Lab 5 Report:

Create Arthur Conan Doyle AI with RNN

Name: Forest Tschirhart

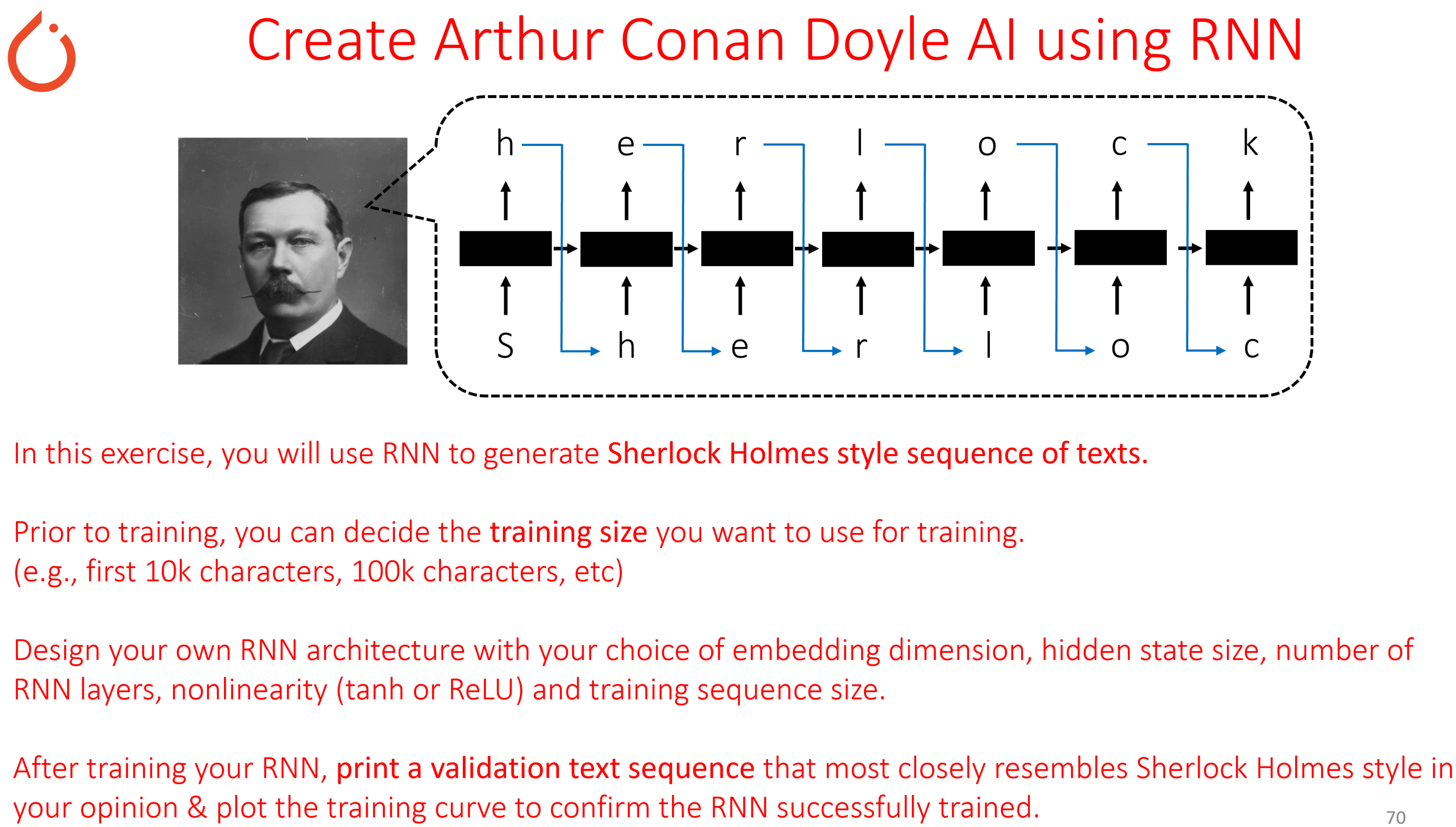
In [23]:
```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import torch
from torch.distributions import Categorical
```

In [24]:
```
from IPython.display import Image # For displaying images in colab jupyter cell
```

In [25]:
```
Image('lab5_exercise.png', width = 1000)
```

Out[25]:



Prepare Data

In [ ]:
```
# You will train on the first N characters of the Sherlock Holmes book
# Pick the size of your training data, i.e. N
data_size_to_train = 20000 # made this larger because the training sequence length is also larger than in
                           # the example

# Load the Sherlock Holmes data up to data_size_to_train

# Skip the table of contents cause its not part of the writing style
start_line = 139
```

```
# Open the file and read from the specified line
with open('sherlock.txt', 'r') as file:
    lines = [line for i, line in enumerate(file) if i >= start_line] # enumerate automatically parses
                                                                     # lines, not chars

# Join the lines and truncate to the desired size
data = ''.join(lines)[:data_size_to_train]

# Find the set of unique characters within the training data
characters = characters = sorted(list(set(data)))

# total number of characters in the training data and number of unique characters
data_size, vocab_size = len(data), len(characters)

print("Data has {} characters, {} unique".format(data_size, vocab_size))
```

```
Data has 20000 characters, 64 unique
```

In [ ]:

```
print("First 100 characters of the training data:\n", data[:100])
# making sure we skipped the table of contents
```

```
First 100 characters of the training data:
                                  PART I

               (Being a reprint from the reminiscences of
```

In [28]:

```
# Use Python Dictionary to map the characters to numbers and vice versa

character_to_num = { ch:i for i,ch in enumerate(characters) }
num_to_character = { i:ch for i,ch in enumerate(characters) }
print(character_to_num)
```

```
{'\n': 0, ' ': 1, '!': 2, '"': 3, "'": 4, '(': 5, ')': 6, ',': 7, '-': 8, '.': 9, '1': 10, '2': 11, '7': 1
2, '8': 13, ';': 14, '?': 15, 'A': 16, 'B': 17, 'C': 18, 'D': 19, 'E': 20, 'F': 21, 'G': 22, 'H': 23, 'I':
24, 'J': 25, 'L': 26, 'M': 27, 'N': 28, 'O': 29, 'P': 30, 'R': 31, 'S': 32, 'T': 33, 'U': 34, 'V': 35, 'W':
36, 'Y': 37, 'a': 38, 'b': 39, 'c': 40, 'd': 41, 'e': 42, 'f': 43, 'g': 44, 'h': 45, 'i': 46, 'j': 47, 'k':
48, 'l': 49, 'm': 50, 'n': 51, 'o': 52, 'p': 53, 'q': 54, 'r': 55, 's': 56, 't': 57, 'u': 58, 'v': 59, 'w':
60, 'x': 61, 'y': 62, 'z': 63}
```

In [ ]:

```
# Use the character_to_num dictionary to map each character in the training dataset to a number

data = list(data)
```

```
for i, ch in enumerate(data):
    data[i] = character_to_num[ch]

print(data[:40]) # making sure we mapped the characters to numbers correctly
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 30,
16, 31, 33, 1, 24]
```

Define Model

In [30]:

```
class CharRNN(torch.nn.Module):

    def __init__(self, num_embeddings, embedding_dim, input_size, hidden_size, num_layers, output_size):

        super(CharRNN, self).__init__()

        self.embedding = torch.nn.Embedding(num_embeddings, embedding_dim)
        self.rnn = torch.nn.RNN(input_size=input_size, hidden_size=hidden_size,
                                num_layers=num_layers,
                                nonlinearity = 'relu')
        self.decoder = torch.nn.Linear(hidden_size, output_size)

    def forward(self, input_seq, hidden_state):

        # Forward pass input sequence to embedding layer
        embedding = self.embedding(input_seq)

        # RNN cell takes output of embedding layer + previous hidden state as inputs
        output, hidden_state = self.rnn(embedding, hidden_state)

        # Forward pass the RNN cell output to decoder to get the probabilities
        output = self.decoder(output)

        # hidden states need to be detached from computation graph to be re-used as input
        return output, hidden_state.detach()
```

Define Hyperparameters

In [ ]:
```
# Fix random seed
torch.manual_seed(25)

# Define RNN network
# embedding_dim = 100 is relatively small because the high dimensionality capturing
# relationships between words is not needed when dealing only with characters.
# input_size = 100 is the size of the embedding layer, these must match.
# hidden_size = 1024 I expanded the size of the hidden layer to 1024 to allow for more
# complex relationships, expecially since we are going character by character
rnn = CharRNN(num_embeddings = vocab_size, embedding_dim = 100,
              input_size = 100, hidden_size = 1024, num_layers = 3,
              output_size = vocab_size)

# Define learning rate and epochs
# chose a slow learning rate so that model could converge towards end of training
learning_rate = 0.0005
epochs = 100 # need a lot of epochs for the loss to converge
loss_target = 0.035 # set a target loss to stop training when reached

# Size of the input sequence to be used during training and validation
training_sequence_len = 200 # this many characters is needed to capture the style specifically.
                            # Any shorter and we would not be able to capture context within a sentence
                            # of standard length.
validation_sequence_len = 200

# Define loss function and optimizer
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)

# add .cuda() for GPU acceleration
rnn
```

Out[ ]:
```
CharRNN(
  (embedding): Embedding(64, 100)
  (rnn): RNN(100, 1024, num_layers=3)
  (decoder): Linear(in_features=1024, out_features=64, bias=True)
)
```

Identify Tracked Values

In [32]:
```
# Tracking training loss per each input/target sequence fwd/bwd pass
train_loss_list = []
```

Train Model

In [ ]:
```
# Convert training data into torch tensor and make it into vertical orientation (N, 1)
# Attach .cuda() if using GPU
data = torch.unsqueeze(torch.tensor(data), dim = 1)

# Training Loop ------------------------------------------------------------------------------------------

for epoch in range(epochs):

    # Randomly select a starting character from first 100 characters in training set
    character_loc = np.random.randint(100)

    # iteration number to keep track of until the sequence reaches the end of training data
    iteration = 0

    # initialize initial hiddens state as None
    hidden_state = None

    # loop continues until target_seq reaches end of the data
    while character_loc + training_sequence_len + 1 < data_size:

        # Define input/target sequence
        input_seq = data[character_loc : character_loc + training_sequence_len]
        target_seq = data[character_loc + 1 : character_loc + training_sequence_len + 1]
        # using teacher forcing here

        # Pass input sequence and hidden_state to RNN
        output, hidden_state = rnn(input_seq, hidden_state)

        # Compute loss between RNN output sequence vs target sequence
        # torch.squeeze removes the column dimension and make them into horizontal orientation
        loss = loss_fn(torch.squeeze(output), torch.squeeze(target_seq))

        # Append loss
        train_loss_list.append(loss.item())
```

```
        # Empty gradient buffer -> backpropagation -> update network
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Update starting character for next sequence
        character_loc += training_sequence_len

        # Update iteration number
        iteration += 1

    epoch_avg_loss = np.mean(train_loss_list[-iteration:]) # need this later
    print("Averaged Training Loss for Epoch ", epoch,": ", epoch_avg_loss)

    # Sample and generate a text sequence after every epoch ----------------------------------------------

    #Initialize character location and hidden state for validation
    character_loc = 0
    hidden_state = None

    # Pick a random character from the dataset as an initial input to RNN
    rand_index = np.random.randint(data_size-1)
    input_seq = data[rand_index : rand_index+1]

    print("----------------------------------------")
    with torch.no_grad():

        # Loop continues until RNN generated sequence is in desired length
        while character_loc < validation_sequence_len:

            # Pass validation sequence to RNN
            # Note that RNN now uses its previous output character as input
            output, hidden_state = rnn(input_seq, hidden_state)

            # Take the softmax of the decoder output to get the probabilities of predicted characters
            output = torch.nn.functional.softmax(torch.squeeze(output), dim=0)
            # Use the  probabilities to sample the output character
            character_distribution = torch.distributions.Categorical(output)
            character_num = character_distribution.sample()

            # Convert the character number selected from sampling to actual character and print
```

```
            print(num_to_character[character_num.item()], end='')

            # Update the input_seq so that it's using the output of the RNN as new input
            input_seq[0][0] = character_num.item()

            # Update the character location
            character_loc += 1

    print("\n----------------------------------------")

    # Break statement to grab best final model state. Once averaged loss is less than the target loss
    # stop training, BUT only if the actual loss is not more than the averaged loss
    # (makes sure we aren't on a noise spike)
    if (train_loss_list[-1] <= epoch_avg_loss) and (epoch_avg_loss < loss_target):
        break
```

```
Averaged Training Loss for Epoch  0 :  2.9806876616044478
----------------------------------------
 werees lif cak we in op letr ortetle cnd

 fond, ar the gecv comind ton lat be hy wome ns of-lginh and hovaind wend usr.er. ax-re checinter gonledesl
ein tin
 cozser th phicacccing Hs Theinis la, oote
----------------------------------------
Averaged Training Loss for Epoch  1 :  2.284456568534928
----------------------------------------
peld atger atcheticand
 mescente and sounl, hourduncahan. wiwh reconlaniocen lioch ov icher whad breiciel
 chogrharlice
 ahe whed minale ilan su tulkedviereg, Nre ceet he he beinarhes dech mlone tare,
----------------------------------------
Averaged Training Loss for Epoch  2 :  2.053362660937839
----------------------------------------
-frropon ofmecceioncorked wisterve
 orson whas'nd Hoen xowe
 hime'reing recailinen''badion
 comvound me,
 had must inther.



iones. Oh yey laar
 Seppean Sopetally

 Hiss-Tping pittres, shir my rojeu
----------------------------------------
Averaged Training Loss for Epoch  3 :  1.8925096988677979
----------------------------------------
ow dictens, whefhicothome ofty, pricey. Lould notice I rare of burales stecrofoliotid li-prasssinain
 hourfther hims hush when at his whothing have in now
 weodevess oper a presJtorierted you prear
 p
----------------------------------------
Averaged Training Loss for Epoch  4 :  1.7638005752756138
----------------------------------------
hotebye.

 Wowtt me
```

```
----------------------------------------
Averaged Training Loss for Epoch  91 :  0.042331864605798866
----------------------------------------
                 (Bemade, too," remarked Sherlock Holmes seemed delighted at the idea of sharing his rooms
 with me. "I things out whe had neither kith nor kin in England, and was therefore as free as
----------------------------------------
Averaged Training Loss for Epoch  92 :  0.04150348550856414
----------------------------------------
ow
 could I meet this friend of yours?"

 "He is sure to be at the laboratory," returned my companiones who was bemoaning himself this cross-examina
tion. "I keep a bull pup," I said,
 mand I object to
----------------------------------------
Averaged Training Loss for Epoch  93 :  0.030371201958394413
----------------------------------------
ever.

 "Woulmory of rows?" he asked,
 anxiously.

 "It depends on the shoulder, and
 turning round I recognized young Stamford, who had been a dresser and settle
 everything," hu answered.

 "That's
----------------------------------------
```

Visualize & Evaluate Model

In [ ]:
```python
# Print a validation text sequence that most closely resembles Sherlock Holmes style
test_sequence_len = 500
#Initialize character location and hidden state for validation
character_loc = 0
hidden_state = None

# Pick a random character from the dataset as an initial input to RNN
rand_index = np.random.randint(data_size-1)
input_seq = data[rand_index : rand_index+1]
```

```
with torch.no_grad():

    while character_loc < test_sequence_len: # Loop continues until RNN generated sequence desired length

        # Pass validation sequence to RNN
        # Note that RNN now uses its previous output character as input
        output, hidden_state = rnn(input_seq, hidden_state)

        # Take the softmax of the decoder output to get the probabilities of predicted characters
        output = torch.nn.functional.softmax(torch.squeeze(output), dim=0)
        # Use the  probabilities to sample the output character
        character_distribution = torch.distributions.Categorical(output)
        character_num = character_distribution.sample()

        # Convert the character number selected from sampling to actual character and print
        print(num_to_character[character_num.item()], end='')

        # Update the input_seq so that it's using the output of the RNN as new input
        input_seq[0][0] = character_num.item()

        # Update the character location
        character_loc += 1
```

```
    Joll was My health forbade me from venturing out unless the weather
was exceptionally genial, and I had no friends who would call upon me
and break the monotony of my daily existence. Under these
circumstances, I eagerly hailed the little mystery which hung around
my companion, and spent much of my time in endeavouring to unravel
it.

He was not studying medicine. He had himself, in reply to a question,
confirmed Stamford's opinion upon that point. Neither did he appear
to have purs
```

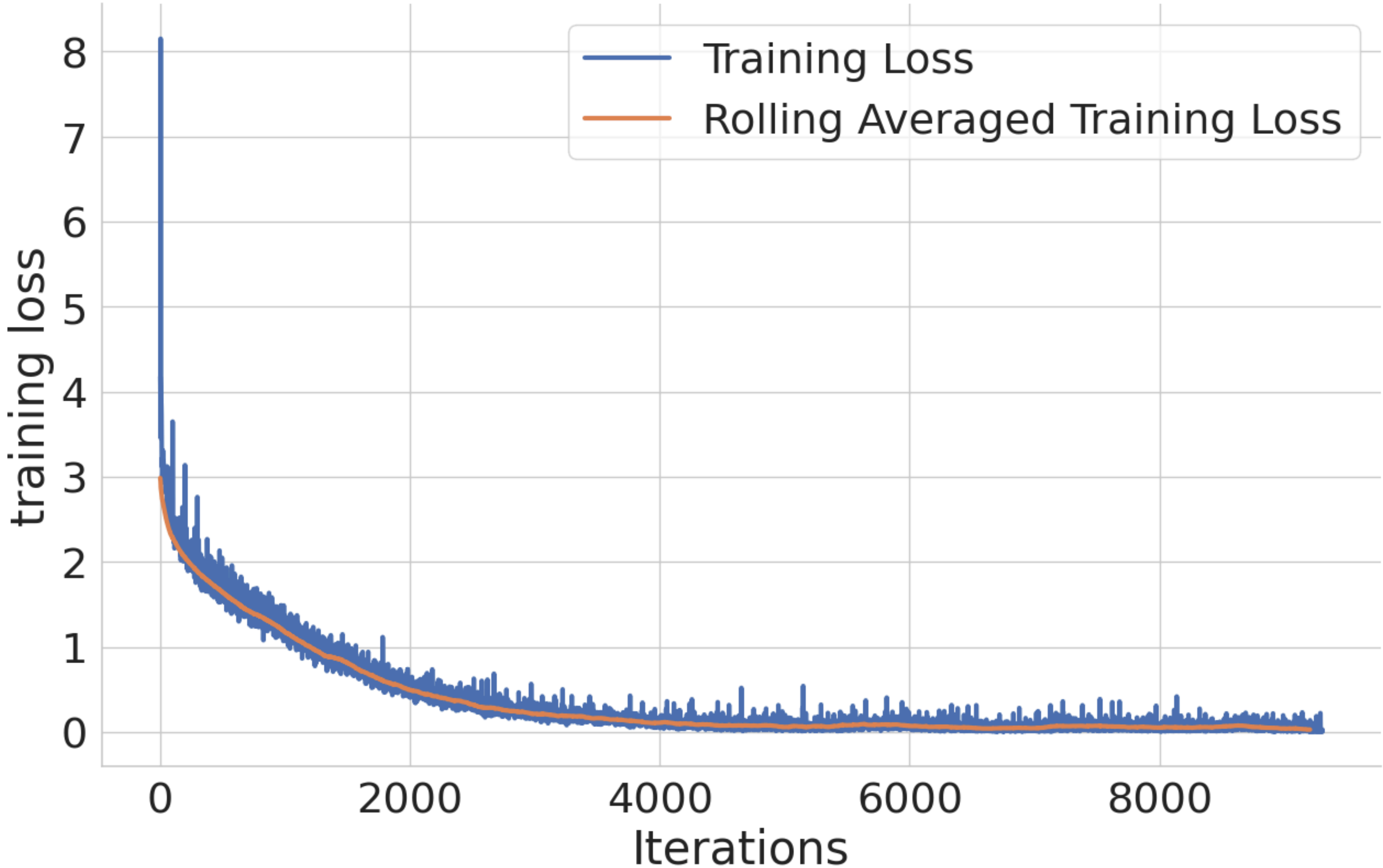
In [44]:
```
# Import seaborn for prettier plot
import seaborn as sns

sns.set(style = 'whitegrid', font_scale = 2.5)
```

In [45]:
```
# Plot the training loss and rolling mean training loss with respect to iterations
# Feel free to change the window size
```

```
plt.figure(figsize = (15, 9))

plt.plot(train_loss_list, linewidth = 3, label = 'Training Loss')
plt.plot(np.convolve(train_loss_list, np.ones(100), 'valid') / 100,
         linewidth = 3, label = 'Rolling Averaged Training Loss')
plt.ylabel("training loss")
plt.xlabel("Iterations")
plt.legend()
sns.despine()
```



In [ ]:

```

```