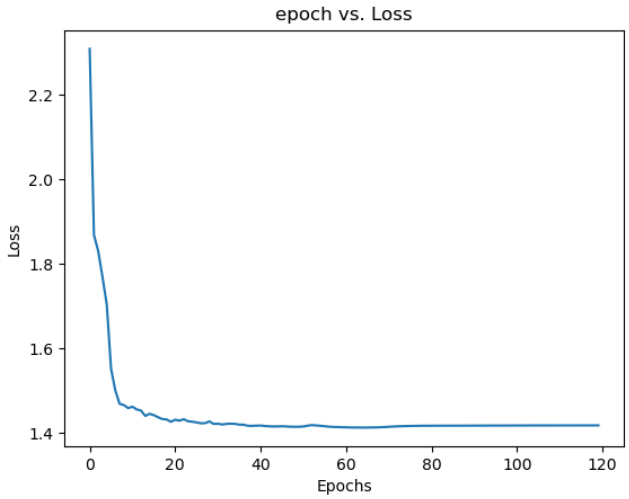
CS559 homework 7

655490960
Huiyang Zhao

2. Train the neural network, name your training code as 0701-IDNumber-LastName.py. Save the trained model as 0702-IDNumber-LastName.ZZZ. There are different ways to go here, but I will let you overfit for the sake of simplicity. Plot the loss value versus epochs and include this graph in your report. Indicate the mathematical expression for the loss function you have used. In particular, discuss the reason why you converged to the particular loss value you got.

Loss value vs. epochs is attached:



Mathematical expression for the loss function:

$$Loss(p, q) = -\sum_{i=1}^{n} p(x_i)\log\left(q(x_i)\right)$$

Where p stands for the distribution for the ground truth and q stands for the distribution for the output. n is the number of labels' size, in this case, 27.

We can see the loss value decreases quickly until around 1.42 and converges after around 30 epochs.
I think the reason loss stops decreasing is that my network cannot make full use of states generated in the training process. I have posted the problem I occurred on Piazza. I paid a lot of effort to solving the problem but I wasn't able to do that.
RuntimeError: one of the variables needed for gradient computation has been modified by an inplace operation: [torch.FloatTensor [128, 512]], which is output 0 of AsStridedBackward0, is at version 2; expected version 1 instead. Hint: the backtrace further above shows the operation that failed to compute its gradient. The variable in question was changed in there or anywhere later. Good luck!

3. Include your design choice in your report.

My design:

Select letters with first 3 highest possibility.
Randomly choose a letter from them.
If the letter is corresponding to EON then the process terminates and name is generated.
Otherwise loop until the length of name is 11.

My results:

Feed a: ['anieenna', 'anaienaaan', 'anaaae', 'anaaaaa', 'aaaaaanaa', 'aneaeenn', 'aanennannnn', 'aneaee', 'aeaaneannan', 'aeaenne', 'aaneenannnn', 'anaiaaaa', 'aanaaa', 'aeenaaan', 'aenaanenn', 'aneannen', 'aaeneaanaa', 'aneiaeaaana', 'aaenea', 'aanennnaaa']
Feed x: ['xaanaea', 'xienaeaa', 'xaaaanan', 'xianiaa', 'xaeeae', 'xeeeena', 'xieannaaa', 'xeenaaa', 'xeiaaannnnn', 'xaeanaannnn', 'xieaane', 'xieean', 'xeaaananaaa', 'xianieaaa', 'xaenaee', 'xeaaenenan', 'xeiaen', 'xaaeen', 'xeiiaannn', 'xieena']

```
# 6701-655490960-Zhao
# CS559 Neural Network
# Huiyang Zhao
# UIN 655490960

import numpy as np
import argparse
import random
import torch
from torch import nn
from torch.utils.data import Dataset
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR
import torch.optim as optim
import matplotlib.pyplot as plt

device = torch.device("cuda") if torch.cuda.is_available() else
torch.device("cpu")
random.seed(655490960)

EON = '/eon'
letters = []
for ch in 'abcdefghijklmnopqrstuvwxyz':
    letters.append(ch)
letters.append(EON)
print(letters)

letters_dict = {}
for key, value in enumerate(letters):
    letters_dict[key] = value
print(letters_dict)


# https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers,
batch_size):
        super().__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_layers = num_layers
        self.batch_size = batch_size
        self.relu = nn.ReLU()

        self.hn = torch.zeros(self.num_layers, self.batch_size,
self.hidden_size).to(device)
        self.cn = torch.zeros(self.num_layers, self.batch_size,
self.hidden_size).to(device)

        self.lstm = nn.LSTM(
            batch_first=True,
            input_size=self.input_size,
```

```
            hidden_size=self.hidden_size,
            num_layers=self.num_layers,
        )

        self.fc = nn.Linear(self.hidden_size, self.output_size)
        # self.linear1 = nn.Linear(self.hidden_size, 64)
        # self.linear2 = nn.Linear(64, self.output_size)

    def initial_hidden(self):
        hc = (torch.zeros(self.num_layers, self.batch_size,
self.hidden_size).to(device),
              torch.zeros(self.num_layers, self.batch_size,
self.hidden_size).to(device))
        return hc

    def forward(self, x, hc):
        x, (h1, c1) = self.lstm(x.float(), hc)
        x = self.relu(x)
        x = self.fc(x)
        # x = self.linear1(x)
        # x = self.linear2(x)

        return x, (h1, c1)


def encode(letter):
    encoded = [0 for i in range(27)]
    index = list(letters_dict.values()).index(letter)
    encoded[index] = 1
    return encoded


def preprocess():
    file = open('names.txt', 'r')
    names = file.readlines()

    input_names = []
    output = []

    for name in names:
        name_list = []
        for ch in name.replace('\n', '').lower():
            name_list.append(ch)
        while len(name_list) < 11:
            name_list.append(EON)
        label = name_list[1:]
        label.append(EON)

        input_names.append(torch.tensor([encode(ch) for ch in name_list]))
        output.append(torch.tensor([encode(ch) for ch in label]))

    return input_names, output
```

```
def train(args, model, device, train_loader, optimizer, epoch,
error_array):
    model.train()
    tot_loss = 0
    correct = 0
    hc = model.initial_hidden()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output, hc = model(data, hc)
        hc = tuple([each.data for each in hc])
        target = target.argmax(axis=2)
        temp = torch.transpose(output, 2, 1)
        loss = torch.nn.CrossEntropyLoss()(temp, target)
        loss.backward(retain_graph=True)
        optimizer.step()

        pred = output.argmax(dim=2, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()

        tot_loss = tot_loss + loss.item()

    print('End of Epoch: {}'.format(epoch))
    print('Training Loss: {:.6f}'.format(tot_loss / (len(train_loader))))
    error_array.append(tot_loss / len(train_loader))


class lstm_dataset(Dataset):
    def __init__(self, data, labels):
        self.data = data
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        data = self.data[idx]
        label = self.labels[idx]
        return data, label


def main():
    data, labels = preprocess()

    parser = argparse.ArgumentParser(description='CS559 hw7')
    parser.add_argument('--batch-size', type=int, default=100, help='input
batch size for training (default: 100)')
    parser.add_argument('--test-batch-size', type=int, default=100,
                        help='input batch size for testing (default:
100)')
    parser.add_argument('--epochs', type=int, default=120, help='number of
epochs to train (default: 60)')
    parser.add_argument('--lr', type=float, default=0.01, help='learning
rate (default: 1.0)')
```

```
    parser.add_argument('--gamma', type=float, default=0.6, help='Learning
rate step gamma (default: 0.7)')
    parser.add_argument('--seed', type=int, default=655490960,
help='random seed (default: 655490960)')
    parser.add_argument('--log-interval', type=int, default=10,
                        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')
    args = parser.parse_args()

    torch.manual_seed(args.seed)
    torch.autograd.set_detect_anomaly(True)

    dataset = lstm_dataset(data, labels)
    data_loader = torch.utils.data.DataLoader(dataset,
batch_size=args.batch_size)
    model = LSTM(input_size=27, hidden_size=64, output_size=27,
num_layers=1, batch_size=args.batch_size).to(device)

    optimizer = optim.Adam(model.parameters(), lr=args.lr,
weight_decay=0.01)
    scheduler = StepLR(optimizer, step_size=6, gamma=args.gamma)

    error_array = []

    for epoch in range(1, args.epochs + 1):
        train(args, model, device, data_loader, optimizer, epoch,
error_array)
        scheduler.step()

    if args.save_model:
        torch.save(model.state_dict(), "train.pt")

    epoch_array = range(args.epochs)

    plt.figure()
    plt.title('epoch vs. Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(epoch_array, error_array)
    plt.savefig('epoch_vs_loss')
    plt.show()


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

```
# 0703-655490960-Zhao
# CS559 Neural Network
# Huiyang Zhao
# UIN 655490960

import numpy as np
import argparse
import random
import torch
from torch import nn
from torch.utils.data import Dataset
from torchvision import datasets, transforms
from torch.optim.lr_scheduler import StepLR
import torch.optim as optim
import matplotlib.pyplot as plt

device = torch.device("cuda") if torch.cuda.is_available() else
torch.device("cpu")
random.seed(655490960)

EON = '/eon'
letters = []
for ch in 'abcdefghijklmnopqrstuvwxyz':
    letters.append(ch)
letters.append(EON)
print(letters)

letters_dict = {}
for key, value in enumerate(letters):
    letters_dict[key] = value
print(letters_dict)


def encode(letter):
    encoded = [0 for i in range(27)]
    index = list(letters_dict.values()).index(letter)
    encoded[index] = 1
    return encoded


def generate(ch, model):
    input_ch = [encode(ch)]
    # input_ch = encode(ch)
    hc = model.initial_hidden()
    generated_name = ch

    for i in range(11):
        torch_input_ch = torch.tensor(input_ch)
        output, hc = model(torch_input_ch, hc)
        # print(output.shape)
        output = output[-1].detach()
        # print(output.shape)
        '''select 3 letters with highest possibilities'''
        indexes = list(np.argpartition(output, -3)[-3:].numpy())
        # print(indexes)
        '''randomly choose one letter'''
        chosen = np.random.choice(indexes)
```

```
        '''if the chosen one is EON then terminates, o.w. loop until length
of name is 11.'''
        if chosen == 26 or len(generated_name) == 11:
            break
        else:
            generated_name += letters_dict.get(chosen)
            input_ch.append(encode(letters_dict.get(chosen)))

    return generated_name


# https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html
class LSTM(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers,
batch_size):
        super().__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.num_layers = num_layers
        self.batch_size = batch_size
        # self.dropout = nn.Dropout(0.25)
        self.relu = nn.ReLU()

        self.hn = torch.zeros(self.num_layers, self.batch_size,
self.hidden_size).to(device)
        self.cn = torch.zeros(self.num_layers, self.batch_size,
self.hidden_size).to(device)

        self.lstm = nn.LSTM(
            batch_first=True,
            input_size=self.input_size,
            hidden_size=self.hidden_size,
            num_layers=self.num_layers,
            # dropout=0.1,
        )

        self.fc = nn.Linear(self.hidden_size, self.output_size)

    def initial_hidden(self):
        hc = (torch.zeros(self.num_layers, self.batch_size,
self.hidden_size).to(device),
              torch.zeros(self.num_layers, self.batch_size,
self.hidden_size).to(device))
        return hc

    def forward(self, x, hc):
        x, (h1, c1) = self.lstm(x.float())
        x = self.relu(x)
        x = self.fc(x)

        return x, (h1, c1)


def main():
    parser = argparse.ArgumentParser(description='CS559 hw7')
```

```
    parser.add_argument('--batch-size', type=int, default=100, help='input
batch size for training (default: 100)')
    parser.add_argument('--test-batch-size', type=int, default=100,
                        help='input batch size for testing (default: 100)')
    parser.add_argument('--epochs', type=int, default=200, help='number of
epochs to train (default: 60)')
    parser.add_argument('--lr', type=float, default=1, help='learning rate
(default: 1.0)')
    parser.add_argument('--gamma', type=float, default=0.7, help='Learning
rate step gamma (default: 0.7)')
    parser.add_argument('--seed', type=int, default=655490960, help='random
seed (default: 655490960)')
    parser.add_argument('--log-interval', type=int, default=10,
                        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')
    args = parser.parse_args()

    path = './0702-655490960-Zhao.pt'
    # path = './train.pt'

    model = LSTM(input_size=27, hidden_size=64, output_size=27, num_layers=1,
batch_size=args.batch_size).to(device)
    saved = torch.load(path)
    model.load_state_dict(saved)

    model.eval()

    num_names = 20
    generated_name_a = []
    generated_name_e = []

    for i in range(num_names):
        generated_name_a.append(generate('a', model))
        generated_name_e.append(generate('x', model))

    print('Feed a: ' + str(generated_name_a))
    print('Feed x: ' + str(generated_name_e))

    inputted = input("Enter the letter here: ")

    print(inputted)
    generated_name_input = []
    for i in range(num_names):
        generated_name_input.append(generate(inputted, model))

    print('Feed ' + inputted + ':' + str(generated_name_input))


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