ECE 57000 Assignment 6 Exercise

Your Name: Tina Xu

Objective: Build an RNN model to predict the next character in a sequence of text data from Shakespeare's plays.

Exercise 1: Data Preprocessing (30 points)

In this part, you will implement some preprocessing functions. Run the following code to load the text data from the given file "shakespeare.txt". Do not change the random seed.

In [1]:

```
import numpy as np
! pip install unidecode
import unidecode
import string
import time
import torch
import pdb

import torch.nn as nn
from torch.autograd import Variable

all_characters = string.printable
print(all_characters)
```

```
Collecting unidecode
  Downloading Unidecode-1.3.8-py3-none-any.whl (235 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 235.5/235.5 kB 4.4 MB/s eta 0:00:00
Installing collected packages: unidecode
Successfully installed unidecode-1.3.8
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@
[\]^_`{|}~
⍰
```

Follow the step on the instructions and mount your google drive on Colab which allows to access the .txt file uploaded on your drive that was included with this assignment.

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:

```
def read_file(filename):
    file = unidecode.unidecode(open(filename).read())
    return file

dir_root = '/content/drive/MyDrive/ECE 473/Assignment-6/Assignment-6'     # Your assig
file_path = dir_root + '/shakespeare.txt'
file = read_file(file_path)
file_len = len(file)
```

```
print(f"file length: {file_len}")
print(file[:100])
```

```
file length: 1115394
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You
```

Task 1: Implement function to get a random chunk of Shakespeare text (15 points)

The `get_random_chunk` function is a helper function that generates a random chunk of **input text data** and **output text data** (which is one character shifted from the input) from the Shakespeare dataset. Specifically, the `chunk_len` argument specifies the size of the input and output sequences. For example, if `chunk_len=4`, then a valid return value would be the two chunks: `('Befo','efor')` or `('proc','roce')`. This function is useful in generating diverse sets of input data for training the RNN model in the assignment.

Hints:

- Start from a random index of the file (but note that the max index must be small enough so that a full chunk can be extracted).
- Based on this random start index, extract `chunk_len` characters for the input sequence and `chunk_len` characters for the output sequence (shifted one character to the right).

In [5]:

```
def get_random_chunk(file, rng, chunk_len = 100):
    ######### Your Code Here ###########
    max = len(file) - chunk_len - 1
    start = rng.randint(0, max)

    first = file[start:start + chunk_len]
    sec = file[start + 1:start + chunk_len + 1]

    return first, sec
    ######### End of your code #########

rng = np.random.RandomState(123) # use this if you need to generate a random sample
curr_chunk, next_chunk = get_random_chunk(file='Hello world!', rng=rng, chunk_len=10)
print(f"curr_chunk =>{curr_chunk}\n next_chunk=> {next_chunk}")

print(f"Is curr_chunk and next_chunk same length: {len(curr_chunk) == len(next_chunk)}
print(f"Is next chunk shifted by one: {curr_chunk[1:] == next_chunk[:-1]}")
```

```
curr_chunk =>Hello worl
 next_chunk=> ello world
Is curr_chunk and next_chunk same length: True
Is next chunk shifted by one: True
```

Task 2: Implement function to convert to tensors (15 points)

Define a function `to_tensor(string)` that takes a string of characters as input and return torch tensor as output, similar to in the demo in class. Specifically,

1. Create an empty tensor of shape `(len(string), 1, len(all_characters))` using the PyTorch `torch.zeros` function, where `len(string)` is the length of the input string, 1 is the batch size, and `len(all_characters)` is the total number of unique characters in the text data.
2. Loop through each character in the input string and convert it to a one-hot encoded vector.

In [6]:

```
def to_tensor(string):
######### Your Code Here ###########
  new = torch.zeros(len(string), 1, len(all_characters))
  for i in range(len(string)):
    new[i][0][all_characters.find(string[i])] = 1
  return new
######### End of your code #########

def get_one_hot_tensors(input, output):
    return to_tensor(input), to_tensor(output)

rng = np.random.RandomState(123) # use this if you need to generate a random sample
input, output = get_random_chunk(file, rng,  50)
print(input.replace('\n', ' '))
print(output.replace('\n', ' '))
input_tensor, output_tensor = get_one_hot_tensors(input, output)
print(f"input shape: {input_tensor.shape}")
print(f"output shape: {output_tensor.shape}")
```

```
g's, which Florizel I now name to you; and with sp
's, which Florizel I now name to you; and with spe
input shape: torch.Size([50, 1, 100])
output shape: torch.Size([50, 1, 100])
```

Exercise 2: Build the RNN model (30 points)

In this part, you will build the RNN model using PyTorch.

- nn.GRU is used to implement the GRU algorithm for processing sequential input data.
    - https://pytorch.org/docs/stable/generated/torch.nn.GRU.html
- The decoder layer is a fully connected neural network layer that maps the output of the GRU layer to the desired output size.
- As we are only implementing a single layer RNN, the model is not powerful enough to learn long-term dependencies in the text data. So don't be surprised if the output sentences are not very meaningful. We are providing you loss plots (`gru_loss_ex2.png`) to help you check if your code is working correctly.

In [10]:

```
import torch
import torch.nn as nn
from torch.autograd import Variable

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, n_layers=1):
        super(RNN, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers

        # Define modules of RNN
        ######### Your Code Here ###########
        # Set `self.rnn_cell` to a nn.GRU
        self.rnn_cell = nn.GRU(input_size, hidden_size, n_layers)
        # Define a linear decoder layer that maps from the hidden size to the output s
        linear = nn.Linear(hidden_size, output_size)
        self.decoder = nn.Sequential(linear)
        ######### End of your code #########

    def forward(self, input, hidden):
        ######### Your Code Here ###########
        # 1. Reshape the input to (1, 1, -1) and pass it to the GRU layer
        # 2. Reshape the rnn_cell output to (1, -1) and pass it to the decoder layer
        output, hidden = self.rnn_cell(input.view(1, 1, -1), hidden)
        output = self.decoder(output.view(1, -1))


        ######### End of your code #########
        return output, hidden

    def init_hidden(self):
        return Variable(torch.zeros(self.n_layers, 1, self.hidden_size))
```

In [8]:

```
def train(inp, target, decoder):
    hidden = decoder.init_hidden()
    decoder.zero_grad()
    loss = 0

    input_tensor, target_tensor = get_one_hot_tensors(inp, target)
    for c in range(len(inp)):
        output, hidden = decoder(input_tensor[c], hidden)
        loss += criterion(output, torch.argmax(target_tensor[c]).unsqueeze(0))

    loss.backward()
    decoder_optimizer.step()
    return loss.item() / max_length
```

In [12]:

```
def evaluate(decoder, prime_str='A', predict_len=100, temperature=0.8):
    hidden = decoder.init_hidden()
    prime_input = to_tensor(prime_str)
    predicted = prime_str

    # Use priming string to "build up" hidden state
    for p in range(len(prime_str) - 1):
        out, hidden = decoder(prime_input[p], hidden)
    inp = prime_input[-1]
    for p in range(predict_len):
        output, hidden = decoder(inp, hidden)

        # Sample from the network as a multinomial distribution
        output_dist = output.data.view(-1).div(temperature).exp()
        top_i = torch.multinomial(output_dist, 1)[0]
        # Add predicted character to string and use as next input
        predicted_char = all_characters[top_i]
        predicted += predicted_char
        inp = to_tensor(predicted_char)

    return predicted
```

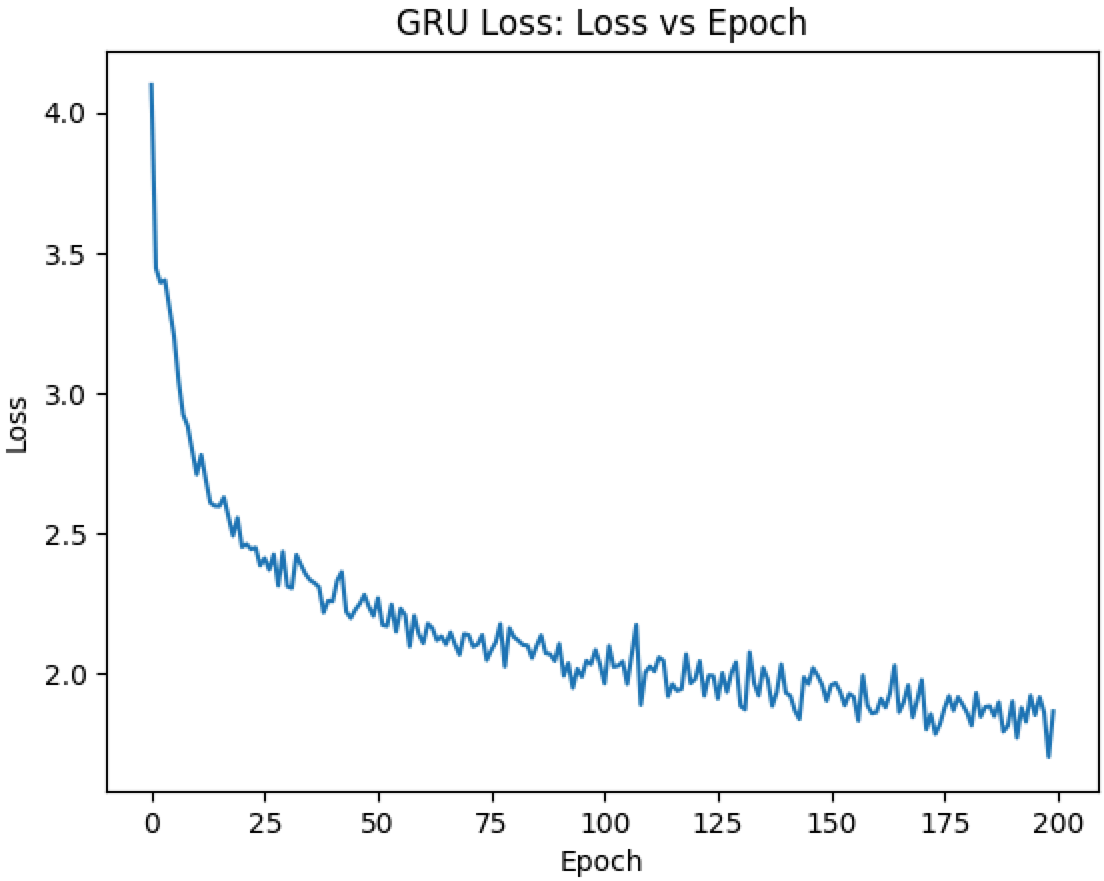
In [13]:

```
n_epochs = 2000
print_every = 100
plot_every = 10
hidden_size = 100
n_layers = 1
lr = 0.005
max_length = len(all_characters)

decoder = RNN(max_length, hidden_size, max_length)
decoder_optimizer = torch.optim.Adam(decoder.parameters(), lr=lr)
criterion = nn.CrossEntropyLoss()

start = time.time()
all_losses = []
loss_avg = 0
rng = np.random.RandomState(123) # use this if you need to generate a random sample

for epoch in range(1, n_epochs + 1):
    loss = train(*get_random_chunk(file, rng), decoder)
    loss_avg += loss

    if epoch % print_every == 0:
        print(f"[({epoch} {epoch / n_epochs * 100}%) {loss}]")
        print(evaluate(decoder, 'Wh', 100), '\n')

    if epoch % plot_every == 0:
        all_losses.append(loss_avg / plot_every)
        loss_avg = 0

print(f"______________________________________________________________")
print(evaluate(decoder, 'Th', 200, temperature=0.2))

import matplotlib.pyplot as plt
plt.plot(all_losses)
plt.title("GRU Loss: Loss vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```

```
[(100 5.0%) 2.7766888427734373]
Whheg mom pas, de s go:
y oruwtor!cl s tis rerind ee veaneeu
and su ses narrny renaw nhepca. band wand

[(200 10.0%) 2.391945037841797]
Whel le tha our rou Ret as in foud, oveded botu ardermer in I handare bllu p- aty a h
e sarand ane hoov

[(300 15.0%) 2.725008239746094]
Whld stris wriss dons the four hind stho d ard, diss she 'ds.

CICEO:
Theh she,
Sames dame thes an'y b

[(400 20.0%) 2.384757843017578]
Whour areprine,
An wang, we hiali ming- Arnd'd shast chanle omo thet,
Thart ood a thit maneid'lengsew?

[(500 25.0%) 2.136343688964844]
Whel thy ceint kng' ter ther werng I love and crint of eren,
The ples serpant for wer wha het? Whop fi

[(600 30.0%) 2.191198272705078]
Whe rucu deall and the of his cack un the to masy chat of your ine me caich surt the
thee spothead how

[(700 35.0%) 2.206630859375]
Whowst
Thou diess you domend--e mishe,
And me pre's, his and mad Onde be:
We alf my he bald thou Bedst

[(800 40.0%) 1.9939183044433593]
Whed sore sheme fore at ion, thith whand tham for foust;
Thours mared Mary the collak:
And with radd.

[(900 45.0%) 2.0678363037109375]
What you greccomed in cave: that manty
And the mastigess will becomy theme will sher hat I waRe htsero

[(1000 50.0%) 2.2002517700195314]
Whor sto be mors.

LANENLA:
Or sain, of to would 'som for no reast.

teas will is the it call she hang

[(1100 55.00000000000001%) 2.1205793762207032]
Wheree's, be you word shang.
O, bray the gore,
Ane if a fray dis the rightwe, I say cothe cingorisefay

[(1200 60.0%) 1.9496417236328125]
Whow beliot for af the done of cordest of you be, aif be the sut with souch hisill co
men fare sughted
```

```
[(1300 65.0%) 2.0815475463867186]
Why to by stay,
The do nut fortwell, your as time.

KING RETHARINA:
O rean, than will r tigh here, and

[(1400 70.0%) 1.8877171325683593]
Whillor, would buris. But all fortert way unwell your, fur our dear hear poutt I lave.

GOLZOLIOS:
You

[(1500 75.0%) 1.9531138610839844]
Which the with that has bit.

MAPEDLO:
And we cad my sine our the angstore: stis for farris;
And of th

[(1600 80.0%) 2.0112237548828125]
Whath tuck renest a ficht I the!

FLORD RICH:
Whol dine dence doubed Come in showny,
Whine then, the h

[(1700 85.0%) 1.933506622314453]
What may that this where the batselfelf I hath have
What not sun triCkes.
Shall fous our the worpal in

[(1800 90.0%) 2.1258026123046876]
Wh then, frost grower,
Our her blown bomer meries of rearns for the courd,
Whow this soor is the queen

[(1900 95.0%) 1.8345394897460938]
What know gand,
Stile no will shold.
Semsice the call healt tund pasty;
But teld you; Oxficesure! me w

[(2000 100.0%) 1.5945953369140624]
Whese so sawny But and live you?

First beliones:
O, crown aroust that Is oon mering unat
Fare may lav
```

```
That so may that me lord the parting and may that mane and the pare the parting and the pare

GLOUCESTER:
What shall the king and the beat the many to me the parter the partere the prows
To a man the pa
```



Exercise 3: Implement an LSTM model (30 points)

Using the equations from the slides in class, write your own LSTM cell module. The code below will use this instead of the GRU cell module and train the model.

Notes:

- Note that for LSTM the hidden state is really both the h_t and C_t so we just unpack the passed hidden state into these two variables at the beginning, and pack them into a tuple for returning.
- We apply a single linear layer to compute all the linear parts of the model that operate on h'_{t-1} and then unpack these using `chunk(4)` into the four separate parts. This is equivalent to having 4 separate linear layers.
- As we are only implementing a single layer RNN, the model is not powerful enough to learn long-term dependencies in the text data. So don't be surprised if the output sentences are not very meaningful. We are providing you loss plots (`lstm_loss_ex3.png`) to help you check if your code is working correctly.

In [15]:
```
class LSTMCell(nn.Module):
    def __init__(self, input_size, hidden_size, bias=True):
        super(LSTMCell, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.bias = bias

        self.xh = nn.Linear(input_size, hidden_size * 4, bias=bias)
        self.hh = nn.Linear(hidden_size, hidden_size * 4, bias=bias)
        self.reset_parameters()

    def reset_parameters(self):
        std = 1.0 / np.sqrt(self.hidden_size)
        for w in self.parameters():
            w.data.uniform_(-std, std)

    def forward(self, input, hidden=None):
        # Unpack hidden state and cell state
        hx, cx = hidden

        # Apply linear layers to input and hidden state
        linear = self.xh(input) + self.hh(hx)

        # Get outputs of applying a linear transform for each part of the LSTM
        input_linear, forget_linear, cell_linear, output_linear = linear.reshape(-1).c

        ######### Your Code Here ###########
        # 1. Apply activation functions to get gates and new cell state information
        # 2. Calculate the new cell state (c_new)
        # 3. Calculate the new hidden state (h_new)

        input = torch.sigmoid(input_linear)
        forget = torch.sigmoid(forget_linear)
        cell = torch.sigmoid(cell_linear)
        output = torch.sigmoid(output_linear)

        c_new = forget * cx + input * cell
        h_new = output * torch.tanh(c_new)
        ######### End of your code #########

        # Pack cell state $C_t$ and hidden state $h_t$ into a single hidden state tupl
        output = h_new # For LSTM the output is just the hidden state
        hidden = (h_new, c_new) # Packed h and C
        return output, hidden
```

In [17]:

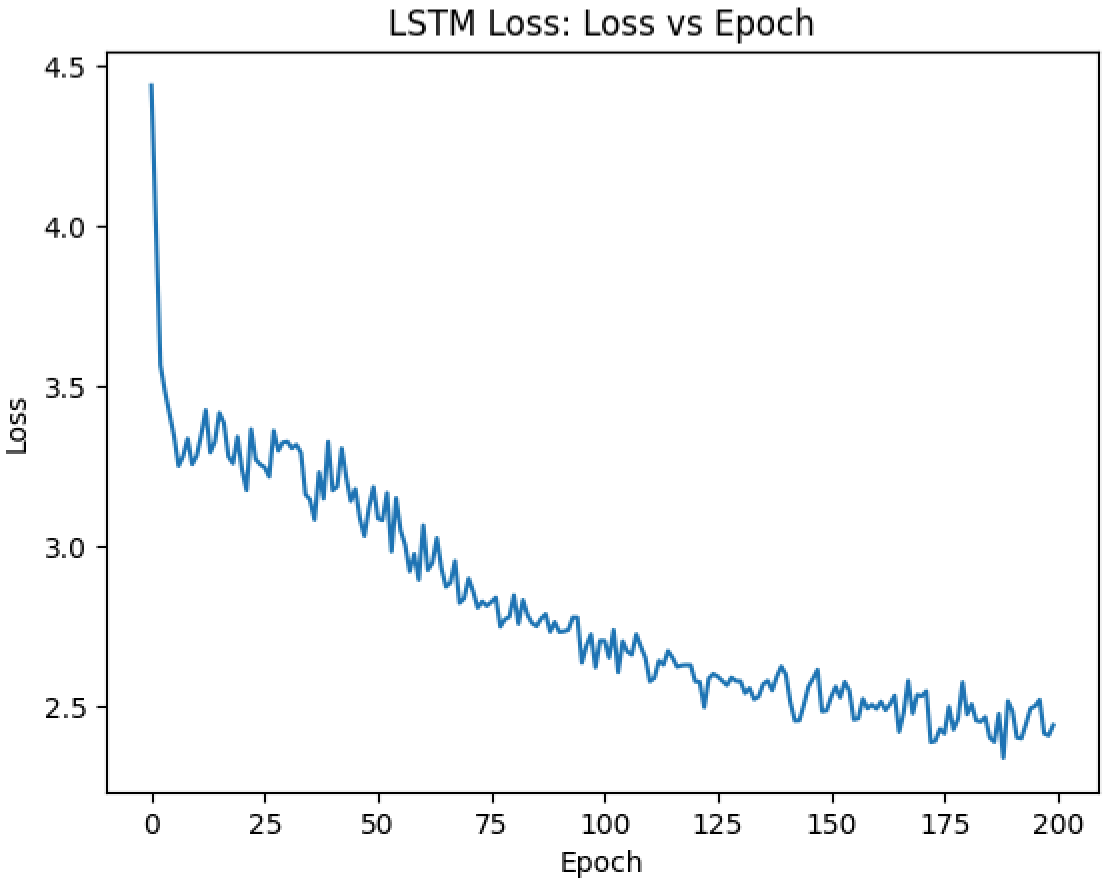
```
lr = 0.001
class LSTM_RNN(RNN):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # Replace the gru cell with LSTM cell
        self.rnn_cell = LSTMCell(max_length, hidden_size, max_length)

    def init_hidden(self):
        # LSTM cells need two hidden variables in a tuple of (h_t,C_t)
        return (Variable(torch.zeros(1, 1, self.hidden_size)), Variable(torch.zeros(1,

decoder = LSTM_RNN(max_length, hidden_size, max_length)
decoder_optimizer = torch.optim.Adam(decoder.parameters(), lr=lr)

all_losses = []
loss_avg = 0
rng = np.random.RandomState(123) # use this if you need to generate a random sample
```

```
for epoch in range(1, n_epochs + 1):
    loss = train(*get_random_chunk(file, rng),decoder)
    loss_avg += loss

    if epoch % print_every == 0:
        print(f"[({epoch} {epoch / n_epochs * 100}%) {loss}]")
        print(evaluate(decoder, 'Wh', 100), '\n')

    if epoch % plot_every == 0:
        all_losses.append(loss_avg / plot_every)
        loss_avg = 0

print(f"______________________________________________________________")
print(evaluate(decoder, 'Th', 200, temperature=0.2))

plt.plot(all_losses)
plt.title("LSTM Loss: Loss vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```

```
[(100 5.0%) 3.155955810546875]
Whmdgmevaloi  nmeh Ip on teRoTsheeta huwy
 soioyeSeab. y eno eflah gl seeGWAPs ecoen;tw o n ht aohleI:

[(200 10.0%) 3.116068115234375]
Whhd.ehss sbg dt ue ,dviice o
nIobw   oteicareisora eoc wadbedIhsm ,no losthhlheuiR ethd roereom  ech

[(300 15.0%) 3.516075439453125]
Whn',batoyega AeiWtlaTe d anenyhnha
nl air, dhno lathr v td t d A dtk  Ilaooa s
 s irs,trsn,i, ergiio

[(400 20.0%) 3.402038269042969]
Whed ypEfaobs gh h
rdi:e  nnh !hl .r urvn meao,s'y an ny od S
 fs  aong eet nu emaiseucie'esn s  l
np

[(500 25.0%) 3.0474594116210936]
Whhdt
 c
rP o
yin, mothil do
tve
:niKe
e
h telat taOad r senh fathwghevhet sitirsa
eks oad ohrhr tld

[(600 30.0%) 2.8581857299804687]
Whaei funt  y oro timns hittmy: , de gou,e
o d ay nn earu s ichttIE
 me :ionh tit spiin :e
Eome wot


[(700 35.0%) 2.9691046142578124]
Whel !o uhot inow y mnace meutI.
spins ees eo hor:d aoas see

i he ghe dvn hy tre os esDa ks or to ger

[(800 40.0%) 2.8824539184570312]
WhLbold t tee ne teno,
bnI satd eakart
,ebehl, TeAik litgy mod sothof .n tho iiaslesenynsabeme a lo yo

[(900 45.0%) 2.7433572387695313]
Whed mat nmoro thnthov she tee, ue t che t jd dnay liwanfes re
tore ney t
te tharops.I

m Sor le'e hea

[(1000 50.0%) 2.757527160644531]
Whe st
he je Hale nrl iut so peas iir ther n re wots llo d best wruot fre tond was th so fes
want snr
```

```
[(1100 55.00000000000001%) 2.734488525390625]
Whe wot rna: wf raus rot boan yise,
 perd, hoal. d pat as siit he bt yhan, Loiu
 fof theneh dot mart v

[(1200 60.0%) 2.5171849060058595]
Whows anencselare ve cisnstAof:
Th sor tin here, kand salr gos ans;
 theransort aniwl sais note as ge

[(1300 65.0%) 2.554895477294922]
Whe thet ae se toou,
she dly it woun th wane foe iu th d.

hRart il ins yotle te sd wr and  ahind n ye

[(1400 70.0%) 2.563016662597656]
Whote.
:rI S:
I dop rene nos palo il,k woumy hathantse
t thamd woof misnss cink tirsal, snc ooleve mer

[(1500 75.0%) 2.506725616455078]
WhCmend Padcing helssur nopl'd end het pi dy sois sot bareos uy moul by thinc?
A:RI
I Dof:
Ano woud
we

[(1600 80.0%) 2.7027182006835937]
Whes !o theme dou lirt am bin bt illle.


orcoo !ogon fit  foe Bond int
and the I wine th ar yol in ao

[(1700 85.0%) 2.516194305419922]
Whod by toue this harcogbeo thete coulR sones

Durne lome poritiromeo hom uum yo ingd Bou.

EGTRINANCS

[(1800 90.0%) 2.6316122436523437]
Whom than singsgon gere ther sare bou, I rulom toidous heed:
momr

TUIOWy I yor bnam son iemn mos the

[(1900 95.0%) 2.382176513671875]
Whicher ino:

Iel:
dard serepear thas ko wo pporoult mend
the the chat the ant.

hand asd thent yourd

[(2000 100.0%) 2.2701629638671874]
WhA lous parerin thy hothic talle thano sanofud shave tiyeino ten that mores goremare
```

```
r t thapine, fame

______________________________________________________________
The that there the there the ther the thar thand there thare the the the the the the
there there the there the there the the thar the the the the there the ther there the
the the ther and ther the that
```



Exercise 4: Implement your own GRU (10 points)

Same as above but implement a GRU instead of an LSTM module. An exmaple of GRU architecture can be found from the lecture slide: https://www.davidinouye.com/course/ece57000-fall-2023/lectures/recurrent-neural-networks.pdf

You output loss plot should be similar in Exercise 2.

In [18]:
```
class GRUCell(nn.Module):
    def __init__(self, input_size, hidden_size, bias=True):
        super(GRUCell, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.bias = bias

        self.h2z = nn.Linear(input_size + hidden_size, hidden_size)
        self.h2r = nn.Linear(input_size + hidden_size, hidden_size)
        self.h2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.reset_parameters()


    def reset_parameters(self):
        std = 1.0 / np.sqrt(self.hidden_size)
        for w in self.parameters():
            w.data.uniform_(-std, std)

    def forward(self, input, hx=None):
        # Inputs:
        #       input: of shape (batch_size, input_size)
        #       hx: of shape (batch_size, hidden_size)
        # Output:
        #       h_t, h_t: h_t is of shape (batch_size, hidden_size)

        if hx is None:
            hx = Variable(input.new_zeros(input.size(0), self.hidden_size))

        ######### Your Code Here ###########

        # Concatenate hidden and input to get h_prime (see torch.cat)
        combined = torch.cat((hx, input), 2)
        # Use self.h2z to calculate z_t
        z_t = torch.sigmoid(self.h2z(combined))
        # Use self.h2r to calculate r_t
        r_t = torch.sigmoid(self.h2r(combined))

        # Use Hadamard product of r_t and hx and concatenate with input
        # Then use h2h to calculate new hidden information h_tbar
        r_hx = r_t * hx
        combined_prime = torch.cat((r_hx, input), 2)
        h_t_bar = torch.tanh(self.h2h(combined_prime))
        # Update h_t with z_t, hx, and h_tbar
        h_t = (1 - z_t) * h_t_bar + z_t * hx

        ######### End of your code #########

        # Reshape h_t match input size
        h_t = h_t.reshape(1, 1, -1)

        return h_t, h_t   # Output and hidden are both h_t
```

In [19]:

```
n_epochs = 2000
print_every = 100
plot_every = 10
hidden_size = 100
n_layers = 1
lr = 0.005
max_length = len(all_characters)

# Replace the RNN module with your implemented GRUcell
class GRU_RNN(RNN):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # Replace wtih your gru cell
        self.rnn_cell = GRUCell(max_length, hidden_size, max_length)

decoder = GRU_RNN(max_length, hidden_size, max_length)
```

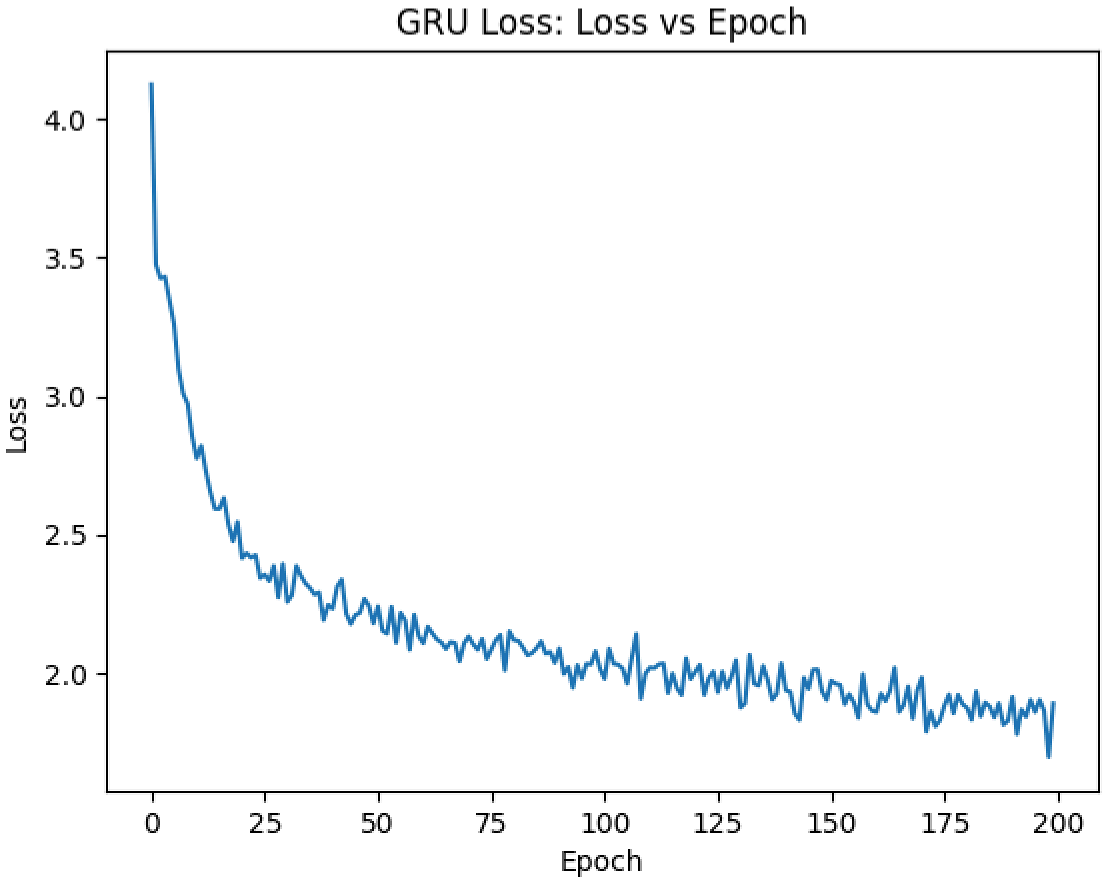
```
decoder_optimizer = torch.optim.Adam(decoder.parameters(), lr=lr)

all_losses = []
loss_avg = 0
rng = np.random.RandomState(123) # use this if you need to generate a random sample

for epoch in range(1, n_epochs + 1):
    loss = train(*get_random_chunk(file, rng),decoder)
    loss_avg += loss

    if epoch % print_every == 0:
        print(f"[({epoch} {epoch / n_epochs * 100}%) {loss}]")
        print(evaluate(decoder, 'Wh', 100), '\n')

    if epoch % plot_every == 0:
        all_losses.append(loss_avg / plot_every)
        loss_avg = 0

print(f"______________________________________________________________")
print(evaluate(decoder, 'Th', 200, temperature=0.2))

plt.plot(all_losses)
plt.title("GRU Loss: Loss vs Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```

```
[(100 5.0%) 2.8400006103515625]
Whcciro sserd io ne, see cerdecsen rise esead msd ut re taet elsi yeut no b s jato ts
to toe dentneS,


[(200 10.0%) 2.4432923889160154]
Whit' sarl brefur mis polI to ree the by ard,
I yetsoe careace tind sour for war andeas foun Wove lbul

[(300 15.0%) 2.678175964355469]
Whee btiscy thilt taif hiul ao.

Wham and sinechrey whit thoud so blery thit bureofseringers shard ind

[(400 20.0%) 2.389241943359375]
Wh enpy air ohs sindirs
I telll waso mare thas.
Youcat co dear ant mene.

PLLO:
I ENELE:
ANUD:
Il RF J

[(500 25.0%) 2.155301055908203]
Whe der,
Shal thete cofmarter.

TAThARD IENO:
Shat hre't my our wis dowhers not we mo thy Emusg and at

[(600 30.0%) 2.112471923828125]
Whall
Be that ant dut us spate the to for me nerive by ding the laing I mive; the sord in a
nd vase the

[(700 35.0%) 2.149075622558594]
What: the morem for for os mestea lut so the sonost't hos our groon.

SSCAUELO:
Hed, bat of in top the

[(800 40.0%) 2.009592742919922]
Whry mlay ald beaken me extofer: in bove at thy ioo my marine;
Thare the heach matry for me tore an at

[(900 45.0%) 2.0608779907226564]
Why cald.
Why, wath what hos ward nty son stee cuven,
I me with in he land your of sseer Cinssnost
I m

[(1000 50.0%) 2.1425775146484374]
Wh on whow, wo thou senkenits wo al bead.

Clowd in wisred
Well wath dey sir that on
And apes be and '

[(1100 55.00000000000001%) 2.1849714660644532]
```

```
Whears this no dard,
To bey sorg a prown afowiog bued ancurg.
Our a, by that
I Pary yours santer, and

[(1200 60.0%) 1.9081629943847656]
Whone withse I shoust searchord?

KINl so lave fors to he weathery monisht of your fall;
The house her

[(1300 65.0%) 2.1849012756347657]
Whiss his an his our late are, fir:
To hour's live stall
Hor he will that sir lade
Ads ane you love hi

[(1400 70.0%) 1.9304212951660156]
Wharen to to now you sid.

MARIIO:
You gout if afesty ixtred but will fails.

KING LICKI:
Gof for a go

[(1500 75.0%) 1.9460189819335938]
Whop to the path is harg?

BENVISA:
And me the weltone how the seath she make ward
Hase with a mise wi

[(1600 80.0%) 2.0056607055664064]
When the pranione such thy sing and poong;
How, my brownots, and world Lace!

LLOED SCENRY OF
BROWES:


[(1700 85.0%) 1.8857994079589844]
Whith the pritous,
Whuss to my it trespeless.

Clerstand
Sempowing all but in came to the er; thou par

[(1800 90.0%) 2.135401306152344]
Wher me be the gradiom.

Lord:
For a so hong: me lend that not meents that your for that
The shourred.

[(1900 95.0%) 1.7817237854003907]
What hath be your farth;
To sir, Wime you masty for to moof him dishore
In comemed nike, le muse the s
```

```
[(2000 100.0%) 1.6379400634765624]
Whessed is our may the mare to a thank:
Amage, no my lay as as calence's lay?

First So to my lant
Ave
```

```
The son the shall the son the son and the lay so the son and the son and the son the
lard.

SICINIUS:
What shall the san and son the can and marries the dear the son and son and sent cent
er:
And the san
```


GRU Loss: Loss vs Epoch
Loss
Epoch