Exercise 1: Neural Networks on the Olivetti faces dataset

Preparing and splitting training and test set

Data set size: 400*4096

Training set has 360 records and test set has 40 records as the train test ratio is 9:1

```
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
from sklearn.datasets import fetch_olivetti_faces
import numpy as np
olivetti = fetch_olivetti_faces()
train = olivetti.images
label = olivetti.target

X = train
Y = label
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, stratify=Y)
np.unique(y_train, return_counts=True)
np.unique(y_test, return_counts=True)

# train_dataset = Dataset(X_train)
trainX_loader = DataLoader(X_train)
trainY_loader = DataLoader(y_train)
testX_loader = DataLoader(X_test)
testY_loader = DataLoader(y_test)


print(len(trainX_loader),len(trainY_loader),len(testX_loader),len(testY_loader))
```

```
360 360 40 40
```

Structure

Input with 4096 features is fed into Fully connected layer and output from this layer is of dimension 2500

Second fully connected layer takes input with 2500 features and gives out outptu with dimension 40

```
import torch.nn as nn
import torch.nn.functional as F
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(4096, 2500)
        self.fc2 = nn.Linear(2500, 40)


    def forward(self, x):

        x = x.view(-1,4096)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x


net = Net()


from mpi4py import MPI


pip install mpi4py
```

```
Collecting mpi4py
  Downloading https://files.pythonhosted.org/packages/ec/8f/bbd8de5ba566dd77e408d8136e2b
     |████████████████████████████████| 1.4MB 3.3MB/s
Building wheels for collected packages: mpi4py
  Building wheel for mpi4py (setup.py) ... done
  Created wheel for mpi4py: filename=mpi4py-3.0.3-cp36-cp36m-linux_x86_64.whl size=2074
  Stored in directory: /root/.cache/pip/wheels/18/e0/86/2b713dd512199096012ceca61429e12b
Successfully built mpi4py
Installing collected packages: mpi4py
Successfully installed mpi4py-3.0.3
```

Training

Loss used: Cross entropy loss

Epochs: 5

```
import torch.optim as optim
import torch
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)


def custom_loss_crossentropy(x,y):
    log_prob = -1.0 * F.log_softmax(x, 1)
    loss = log_prob.gather(1, y.unsqueeze(1))
    loss = loss.mean()
    return loss
```

```
        return loss

    start = MPI.Wtime()
    i=0
    train_accuracy = []
    test_accuracy = []
    for epoch in range(5):  # loop over the dataset multiple times
        print('epoch', epoch)
        running_loss = 0.0
        for inputs, labels in zip(trainX_loader, trainY_loader):

            # zeroing the gradients
            optimizer.zero_grad()
            #Feeding the input to the model
            outputs = net(inputs)
            loss= criterion(outputs, labels)

            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            if i % 2000 == 1999:
                print('[%d, %5d] loss: %.3f' %
                      (epoch + 1, i + 1, running_loss / 2000))
                running_loss = 0.0
            i+=1
        #Calculating the training and test accuracy after every 2000 iterations
        total = 0
        correct = 0
        for images,labels in zip(trainX_loader,trainY_loader):

            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            #         _, predicted2 = torch.max(outputs[1].data, 1)
            total += labels.size(0)
            correct += (predicted == labels ).sum().item()
        train_accuracy.append(100 * correct / total)
            # print('[%d, %5d] loss: %.3f' %
            #           (epoch + 1, i + 1, running_loss / 2000))
        total = 0
        correct = 0
        for images,labels in zip(testX_loader,testY_loader):

            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            #   _, predicted2 = torch.max(outputs[1].data, 1)
            total += labels.size(0)
            correct += (predicted == labels ).sum().item()
        test_accuracy.append(100 * correct / total)

    print('Finished Training')
    totaltime = MPI.Wtime() - start
```

```
epoch 0
epoch 1
epoch 2
epoch 3
epoch 4
Finished Training
```

Accuracy observed: 2%

```
correct = 0
total = 0
with torch.no_grad():
    for images,labels in zip(testX_loader, testY_loader):
        # images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        #print(predicted,labels)
print('Accuracy : %d %%' % (
    100 * correct / total))
```

```
Accuracy : 2 %
```

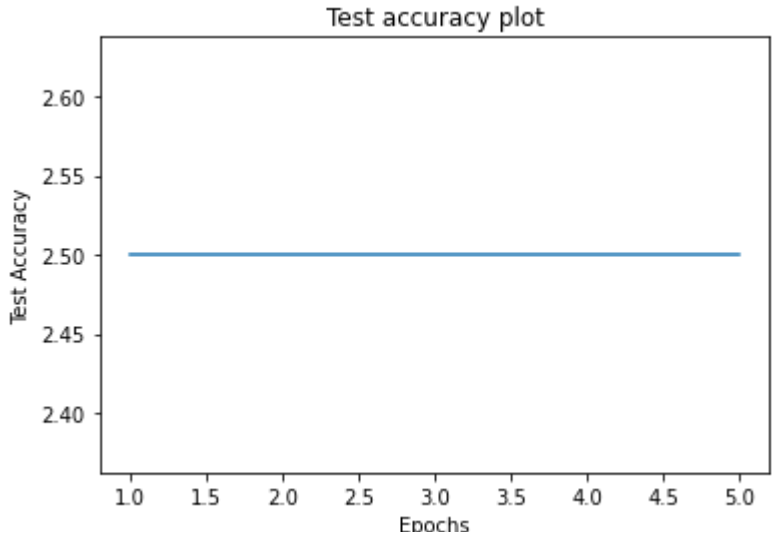
```
#


print(train_accuracy,test_accuracy, totaltime)
```

```
[2.5, 2.5, 2.5, 2.5, 2.5] [5.0, 5.0, 5.0, 5.0, 5.0] 138.04636402100004
```

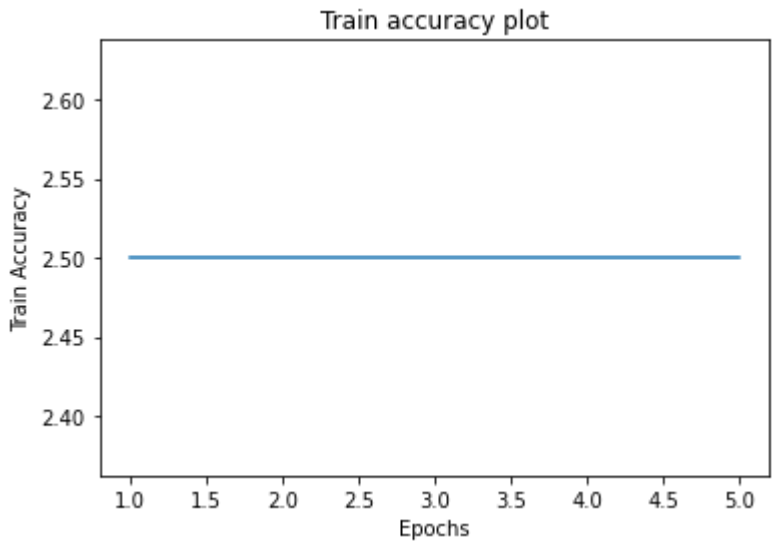
Time taken: 138 seconds

Train and test accuracy plots

```
import matplotlib.pyplot as plt
epochs = [1,2,3,4,5]

test_acc = [2.5, 2.5, 2.5, 2.5, 2.5]
train_acc = [100.0, 100.0, 100.0, 100.0]
plt.plot(epochs,test_acc)
plt.ylabel('Test Accuracy')
plt.xlabel('Epochs')
plt.title('Test accuracy plot')
plt.show()
```




```
import matplotlib.pyplot as plt
epochs = [1,2,3,4,5]

train_acc = [2.5, 2.5, 2.5, 2.5, 2.5]

plt.plot(epochs,test_acc)
plt.ylabel('Train Accuracy')
plt.xlabel('Epochs')
plt.title('Train accuracy plot')
plt.show()
```



Excercise 2: Deep Neural Networks on the Olivetti faces dataset

Structure

Input with 4096 features is fed into Fully connected layer and output from this layer is of dimension 3000

Second fully connected layer takes input with dimension 2500 and gives out outptu with dimension 1500

Output layer takes input with size 1500 and gives out output of dimension 40

```
#Ex2:
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(4096, 3000)
        self.fc2 = nn.Linear(3000, 1500)
        self.fc3 = nn.Linear(1500, 40)


    def forward(self, x):

        x = x.view(-1,4096)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

Training

Loss used: Cross entropy loss

Epochs: 5

```
def custom_loss_crossentropy(x,y):
    log_prob = -1.0 * F.log_softmax(x, 1)
    loss = log_prob.gather(1, y.unsqueeze(1))
    loss = loss.mean()
    return loss
def custom_loss_mse(output, target):
    loss = torch.mean((output - target)**2)
    return loss
```

```
    return loss
start = MPI.Wtime()
i=0
train_accuracy = []
test_accuracy = []
for epoch in range(5):  # loop over the dataset multiple times
    print('epoch', epoch)
    running_loss = 0.0
    for inputs, labels in zip(trainX_loader, trainY_loader):

        # zeroing the gradients
        optimizer.zero_grad()

        #feeding the input to the model
        outputs = net(inputs)
        loss= criterion(outputs, labels)
                loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 2000 == 1999:
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
        i+=1
    #Calculating test and train accuracy for every 2000 iterations
    total = 0
    correct = 0
    for images,labels in zip(trainX_loader,trainY_loader):

        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels ).sum().item()
    train_accuracy.append(100 * correct / total)

    total = 0
    correct = 0
    for images,labels in zip(testX_loader,testY_loader):

        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels ).sum().item()
    test_accuracy.append(100 * correct / total)

print('Finished Training')
totaltime = MPI.Wtime() - start
```

```
epoch 0
epoch 1
epoch 2
epoch 3
epoch 4
Finished Training
```

Accuracy observed: 2%

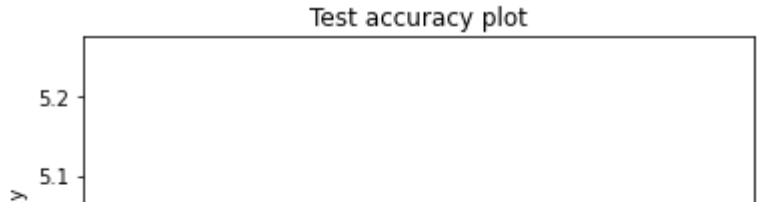
```
#training set
correct = 0
total = 0
with torch.no_grad():
    for images,labels in zip(trainX_loader, trainY_loader):
        # images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        #print(predicted,labels)
print('Accuracy : %d %%' % (
    100 * correct / total))
```

```
Accuracy : 2 %
```

```
print(train_accuracy, test_accuracy)
```

```
[2.5, 2.5, 2.5, 2.5, 2.5] [5.0, 5.0, 5.0, 5.0, 5.0]
```

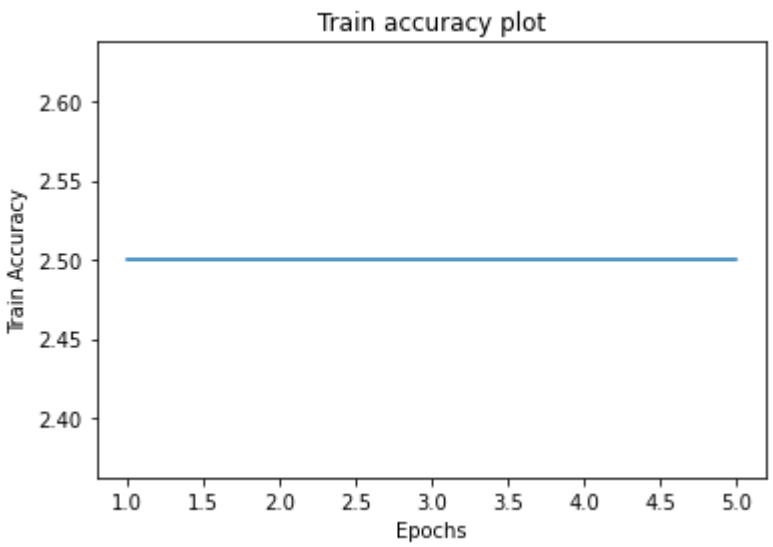
```
import matplotlib.pyplot as plt
epochs = [1,2,3,4,5]

train_acc = [2.5, 2.5, 2.5, 2.5, 2.5]
test_acc = [5.0, 5.0, 5.0, 5.0, 5.0]
plt.plot(epochs,test_acc)
plt.ylabel('Test Accuracy')
plt.xlabel('Epochs')
plt.title('Test accuracy plot')
plt.show()
```




```
print(totaltime)
```

```
138.04636402100004
```

```
train_acc = [2.5, 2.5, 2.5, 2.5, 2.5]
test_acc = [5.0, 5.0, 5.0, 5.0, 5.0]
plt.plot(epochs,train_acc)
plt.ylabel('Train Accuracy')
plt.xlabel('Epochs')
plt.title('Train accuracy plot')
plt.show()
```



Time taken: 138 seconds

Comparing the time taken for ex1 and 2: Interestingly, the time taken for both the models is the same that is 138 seconds. This might be because that the addition of one additional hidden layer

Number of trainable parameters EX1:

first layer : 4096X2500 (weights)+ 2500(bias)

second layer: 2500X40(weights) + 40(bias) Totally: 10342540 trainable parameters

Number of trainable parameters EX2:

first layer : 4096X3000 (weights)+ 3000(bias)

second layer: 3000X1500(weights) + 1500(bias)

third layer: 1500X40(weights) + 40(bias)

Totally: 16852540 trainable parameters

Percentage accuracy gain is 0. Since accuracy for both models is 2%. As there is information loss while flattening an image into a a vector, the accuracy is very low for both the models