Network Configuration used:

Hidden layer dimensions = 512
Batch_size = 20
Epochs = 300
Optimizer = Adam
Learning rate = 0.01

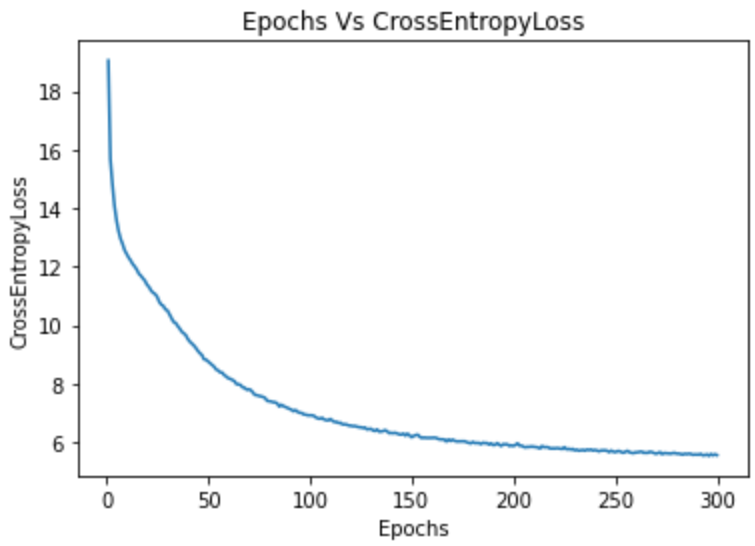
2. Loss function used is Cross Entropy Loss and the equation is

CrossEntropyLoss = $\Sigma y log(p(x))$

Where y is the true label and p(x) corresponds to the probability of the true label which is the output of the model. CrossEntropyLoss in pytorch applies a softmax to the output of a neural network to give probabilities. The summation happens around all the data.

I have used a simple network which is a LSTM cell followed by a fully connected network. I have used an LSTM cell because it gives flexibility when trying to generate names. All I need to feed a letter which is one hot encoded whereas in case of LSTM we may need to append EON to generate a sequence of length 11 before feeding it to the LSTM. Due to the nature of the simple network the loss value decreases after every epoch and as we can see from the graph the decrease in loss is smooth.

One of the things I have changed in loss is I have used reduction=”sum” in cross entropy because when reduction=”mean” for mean pytorch divides the loss by batch_size * input_shape In my case it will divide the loss by 20*27*11 but we need to divide the loss with only batch_size. Because the loss value may be higher than when we use reduction=”mean” but due to this change the gradient will be higher which will help the network learn better weights and better generalize the dataset.



3.
For the generation part I have used two different designs.
First design just takes the argmax of the last layer.
And
Output with starting letter “a” is
avery
alexandria
amari
alexandria
angelique
alexandria
aliana
alistair
ariana
amari
alexis
ariana
alan
ariah
amari
angelique
alexandria
alisha
aliana
alexandria

Output with starting letter “x” is
xander
ximena
xalentin
xaliyah
xariyah
ximena
xavier
xavier
xander
xander
xander
xterling
xamir
xavier
xander
xander
xleighton
ximena
xallas
xavier

Second design choice I randomly chose one of the top 5 highest probable sample.

Output with starting letter “a” is
amiyeatpi
aspreltphr
adamiyaniab
angrodymmad
angietosephe
ariyanaath
adriahnaas
alyvohadys
alejamiroo
anahtonaor
alishellis
adrysortpe
alayontrey
aranatsiah
anashlaqup
aresitreson
annikoleamie
ameertqoy
avinre

allensonara

Output with starting letter “x” is
xesminyarso
ximoreimel
xiagenaydrp
xsavielze
xamarielaezi
xloriyahama
xemilyehoped
xincelltt
xzerredahadd
xalite
xerariso
xoperyonedp
xulienahryn
xeridiamth
xuntantant
xantasocqo
xarielleela
xorehrf
xiriham
xordamalle

Code:

0701-659026651-Chintakunta.py

```
# -*- coding: utf-8 -*-
"""characterLSTM.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/10QKTa4JXJt3ogDgJMpQUNaDhOu7YOD9i
"""

import numpy as np
import torch
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
from torch import nn
import matplotlib.pyplot as plt
```

```
names = np.loadtxt("/content/names.txt",dtype='str')
names = [list(name.lower()) for name in names]
labels = [name[1:] for name in names]

for i in range(0,len(names)):
  if len(names[i]) < 11:
    j = 11 - len(names[i])
    while j:
      names[i].append("EON")
      j-=1
for i in range(0,len(labels)):
   if len(labels[i]) < 11:
     j = 11 - len(labels[i])
     while j:
       labels[i].append("EON")
       j-=1

for i in range(0,len(names)):
  for j in range(0,len(names[i])):
    if names[i][j] == "EON":
      names[i][j] = 0
    else:
      names[i][j] = ord(names[i][j]) - 96

for i in range(0,len(labels)):
  for j in range(0,len(labels[i])):
    if labels[i][j] == "EON":
      labels[i][j] = 0
    else:
      labels[i][j] = ord(labels[i][j]) - 96

names = torch.tensor(names)
labels = torch.tensor(labels)
names = torch.nn.functional.one_hot(names,num_classes=27)
# labels = torch.nn.functional.one_hot(labels,num_classes=27)

train_data = TensorDataset(names,labels)
train_dataloader = DataLoader(train_data,batch_size=20,shuffle=True)
device = "cuda" if torch.cuda.is_available() else "cpu"

class characterLSTM(nn.Module):
  def __init__(self):
    super(characterLSTM,self).__init__()
```

```
    self.hidden_dim = 512
    self.lstm = nn.LSTMCell(27,self.hidden_dim)
    self.linear = nn.Linear(self.hidden_dim,27)

  def forward(self,X):
                                                                                h,c                    =
torch.randn((X.shape[0],self.hidden_dim),requires_grad=True,device=device),torch.randn((X.sh
ape[0],self.hidden_dim),requires_grad=True,device=device)
    output = torch.empty((X.shape[0],27,11))
    for i in range(0,11):
      h,c = self.lstm(X[:,i,:],(h,c))
      out = self.linear(h)
      output[:,:,i] = out
    return output

  def generate_names_argmax(self,letter):
    output = []
    output.append(letter)
    letter = ord(letter) - 96
    letter = torch.tensor(letter)
    letter = torch.nn.functional.one_hot(letter,num_classes=27)
    letter = letter.unsqueeze(dim=0)
                                                                                h,c                    =
torch.randn((1,self.hidden_dim),device=device),torch.randn((1,self.hidden_dim),device=device)
    for i in range(0,11):
      h,c = self.lstm(letter.type(torch.FloatTensor).to(device),(h,c))
      out = self.linear(h)
      character = torch.argmax(out)
      if character != 0:
        output.append(chr(character + 96))
      letter = torch.nn.functional.one_hot(character,num_classes=27)
      letter = letter.unsqueeze(dim=0)
    st = ""
    output = st.join(output)
    return output

  def generate_names_top5(self,letter):
    output = []
    output.append(letter)
    letter = ord(letter) - 96
    letter = torch.tensor(letter)
    letter = torch.nn.functional.one_hot(letter,num_classes=27)
    letter = letter.unsqueeze(dim=0)
```

```
                                                                    h,c                    =
torch.randn((1,self.hidden_dim),device=device),torch.randn((1,self.hidden_dim),device=device)
    TopK = 5
    for i in range(0,11):
      h,c = self.lstm(letter.type(torch.FloatTensor).to(device),(h,c))
      out = self.linear(h)
      _,idxs = out[-1].topk(TopK)
      idx = torch.randint(0,TopK-1,(1,))
      character = idxs[idx]
      if character != 0:
        output.append(chr(character + 96))
      letter = torch.nn.functional.one_hot(character,num_classes=27)
    st = ""
    output = st.join(output)
    return output

loss_fn = torch.nn.CrossEntropyLoss(reduction="sum")
model = characterLSTM().to(device)
optimizer = torch.optim.Adam(model.parameters(),lr = 0.001)
epochs = 300
total_loss = []
for epoch in range(0,epochs):
  print("----------------Epoch " + str(epoch) + " ---------------------------------")
  loss_per_epoch = 0
  for batch,(X,y) in enumerate(train_dataloader):
    output = model(X.type(torch.FloatTensor).to(device))
                                                                        loss                    =
loss_fn(output.type(torch.FloatTensor).to(device),y.type(torch.LongTensor).to(device))/X.shape[0
]
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    loss_per_epoch += loss.item()
    if batch%80 == 0:
      print("Loss after " + str(batch) + " ",loss.item())
  loss_per_epoch/= len(train_dataloader)
  total_loss.append(loss_per_epoch)
  print("----------------End of Epoch " + str(epoch) + " ---------------------------")

plt.plot(list(range(1,epochs+1)),total_loss)
plt.xlabel("Epochs")
plt.ylabel("CrossEntropyLoss")
plt.title("Epochs Vs CrossEntropyLoss")
plt.show()
```

```
for i in range(0,20):
  print(model.generate_names_argmax("x"))

for i in range(0,20):
  print(model.generate_names_top5("x"))

torch.save(model,"0702-659026651-Chintakunta")
```

0703-659026651-Chintakunta.py

```
# -*- coding: utf-8 -*-
"""0703-659026651-Chintakunta.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1K--dvExQWPCpk6lw15K9mxLgmA2OqSQF
"""

import torch
from torch import nn

class characterLSTM(nn.Module):
  def __init__(self):
    super(characterLSTM,self).__init__()
    self.hidden_dim = 512
    self.lstm = nn.LSTMCell(27,self.hidden_dim)
    self.linear = nn.Linear(self.hidden_dim,27)

  def forward(self,X):
                                                                                h,c                    =
torch.randn((X.shape[0],self.hidden_dim),requires_grad=True,device=device),torch.randn((X.sh
ape[0],self.hidden_dim),requires_grad=True,device=device)
    output = torch.empty((X.shape[0],27,11))
    for i in range(0,11):
      h,c = self.lstm(X[:,i,:],(h,c))
      out = self.linear(h)
      output[:,:,i] = out
    return output

  def generate_names_argmax(self,letter):
    output = []
```

```
    output.append(letter)
    letter = ord(letter) - 96
    letter = torch.tensor(letter)
    letter = torch.nn.functional.one_hot(letter,num_classes=27)
    letter = letter.unsqueeze(dim=0)
                                                                    h,c                    =
torch.randn((1,self.hidden_dim),device=device),torch.randn((1,self.hidden_dim),device=device)
    for i in range(0,11):
      h,c = self.lstm(letter.type(torch.FloatTensor).to(device),(h,c))
      out = self.linear(h)
      character = torch.argmax(out)
      if character != 0:
        output.append(chr(character + 96))
      letter = torch.nn.functional.one_hot(character,num_classes=27)
      letter = letter.unsqueeze(dim=0)
    st = ""
    output = st.join(output)
    return output

  def generate_names_top5(self,letter):
    output = []
    output.append(letter)
    letter = ord(letter) - 96
    letter = torch.tensor(letter)
    letter = torch.nn.functional.one_hot(letter,num_classes=27)
    letter = letter.unsqueeze(dim=0)
                                                                    h,c                    =
torch.randn((1,self.hidden_dim),device=device),torch.randn((1,self.hidden_dim),device=device)
    TopK = 5
    for i in range(0,11):
      h,c = self.lstm(letter.type(torch.FloatTensor).to(device),(h,c))
      out = self.linear(h)
      _,idxs = out[-1].topk(TopK)
      idx = torch.randint(0,TopK-1,(1,))
      character = idxs[idx]
      if character != 0:
        output.append(chr(character + 96))
      letter = torch.nn.functional.one_hot(character,num_classes=27)
    st = ""
    output = st.join(output)
    return output

device = "cpu"
```

```
model = characterLSTM()
PATH = "/content/0702-659026651-Chintakunta"
model = torch.load(PATH,map_location=torch.device("cpu"))
model.eval()

letter = "a" # change this to generate names with different letters
for i in range(0,20):
  print(model.generate_names_argmax(letter))

for i in range(0,20):
  print(model.generate_names_top5(letter))
```