Data Downloading

```
!wget --no-check-certificate --content-disposition https://raw.githubusercontent.com/Ju

!unzip language_data.zip
!rm language_data.zip
```

Generating Text with an RNN

```
!pip install unidecode
```

```
import unidecode
import string
import random
import re
import time

import torch
import torch.nn as nn

%matplotlib inline

%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
 from rnn.model import RNN
 from rnn.helpers import time_since
 from rnn.generate import generate
```

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

Data Processing

The file we are using is a plain text file. We turn any potential unicode characters into plain ASCII by using the `unidecode` package (which you can install via `pip` or `conda`).

```
all_characters = string.printable
n_characters = len(all_characters)

file_path = 'language_data/shakespeare.txt'
file = unidecode.unidecode(open(file_path).read())
# print(file[:20])
```

✓ 0s completed at 5:39 PM

```
# we will leave the last 1/10th of text as test
split = int(0.9*file_len)
train_text = file[:split]
test_text = file[split:]

print('train len: ', len(train_text))
print('test len: ', len(test_text))
```

```
file_len = 1115394
train len:  1003854
test len:  111540
```

```
chunk_len = 200

def random_chunk(text):
    start_index = random.randint(0, len(text) - chunk_len)
    end_index = start_index + chunk_len + 1
    return text[start_index:end_index]

print(random_chunk(train_text))
```

```
f will choose:
'Tis breath thou lack'st, and that breath wilt thou lose.

JOHN OF GAUNT:
Methinks I am a prophet new inspired
And thus expiring do foretell of him:
His rash fierce blaze of riot cannot
```

Input and Target data

To make training samples out of the large string of text data, we will be splitting the text into chunks.

Each chunk will be turned into a tensor, specifically a `LongTensor` (used for integer values), by looping through the characters of the string and looking up the index of each character in `all_characters`.

```
# Turn string into list of longs
def char_tensor(string):
    tensor = torch.zeros(len(string), requires_grad=True).long()
    for c in range(len(string)):
        tensor[c] = all_characters.index(string[c])
    return tensor
```

The following function loads a batch of input and target tensors for training. Each sample comes from a random chunk of text. A sample input will consist of all characters *except the last*, while the target wil contain all characters *following the first*. For example: if random_chunk='abc', then input='ab' and target='bc'

```
    target = torch.zeros(batch_size, chunk_len).long().to(device)
    for i in range(batch_size):
        start_index = random.randint(0, len(text) - chunk_len - 1)
        end_index = start_index + chunk_len + 1
        chunk = text[start_index:end_index]
        input_data[i] = char_tensor(chunk[:-1])
        target[i] = char_tensor(chunk[1:])
    return input_data, target
```

Implement model

Your RNN model will take as input the character for step t_{-1} and output a prediction for the next character t. The model should consiste of three layers - a linear layer that encodes the input character into an embedded state, an RNN layer (which may itself have multiple layers) that operates on that embedded state and a hidden state, and a decoder layer that outputs the predicted character scores distribution.

You must implement your model in the `rnn/model.py` file. You should use a `nn.Embedding` object for the encoding layer, a RNN model like `nn.RNN` or `nn.LSTM`, and a `nn.Linear` layer for the final a predicted character score decoding layer.

TODO: Implement the model in RNN `rnn/model.py`

Evaluating

To evaluate the network we will feed one character at a time, use the outputs of the network as a probability distribution for the next character, and repeat. To start generation we pass a priming string to start building up the hidden state, from which we then generate one character at a time.

Note that in the `evaluate` function, every time a prediction is made the outputs are divided by the "temperature" argument. Higher temperature values make actions more equally likely giving more "random" outputs. Lower temperature values (less than 1) high likelihood options contribute more. A temperature near 0 outputs only the most likely outputs.

You may check different temperature values yourself, but we have provided a default which should work well.

```
def evaluate(rnn, prime_str='Ab', predict_len=100, temperature=0.8):
    hidden = rnn.init_hidden(1, device=device)
    prime_input = char_tensor(prime_str)
    predicted = prime_str

    # Use priming string to "build up" hidden state
    for p in range(len(prime_str) - 1):
```

```
    inp = prime_input[-1]
    inp = inp.unsqueeze(0)
    # print(inp)
    for p in range(predict_len):
        output, hidden = rnn(inp.to(device), hidden)
        # output, hidden = rnn(inp.to(device), 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 = char_tensor(predicted_char)
        # print(char_tensor(predicted_char))

    return predicted
```

Train RNN

```
batch_size = 1
n_epochs = 7000
hidden_size = 100
n_layers = 1
learning_rate = 0.01
model_type = 'rnn'
print_every = 100
plot_every = 100
```

```
def eval_test(rnn, inp, target):
    with torch.no_grad():
        hidden = rnn.init_hidden(batch_size, device=device)
        loss = 0
        for c in range(chunk_len):
            output, hidden = rnn(inp[:,c], hidden)
            loss += criterion(output.view(batch_size, -1), target[:,c])

    return loss.data.item() / chunk_len
```

Train function

TODO: Fill in the train function. You should initialize a hidden layer representation using your RNN's

```
def train(rnn, input, target, optimizer, criterion):
    """
    Inputs:
    - rnn: model
    - input: input character data tensor of shape (batch_size, chunk_len)
    - target: target character data tensor of shape (batch_size, chunk_len)
    - optimizer: rnn model optimizer
    - criterion: loss function

    Returns:
    - loss: computed loss value as python float
    """
    loss = 0

    ####################################
    #          YOUR CODE HERE          #
    ####################################
    hidden = rnn.init_hidden(batch_size, device=device)
    # optimizer.zero_grad()
    # target.unsqueeze_(-1)
    rnn.zero_grad()
    # input = input.to(device)
    # target = target.to(device)
    for c in range(chunk_len):
      output, hidden = rnn(input[:,c], hidden)
      # print(input[:,c])
      # print("---------------------------------------- ", c)
      # loss += criterion(output.view(batch_size, -1), target[:,c])
      loss += criterion(output, target[:,c])

    loss.backward()
    optimizer.step()
    loss = loss.item() / chunk_len
    ##########       END      ##########


    return loss
    # return loss.data.item() / chunk_len
```

```
rnn = RNN(n_characters, hidden_size, n_characters, model_type=model_type, n_layers=n_
rnn_optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()

start = time.time()
all_losses = []
test_losses = []
loss_avg = 0
```

```
    test_loss_avg += test_loss

    if epoch % print_every == 0:
        print('[%s (%d %d%%) train loss: %.4f, test_loss: %.4f]' % (time_since(start)
        print(generate(rnn, 'Wh', 100, device=device), '\n')
        # print(evaluate(rnn, prime_str='Th', predict_len=100), '\n')

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

```
Training for 7000 epochs...
[0m 19s (100 1%) train loss: 2.0823, test_loss: 2.2419]
The treat mons Trow hirest fattighs ave awill seat tou cathy harest neeas, ow I ou th

[0m 39s (200 2%) train loss: 2.1141, test_loss: 2.1795]
The the in youge thought hat ard the thour the god thy my the the wisht.

KINENG VI:
Atle the to my to

[0m 58s (300 4%) train loss: 2.0046, test_loss: 2.2796]
Ths his do be me the maro
con oincts shems;
Oighted seall the le her.

KING EDI:
That that thes acll i

[1m 18s (400 5%) train loss: 2.0257, test_loss: 2.0690]
The ap shat on hie
Shapce!
As fallied
To of to forder'd say manes!

And be bunch not I shanftif practi

[1m 38s (500 7%) train loss: 2.1403, test_loss: 1.9334]
Th woat woy le that rusure.

TARCULES:
Is
You would death the hough comes wish of heal brein you rayst

[1m 57s (600 8%) train loss: 1.9966, test_loss: 1.8712]
Th wown you the falls.
```

```
[2m 37s (800 11%) train loss: 1.9156, test_loss: 1.9660]
Theson, thou day nothich thy ware you agret ware no will at mace's thy my noth this r
In t

[2m 56s (900 12%) train loss: 1.4822, test_loss: 2.1018]
The ele.

CORIOLANUS:
In owre with thas sand and to
ure in of the loth age,
```

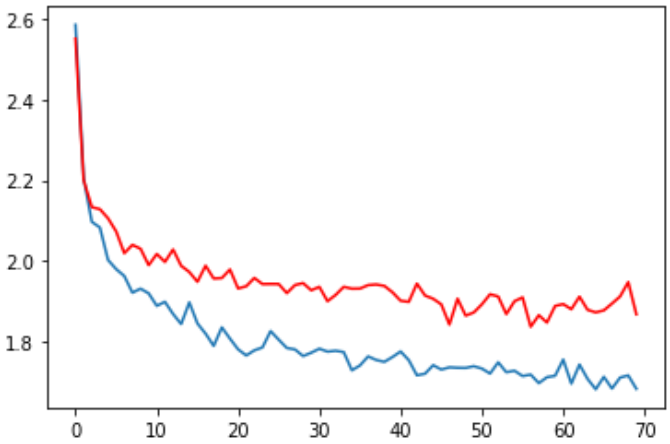
```
# save network
# torch.save(classifier.state_dict(), './rnn_generator.pth')
```

Plot the Training and Test Losses

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

plt.figure()
plt.plot(all_losses)
plt.plot(test_losses, color='r')
```

```
[<matplotlib.lines.Line2D at 0x7f1c84b3c650>]
```



Evaluate text generation

```
DUKE VINCENTIO:
A one aid be talk a redight.

GRUMIO:
More derrace and sorride my left the emen;
The other stand treath in proplear,
For blows holving's fear
And our good a mens,--
And for be hose by me to strathing eithers Richold shell not bear.

KING RICHARD III:

FROMESBLO:
And And too, dreafurede:
Your are uponied.

First God not the would, lanse
I fear stoness fall out year of in not Lark hist his law?

GRUMIO:
For majesty the pread day the and the ip death.

Go Have sprothies had;
In house me rouson part time that grues,
Ams had in a fix; what then shall he kneashd are you beged
And so speak you hear true:
A countolds, our then with s
```

Hyperparameter Tuning

Some things you should try to improve your network performance are:

- Different RNN types. Switch the basic RNN network in your model to a GRU and LSTM to compare all three.
- Try adding 1 or two more layers
- Increase the hidden layer size
- Changing the learning rate

TODO: Try changing the RNN type and hyperparameters. Record your results.