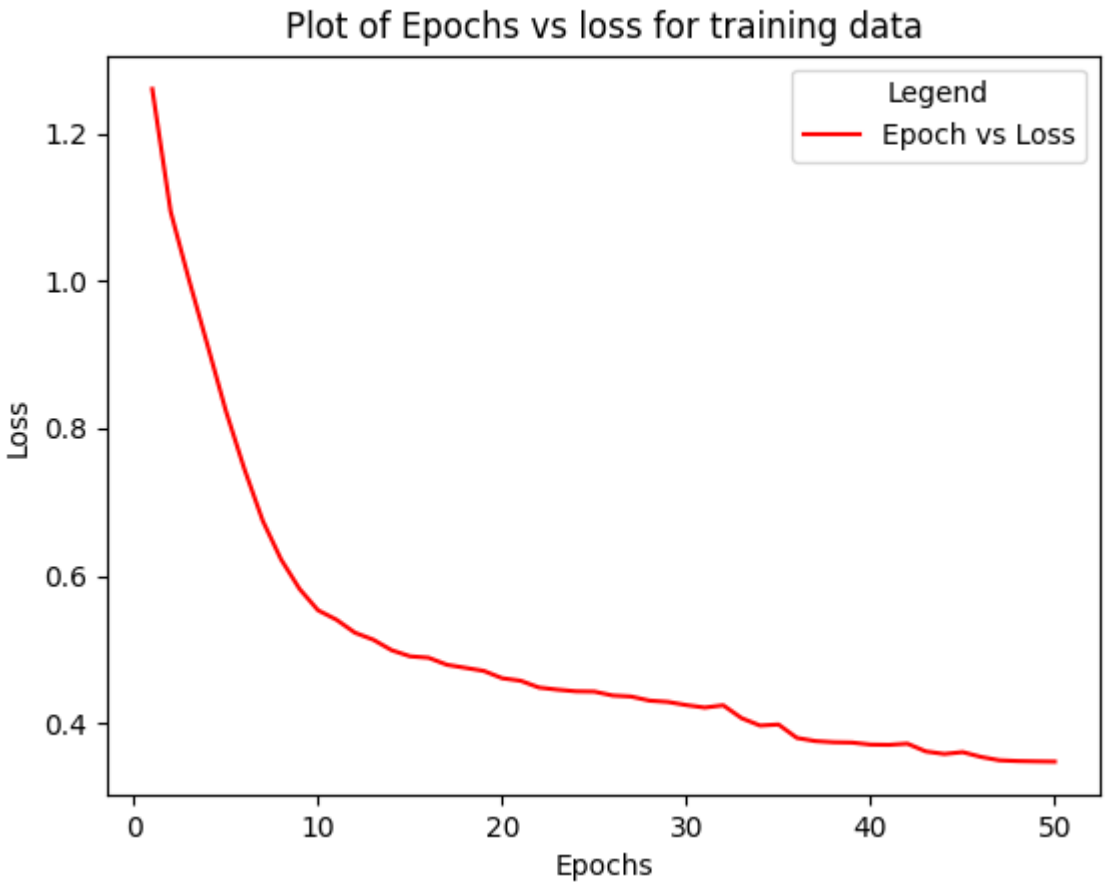
Name: Sai Anish Garapati
UIN: 650208577

Assignment 7:

2) Plot of epochs vs loss for list of names:



Final loss after 50 epochs: 0.347

- Loss function used is Categorical cross entropy loss
  $$loss(X, Y) = -\sum_{c_i \epsilon C} Y_{c_i} * log(Softmax(X_{c_i}))$$
  -

- The loss value decreases fine until 0.347, but after that the value does not change much and keeps fluctuating between 0.35 and 0.4. It could be that other optimizers or regularization might have been required.

- Design choice for inference model:
  - From the generated list of output characters with probabilities, choose the characters with top 3 probabilities and the make a random choice among the 3 to append to the name and feed as input in the next iteration
  - When an EOS is encountered on random:
    - Considered the name only when length of name is at least 3 and the probability of EOS is greater than or equal to 0.5
  - Tried the same with some probabilities, but the generated names do not vary much

- 20 names starting with letter ‘a’:
    - ['aly', 'aralgeturelderottot', 'aralx', 'ani', 'ariatnofopelia', 'aratxt', 'anierleyt', 'aria', 'araltsjro', 'arlatsorellatte', 'arl', 'ann', 'alasopiagelvet', 'ala', 'aleiahotglie', 'arled', 'alistan', 'aliatraripfow', 'arallie', 'anlahh']

- 20 names starting with letter ‘x’:
    - ['xadetor', 'xinett', 'xiderdteoteroo', 'xzael', 'xzanitonoteltel', 'xzenil', 'xivoeyorre', 'xzanielototte', 'xadeeryttetouh', 'xzini', 'xinn', 'xade', 'xzeetett', 'xzere', 'xadroe', 'xzeelee', 'xzior', 'xanntet', 'xan', 'xidrettette']

Python code:

- **Training module**

```
# Name: Sai Anish Garapati
# UIN: 650208577

from math import gamma
import torch
import os
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim import optimizer
from torch.optim.lr_scheduler import StepLR
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
from random import shuffle


class RNN(nn.Module):
    def __init__(self, input_size=27, hidden_size=256, num_layers=2):
        super(RNN, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc1 = nn.Linear(hidden_size, input_size)

    def forward(self, x, hidden, cell):
        x, (hidden, cell) = self.lstm(x, (hidden, cell))
        x = self.fc1(x)
        return x, (hidden, cell)

    def init_hidden_and_cell(self):
        hidden = torch.zeros(self.num_layers, 1, self.hidden_size).to(device)
        cell = torch.zeros(self.num_layers, 1, self.hidden_size).to(device)
        return hidden, cell


def one_hot_encoder(name, seq_length):
```

```
    name_tensor = torch.zeros(seq_length, 1, 27)
    for i in range(len(name)):
        name_tensor[i][0][ord(name[i]) - ord('a') + 1] = 1
    for i in range(seq_length - len(name)):
        name_tensor[len(name) + i][0][0] = 1

    target_tensor = torch.zeros(seq_length)

    for i in range(1, len(name)):
        target_tensor[i - 1] = ord(name[i]) - ord('a') + 1
    for i in range(seq_length - len(name) + 1):
        target_tensor[len(name) - 1 + i] = 0

    return name_tensor, target_tensor


def train(model, device, names, seq_length, optimizer, epoch):
    model.train()
    tot_loss = 0

    for name in names:
        name_tensor, target_tensor = one_hot_encoder(name, seq_length)
        hidden, cell = model.init_hidden_and_cell()
        loss = 0

        optimizer.zero_grad()

        output, (hidden, cell) = model(name_tensor.reshape(1, seq_length, 27), hidden, cell)
        loss += torch.nn.CrossEntropyLoss()(output.squeeze(0),
target_tensor.type(torch.LongTensor))
        loss.backward()
        optimizer.step()

        tot_loss += loss.item()

    return tot_loss / len(names)


if __name__ == '__main__':
    max_seq_length = 11

    learning_rate = 0.002
    num_epochs = 50

    names_list = open('./names.txt', 'r').read().split('\n')
    names_dataset = [name.lower() for name in names_list]

    random.Random(2021).shuffle(names_dataset)

    torch.manual_seed(2021)

    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

```
    model = RNN().to(device)
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    scheduler = StepLR(optimizer, step_size=1, gamma=0.8)

    prev_loss = -1
    cur_loss = 0
    epoch_loss = []
    for epoch in range(num_epochs):
        cur_loss = train(model, device, names_dataset, max_seq_length, optimizer, epoch + 1)
        epoch_loss.append(cur_loss)
        if prev_loss != -1:
            if cur_loss > prev_loss:
                scheduler.step()

        if abs(cur_loss - prev_loss) < 1e-5:
            break;

        prev_loss = cur_loss
        print('End of epoch {}: loss: {}'.format(epoch + 1, cur_loss))

    plt.title('Plot of Epochs vs loss for training data')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(list(range(1, num_epochs + 1)), epoch_loss, 'r', label='Epoch vs Loss')
    plt.legend(title='Legend')
    plt.show()

    torch.save(model.state_dict(), './0702-650208577-Garapati.pt')
```

- **Inference module**

```
# Name: Sai Anish Garapati
# UIN: 650208577

from math import gamma
import torch
import os
from torch._C import _get_warnAlways
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.optim import optimizer
from torch.optim.lr_scheduler import StepLR
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
from random import shuffle

from torch.serialization import load
```

```
class RNN(nn.Module):
    def __init__(self, input_size=27, hidden_size=256, num_layers=2):
        super(RNN, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc1 = nn.Linear(hidden_size, input_size)

    def forward(self, x, hidden, cell):
        x, (hidden, cell) = self.lstm(x, (hidden, cell))
        x = self.fc1(x)
        return x, (hidden, cell)

    def init_hidden_and_cell(self):
        hidden = torch.zeros(self.num_layers, 1, self.hidden_size).to(device)
        cell = torch.zeros(self.num_layers, 1, self.hidden_size).to(device)
        return hidden, cell


def one_hot_encoder(name, seq_length):
    name_tensor = torch.zeros(seq_length, 1, 27)
    for i in range(len(name)):
        name_tensor[i][0][ord(name[i]) - ord('a') + 1] = 1
    for i in range(seq_length - len(name)):
        name_tensor[len(name) + i][0][0] = 1

    target_tensor = torch.zeros(seq_length)

    for i in range(1, len(name)):
        target_tensor[i - 1] = ord(name[i]) - ord('a') + 1
    for i in range(seq_length - len(name) + 1):
        target_tensor[len(name) - 1 + i] = 0

    return name_tensor, target_tensor

def test_model(device, model, char, seq_length):
    model.eval()
    hidden, cell = model.init_hidden_and_cell()
    generated_name = char

    with torch.no_grad():
        while True:
            name_tensor = one_hot_encoder(generated_name, len(generated_name))[0]
            output, (hidden, cell) = model(name_tensor.reshape(1, len(generated_name), 27),
hidden, cell)
            output = F.softmax(output[0][-1], dim=0).reshape(1, 27)
            topk_chars = torch.topk(output, 3)

            char_pick = np.random.randint(0, len(topk_chars.indices[0]), 1)

            idx, val = topk_chars.indices[0][char_pick], topk_chars.values[0][char_pick]
```

```
            if idx == 0:
                if len(generated_name) < 3 or val < 0.5:
                    continue
                return generated_name
            generated_name += chr(96 + idx)


if __name__ == '__main__':
    torch.manual_seed(2021)
    np.random.seed(2021)

    max_seq_length = 11

    device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

    loaded_model = RNN().to(device)
    loaded_model.load_state_dict(torch.load('./0702-650208577-Garapati.pt'))

    generated_names = []

    char_input = input('Enter character between a and z:\n')

    while True:
        name = test_model(device, loaded_model, char_input, 11)
        if name not in generated_names:
            generated_names.append(name)
        if len(generated_names) == 20:
            break
    print(generated_names)
```