In [1]:
```python
from nltk.tokenize import wordpunct_tokenize
import torch
import torch.nn as nn
import torch.optim as optim
from tqdm import tqdm
```

PART B

In [2]:
```python
# Read and preprocess the data
with open("shakespeare.txt", 'r') as file:
    data = file.read().lower()

chars = sorted(list(set(data)))
data_size, vocab_size = len(data), len(chars)
print("----------------------------------------")
print("Data has {} characters, {} unique".format(data_size, vocab_size))
print("----------------------------------------")

# Char to index and index to char maps
char_to_ix = {ch: i for i, ch in enumerate(chars)}
ix_to_char = {i: ch for i, ch in enumerate(chars)}

# Convert data from chars to indices
data = [char_to_ix[ch] for ch in data]


class RNN(nn.Module):
    def __init__(self, input_size, embedding_size, hidden_size, num_layers, output_size)
        super(RNN, self).__init__()
        self.embedding = nn.Embedding(input_size, embedding_size)
        self.rnn = nn.LSTM(embedding_size, hidden_size, num_layers=num_layers)
        self.decoder = nn.Linear(hidden_size, output_size)

    def forward(self, input_seq, hidden_state=None):
        embedding = self.embedding(input_seq)
        output, hidden_state = self.rnn(embedding.view(len(input_seq), 1, -1), hidden_st
        output = self.decoder(output.view(len(input_seq), -1))
        return output, hidden_state
# Set up the model and training parameters
embedding_size = 64
hidden_size = 64
num_layers = 1
model = RNN(vocab_size, embedding_size, hidden_size, num_layers, vocab_size)
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())
```

```
----------------------------------------
Data has 98029 characters, 48 unique
----------------------------------------
```

In [97]:
```python
epochs = 5
sequence_length = 40
step_size = 10

for i_epoch in range(1, epochs + 1):
    n = 0
    running_loss = 0

    with tqdm(range(sequence_length, len(data) - 1, step_size)) as pbar:
        for i in pbar:
            hidden_state = None
            input_seq = torch.tensor(data[i - sequence_length: i])
            target_seq = torch.tensor(data[i - sequence_length + 1: i + 1])

            # Forward pass
            output, _ = model(input_seq, hidden_state)

            # Compute loss
            loss = loss_fn(torch.squeeze(output), torch.squeeze(target_seq))
            running_loss += loss.item()
            n += 1

            # Compute gradients and take optimizer step
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Update progress bar description
            pbar.set_description(f"Epoch: {i_epoch}")
            pbar.set_postfix(loss=running_loss / n)

    # Print loss after every epoch
    print("Epoch: {0}\tLoss: {1:.8f}".format(i_epoch, running_loss / n))
```

```
Epoch: 1: 100%|██████████| 9799/9799 [04:22<00:00, 37.36it/s, loss=1.95]
Epoch: 1        Loss: 1.95346724
Epoch: 2: 100%|██████████| 9799/9799 [04:02<00:00, 40.39it/s, loss=1.72]
Epoch: 2        Loss: 1.72336277
Epoch: 3: 100%|██████████| 9799/9799 [03:53<00:00, 42.03it/s, loss=1.66]
Epoch: 3        Loss: 1.65716395
Epoch: 4: 100%|██████████| 9799/9799 [03:45<00:00, 43.42it/s, loss=1.62]
Epoch: 4        Loss: 1.61786130
Epoch: 5: 100%|██████████| 9799/9799 [04:11<00:00, 38.94it/s, loss=1.59]
Epoch: 5        Loss: 1.59191297
```

In [98]:

```
# Generate poems
def generate_poem(seed, temperature):
    with torch.no_grad():
        model.eval()
        hidden = None
        poem = seed

        for _ in range(200):
            input_seq = torch.tensor([char_to_ix[ch] for ch in seed])
            output, hidden = model(input_seq, hidden)
            output = output[-1, :] / temperature
            output = torch.softmax(output, dim=0)
            char_idx = torch.multinomial(output, num_samples=1).item()
            char = ix_to_char[char_idx]
            poem += char
            seed = seed[1:] + char

        return poem

# Generate poems with different temperatures
seed = "shall i compare thee to a summer's day?\n"
temperatures = [1.5, 0.75, 0.25]

for temp in temperatures:
    poem = generate_poem(seed, temp)
    print("Temperature: {}\nPoem:\n{}\n".format(temp, poem))
```

```
Temperature: 1.5
Poem:
shall i compare thee to a summer's day?
```

```
yet eat mid hyfulighos the hild
if ofeenest.1
 nyow poan chottand,
the negion me:
my myrater'st.
wheshel,
but mey,
with formmehy must thee me uffeence.wates con
thy useath sayong:
who harth deare. add

Temperature: 0.75
Poem:
shall i compare thee to a summer's day?
is my cound so sines frult and thee, and from frame lead?
in hate,
come i call,
which frace the best is strangle sick she by what i a can proved my fire doth a date.




Temperature: 0.25
Poem:
shall i compare thee to a summer's day?
which but despire the many doth by fire the be the be by fair my beauty the many the bar
e the bare by of my hearth from the be of the by the be by the bare lies and a faired ha
te,
  but the lies the b
```

PART C

In [24]:

```python
class RNN(nn.Module):
    def __init__(self, input_size, embedding_size, hidden_size, num_layers, output_size)
        super(RNN, self).__init__()
        self.embedding = nn.Embedding(input_size, embedding_size)
        self.rnn = nn.LSTM(embedding_size, hidden_size, num_layers=num_layers)
        self.decoder = nn.Linear(hidden_size, output_size)

    def forward(self, input_seq, hidden_state=None):
        embedding = self.embedding(input_seq)
        output, hidden_state = self.rnn(embedding.view(len(input_seq), 1, -1), hidden_st
        output = self.decoder(output.view(len(input_seq), -1))
        return output, hidden_state

# Set up the model and training parameters
embedding_size = 128
hidden_size = 128
num_layers = 2
model = RNN(vocab_size, embedding_size, hidden_size, num_layers, vocab_size)
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

epochs = 15 # 7 hours
sequence_length = 40
step_size = 5

for i_epoch in range(1, epochs + 1):
    n = 0
    running_loss = 0

    with tqdm(range(sequence_length, len(data) - 1, step_size)) as pbar:
        for i in pbar:
            hidden_state = None
            input_seq = torch.tensor(data[i - sequence_length: i])
            target_seq = torch.tensor(data[i - sequence_length + 1: i + 1])

            # Forward pass
            output, _ = model(input_seq, hidden_state)

            # Compute loss
            loss = loss_fn(torch.squeeze(output), torch.squeeze(target_seq))
            running_loss += loss.item()
            n += 1

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            pbar.set_description(f"Epoch: {i_epoch}")
            pbar.set_postfix(loss=running_loss / n)

    # Print loss after every epoch
    print("Epoch: {0}\tLoss: {1:.8f}".format(i_epoch, running_loss / n))
```

```
Epoch: 1: 100%|██████████| 6441/6441 [18:54<00:00,  5.68it/s, loss=5.98]
Epoch: 1        Loss: 5.97565511
Epoch: 2: 100%|██████████| 6441/6441 [18:33<00:00,  5.78it/s, loss=5.13]
Epoch: 2        Loss: 5.13261113
Epoch: 3: 100%|██████████| 6441/6441 [18:37<00:00,  5.76it/s, loss=4.67]
Epoch: 3        Loss: 4.66553846
Epoch: 4: 100%|██████████| 6441/6441 [22:17<00:00,  4.82it/s, loss=4.2]
Epoch: 4        Loss: 4.20117602
Epoch: 5: 100%|██████████| 6441/6441 [22:06<00:00,  4.86it/s, loss=3.74]
Epoch: 5        Loss: 3.73910171
Epoch: 6: 100%|██████████| 6441/6441 [25:30<00:00,  4.21it/s, loss=3.29]
Epoch: 6        Loss: 3.28800035
Epoch: 7: 100%|██████████| 6441/6441 [25:06<00:00,  4.27it/s, loss=2.91]
Epoch: 7        Loss: 2.91348283
Epoch: 8: 100%|██████████| 6441/6441 [22:41<00:00,  4.73it/s, loss=2.57]
Epoch: 8        Loss: 2.56538899
Epoch: 9: 100%|██████████| 6441/6441 [23:16<00:00,  4.61it/s, loss=2.3]
Epoch: 9        Loss: 2.29831699
Epoch: 10: 100%|██████████| 6441/6441 [21:18<00:00,  5.04it/s, loss=2.06]
Epoch: 10       Loss: 2.06350198
Epoch: 11: 100%|██████████| 6441/6441 [18:26<00:00,  5.82it/s, loss=1.84]
Epoch: 11       Loss: 1.83815795
Epoch: 12: 100%|██████████| 6441/6441 [18:17<00:00,  5.87it/s, loss=1.64]
Epoch: 12       Loss: 1.64285745
Epoch: 13: 100%|██████████| 6441/6441 [18:33<00:00,  5.79it/s, loss=1.46]
Epoch: 13       Loss: 1.45635814
Epoch: 14: 100%|██████████| 6441/6441 [19:40<00:00,  5.46it/s, loss=1.3]
Epoch: 14       Loss: 1.29959000
Epoch: 15: 100%|██████████| 6441/6441 [18:35<00:00,  5.77it/s, loss=1.16]
Epoch: 15       Loss: 1.16426462
```

In [25]:

```
def generate_poem(model, start_string, length, temperature=1.0):
    model.eval()


    start_tokens = word_tokenize(start_string.lower())
    input_seq = torch.tensor([word_to_ix[token] for token in start_tokens])
```

```python
    generated = start_string

    with torch.no_grad():
        for i in range(length):
            output, _ = model(input_seq)
            output = output[-1]
            output = torch.nn.functional.softmax(output / temperature, dim=0)
            predicted_index = torch.multinomial(output, 1).item()
            generated += ' ' + ix_to_word[predicted_index]
            input_seq = torch.cat([input_seq, torch.tensor([predicted_index])])

    return generated

seed = "shall i compare thee to a summer's day?\n"
generated_length = 100 # You can adjust this value

print("Generated poem with temperature 1.5:")
print(generate_poem(model, seed, generated_length, temperature=1.5))

print("Generated poem with temperature 0.75:")
print(generate_poem(model, seed, generated_length, temperature=0.75))

print("Generated poem with temperature 0.25:")
print(generate_poem(model, seed, generated_length, temperature=0.25))
```

```
Generated poem with temperature 1.5:
shall i compare thee to a summer's day?
 seeing him i did glance , in which unspotted would straight her most all comfortless ,
to which beholding their bane : let when ye most others are which can wont and though th
ey could know themselves as willing at last ye deigned , that little not deem and summer
's 125 did form enough fear to envy , are both i did leave , in which your worth do obst
inate in cause did pleasauns ; ' was of hideous being kind , and they golden fill . thou
the both thereof with lov'st note to contentment false spark ,
Generated poem with temperature 0.75:
shall i compare thee to a summer's day?
 i should approach . for i have with whose hand remain . that is true love doth from mak
e him which to steel , which with mine eyes i mean , and under with heavenly aught , the
joyous sight or it . let them have i hope of winter and horrid . when when my joy as har
dly have it be the day , on the year 's sins forepast let us leave , and in my soul was
ravished made old . with precious like conceive , that honour alive one spark of filthy
lustfull fire of thine
Generated poem with temperature 0.25:
shall i compare thee to a summer's day?
 i should approach . so let us fair , which like a sovereign love to there i being fire
: and tell her good one and perfect pleasure from her too bower . so when mine eyes i th
ereunto direct , is ready , that may admire the sky . but let none return did the joyous
safety of filthy lustfull fire ne one light glance of sensual desire attempt to work her
gentle mind 's unrest . but pure affections bred in spotless breast , and modest thought
s breathed from well tempered sprites go visit her eyes do blind
```

```
In [ ]:
```