:
```
def read_images(filename):
    """Read MNIST images"""

    with gzip.open(filename, 'r') as f:
        # first 4 bytes is a magic number
        magic_number = int.from_bytes(f.read(4), 'big')
        # second 4 bytes is the number of images
        image_count = int.from_bytes(f.read(4), 'big')
        # third 4 bytes is the row count
        row_count = int.from_bytes(f.read(4), 'big')
        # fourth 4 bytes is the column count
        column_count = int.from_bytes(f.read(4), 'big')
        # rest is the image pixel data, each pixel is stored as an unsigned byte
        # pixel values are 0 to 255
        image_data = f.read()
        images = np.frombuffer(image_data, dtype=np.uint8).
 ↪reshape((image_count, row_count, column_count))

    return images

def read_labels(filename):
    """Read MNIST labels"""
```

```
    with gzip.open(filename, 'r') as f:
        # first 4 bytes is a magic number
        magic_number = int.from_bytes(f.read(4), 'big')
        # second 4 bytes is the number of labels
        label_count = int.from_bytes(f.read(4), 'big')
        # rest is the label data, each label is stored as unsigned byte
        # label values are 0 to 9
        label_data = f.read()
        labels = np.frombuffer(label_data, dtype=np.uint8)

    return labels


dataset_path = r"C:\Users\DEBARSHI\Documents\Programs\Python\Deep Learning and␣
 ↪Natural Language Processing, DA345\Assignments\Assignment 2\data"

train_image_filename = os.path.join(dataset_path, 'train-images-idx3-ubyte.gz')
train_label_filename = os.path.join(dataset_path, 'train-labels-idx1-ubyte.gz')

test_image_filename = os.path.join(dataset_path, 't10k-images-idx3-ubyte.gz')
test_label_filename = os.path.join(dataset_path, 't10k-labels-idx1-ubyte.gz')

train_images = read_images(train_image_filename)
train_labels = read_labels(train_label_filename)

print('Train data (X) size: {}, and labels (Y) size: {}' .format(train_images.
 ↪shape, train_labels.shape))

test_images = read_images(test_image_filename)
test_labels = read_labels(test_label_filename)

print('Test data (X) size: {}, and labels (Y) size: {}' .format(test_images.
 ↪shape, test_labels.shape))



rand_ids = np.random.choice(train_images.shape[0])
plt.imshow(train_images[rand_ids, :, :], cmap='gray')
plt.title('class-'+str(train_labels[rand_ids]))
plt.show()
```

```
Train data (X) size: (60000, 28, 28), and labels (Y) size: (60000,)
Test data (X) size: (10000, 28, 28), and labels (Y) size: (10000,)
```



[5]:
```
class TwoLayerModel(nn.Module):
    def __init__(self,input_size=784,hidden_size=128,output_size=10):
        super(TwoLayerModel,self).__init__()
        self.fc1=nn.Linear(input_size,hidden_size)
        self.fc2=nn.Linear(hidden_size,output_size)

    def forward(self,x):
        x=x.view(-1,28*28)
        x=torch.relu(self.fc1(x))
        x=self.fc2(x)
        return x
```

[6]:
```
model=TwoLayerModel().to(device)
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.01,momentum=0.9)
```

[ ]:
```
num_epochs=5
train_losses=[]
train_accuracies=[]
test_accuracies=[]
```

```
print("Training the model...")
for epoch in range(num_epochs):
    model.train()
    running_loss=0.0
    correct=0
    total=0
```

[ ]: