```python
import torch
import numpy as np
import pandas as pd
import string
import argparse
import os
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
import matplotlib.pyplot as plt


class LSTM(nn.Module):
    def __init__(self):
        super(LSTM, self).__init__()
        self.lstm = nn.LSTM(27,54,batch_first=True) # lstm
        self.fc = nn.Linear(54,128)
        self.fc2 = nn.Linear(128,27)


    def forward(self, x):
        h_n = torch.zeros(1,1,54)
        c_n = torch.zeros(1,1,54)
        output, (hn, cn) = self.lstm(x,(h_n,c_n))
        out = F.relu(output)
        out = self.fc(out)
        out = F.relu(out)
        out = self.fc2(out)
        return out


def train(args, model, list_names, list_y, optimizer, epoch):
    #dev = torch.device("cuda")
    model.train()
    tot_loss = 0
    correct = 0
    optimizer.zero_grad()
    i=0
    count_letter=0
    for name,y_name in zip(list_names,list_y):
        i+=1
        name = torch.as_tensor(name).reshape(1,11,27).type(torch.FloatTensor)
        y_name = torch.as_tensor(y_name).reshape(1,11,27).type(torch.FloatTensor)
        output = model(name)


        for pred_lett,y_lett in zip(output[0],y_name[0]):
            count_letter+=1
            if np.argmax(pred_lett.detach().numpy()) == np.argmax(y_lett.detach().numpy()):
                correct+=1


        loss = torch.nn.MSELoss()(output[0], y_name[0])
        loss.backward()
        optimizer.step()
        tot_loss = tot_loss + loss.item()
        if i % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}, Accuracy Lettera: {:.2f}%'.format(
                epoch, i , len(list_names),
                       100. * i / len(list_names), tot_loss / i,
                       100.0 * correct / ( count_letter * args.batch_size)))

    print('End of Epoch: {}'.format(epoch))
    print('Training Loss: {:.6f}, Training Accuracy : {:.2f}%'.format(
        tot_loss / (len(list_names)), 100.0 * correct / (len(list_names) * 11)))
    return tot_loss/ (len(list_names))


#defining the main
```

```
def main():
    # Training settings
    os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'
    parser = argparse.ArgumentParser(description='PyTorch Shape Example')
    parser.add_argument('--batch-size', type=int, default=1, help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=300,
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=50, help='number of epochs to train (default: 14)')
    parser.add_argument('--lr', type=float, default=1e-4, help='learning rate (default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.9, help='Learning rate step gamma (default: 0.7)')
    parser.add_argument('--seed', type=int, default=1, help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=100,
                        help='how many batches to wait before logging training status')
    parser.add_argument('--save-model', action='store_true', default=True, help='For Saving the current Model')
    parser.add_argument('--dev', default='cuda', help='dev')
    args = parser.parse_args()



    #Definition of the training set x and y
    names = open("names.txt", "r")
    list_names = []
    for el in names:
        list_names.append(el.lower())

    alpha_dict = {k: ord(k)-96 for k in string.
ascii_lowercase}
    alpha_dict['\n'] = 0
    encoded_names = []
    for name in list_names:
        encoding_name=[]
        for i in range(11):
            zeros = np.zeros(27,dtype=np.int8)
            if i>=len(name):
                zeros[alpha_dict['\n']]=1
            else:
                zeros[alpha_dict[name[i]]] = 1
            encoding_name.append(zeros.tolist())
        encoded_names.append(encoding_name)

    y = []
    for name in encoded_names:
        y_i =[]
        for i in range(11):
            if i != 10:
               y_i.append(name[i+1])
            else:
               eon = [0 for el in range(27)]
               eon[0] = 1
               y_i.append(eon)
        y.append(y_i)

    is_cuda = torch.cuda.is_available()
    if is_cuda:
        device = torch.device("cuda")
    else:
        device = torch.device("cpu")
    x = encoded_names.copy()
    y = y.copy()

    #model = LSTM().to(device)
    model = LSTM()

    optimizer = optim.Adam(model.parameters(), lr=
args.lr)
    scheduler = StepLR(optimizer, step_size=1, gamma
=args.gamma)

    loss_for_epoch=[]
    for epoch in range(1, args.epochs + 1):
        loss_for_epoch.append(train(args, model, x, y, optimizer, epoch))
        scheduler.step()

    plt.plot([i for i in range (1,args.epochs+1)], loss_for_epoch)
    plt.title('Plot Loss vs Epochs')
    plt.show()

    if args.save_model:
        torch.save(model.state_dict(), "f0702_662965513_Bartoletti.ZZZ")



if __name__ == '__main__':
    main()
```