final_project_sherlock

June 14, 2024

[1]:
```
import string
import random
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
```

Prepare for Dataset

[2]:
```
file_path = './sherlock.txt'

with open(file_path, 'r') as f:
    file = f.read()

all_chars = set(file)
all_chars.update(set(string.printable))
all_chars = sorted(all_chars)
n_chars = len(all_chars)
file_len = len(file)

print('Length of file: {}'.format(file_len))
print('All possible characters: {}'.format(all_chars))
print('Number of all possible characters: {}'.format(n_chars))
```

```
Length of file: 3381928
All possible characters: ['\t', '\n', '\x0b', '\x0c', '\r', ' ', '!', '"', '#',
'$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3',
'4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C',
'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c',
'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's',
't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '£', '°', '½', 'ß', 'à',
'â', 'è', 'é', 'ê', 'î', 'ñ', 'ô', 'ö', 'û', 'ü', '’']
Number of all possible characters: 116
```

[3]:
```
# Get a random sequence of the Shakespeare dataset.
def get_random_seq():
    seq_len     = 128  # The length of an input sequence.
    start_index = random.randint(0, file_len - seq_len)
```

```
    end_index   = start_index + seq_len + 1
    return file[start_index:end_index]

# Convert the sequence to one-hot tensor.
def seq_to_onehot(seq):
    n_chars = len(all_chars)
    tensor  = torch.zeros(len(seq), 1, n_chars)
    # Here we use batch size = 1 and classes = number of unique characters.
    for t, char in enumerate(seq):
        try:
            index = all_chars.index(char)
            tensor[t][0][index] = 1
        except ValueError:
            print(f"Character '{char}' not found in all_chars.")
            raise
    return tensor


# Convert the sequence to index tensor.
def seq_to_index(seq):
    tensor = torch.zeros(len(seq), 1)
    # Shape of the tensor:
    #     (sequence length, batch size).
    # Here we use batch size = 1.
    for t, char in enumerate(seq):
        tensor[t] = all_chars.index(char)
    return tensor

# Sample a mini-batch including input tensor and target tensor.
def get_input_and_target():
    seq    = get_random_seq()
    input1  = seq_to_onehot(seq[:-1])      # Input is represented in one-hot.
    target = seq_to_index(seq[1:]).long() # Target is represented in index.
    return input1, target
```

Choose a Device

[4]:
```
# If there are GPUs, choose the first one for computing. Otherwise use CPU.
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
# If 'cuda:0' is printed, it means GPU is available.
```

```
cuda:0
```

Network Definition

[5]:
```
class Net(nn.Module):
    def __init__(self):
```

```
        # Initialization.
        super(Net, self).__init__()
        self.input_size  = n_chars   # Input size: Number of unique chars.
        self.hidden_size = 100       # Hidden size: 100.
        self.output_size = n_chars   # Output size: Number of unique chars.

        self.rnn = nn.RNNCell(input_size=self.input_size, hidden_size=self.
↪hidden_size, bias=False)
        self.linear = nn.Linear(in_features=self.hidden_size, out_features=self.
↪output_size, bias=False)

    def forward(self, input, hidden):
        """ Forward function.
              input:  One-hot input. It refers to the x_t in homework write-up.
              hidden: Previous hidden state. It refers to the h_{t-1}.
            Returns (output, hidden) where output refers to y_t and
                     hidden refers to h_t.
        """
        # Forward function.
        hidden = self.rnn(input, hidden)
        output = self.linear(hidden)

        return output, hidden

    def init_hidden(self):
        # Initial hidden state.
        # 1 means batch size = 1.
        return torch.zeros(1, self.hidden_size).to(device)

net = Net()     # Create the network instance.
net.to(device)  # Move the network parameters to the specified device.
```

[5]:

```
Net(
  (rnn): RNNCell(116, 100, bias=False)
  (linear): Linear(in_features=100, out_features=116, bias=False)
)
```

Training Step and Evaluation Step

[6]:

```
# Training step function.
def train_step(net, opt, input, target):
    """ Training step.
        net:    The network instance.
        opt:    The optimizer instance.
        input:  Input tensor.  Shape: [seq_len, 1, n_chars].
        target: Target tensor. Shape: [seq_len, 1].
    """
```

```
    seq_len = input.shape[0]     # Get the sequence length of current input.
    hidden = net.init_hidden()  # Initial hidden state.
    net.zero_grad()             # Clear the gradient.
    loss = 0                    # Initial loss.

    for t in range(seq_len):    # For each one in the input sequence.
        output, hidden = net(input[t], hidden)
        loss += loss_func(output, target[t])

    loss.backward()             # Backward.
    opt.step()                  # Update the weights.

    return loss / seq_len       # Return the average loss w.r.t sequence length.
```

[7]:
```
# Evaluation step function.
def eval_step(net, init_seq='W', predicted_len=100):
    # Initialize the hidden state, input and the predicted sequence.
    hidden        = net.init_hidden()
    init_input    = seq_to_onehot(init_seq).to(device)
    predicted_seq = init_seq

    # Use initial string to "build up" hidden state.
    for t in range(len(init_seq) - 1):
        output, hidden = net(init_input[t], hidden)

    # Set current input as the last character of the initial string.
    input = init_input[-1]

    # Predict more characters after the initial string.
    for t in range(predicted_len):
        # Get the current output and hidden state.
        output, hidden = net(input, hidden)

        # Sample from the output as a multinomial distribution.
        predicted_index = torch.multinomial(output.view(-1).exp(), 1)[0]

        # Add predicted character to the sequence and use it as next input.
        predicted_char  = all_chars[predicted_index]
        predicted_seq  += predicted_char

        # Use the predicted character to generate the input of next round.
        input = seq_to_onehot(predicted_char)[0].to(device)

    return predicted_seq
```

Training Procedure

[8]:
```python
# Number of iterations.
# NOTE: You may reduce the number of training iterations if the training takes␣
↪long.
iters       = 100000  # Number of training iterations.
print_iters = 5000    # Number of iterations for each log printing.

# The loss variables.
all_losses = []
loss_sum   = 0

# Initialize the optimizer and the loss function.
opt       = torch.optim.Adam(net.parameters(), lr=0.005)
loss_func = nn.CrossEntropyLoss()

# Training procedure.
for i in range(iters):
    input, target = get_input_and_target()            # Fetch input and target.
    input, target = input.to(device), target.to(device) # Move to GPU memory.
    loss      = train_step(net, opt, input, target)   # Calculate the loss.
    loss_sum += loss                                  # Accumulate the loss.

    # Print the log.
    if i % print_iters == print_iters - 1:
        print('iter:{}/{} loss:{}'.format(i, iters, loss_sum / print_iters))
        print('generated sequence: {}\n'.format(eval_step(net)))

        # Track the loss.
        all_losses.append(float(loss_sum) / print_iters)
        loss_sum = 0
```

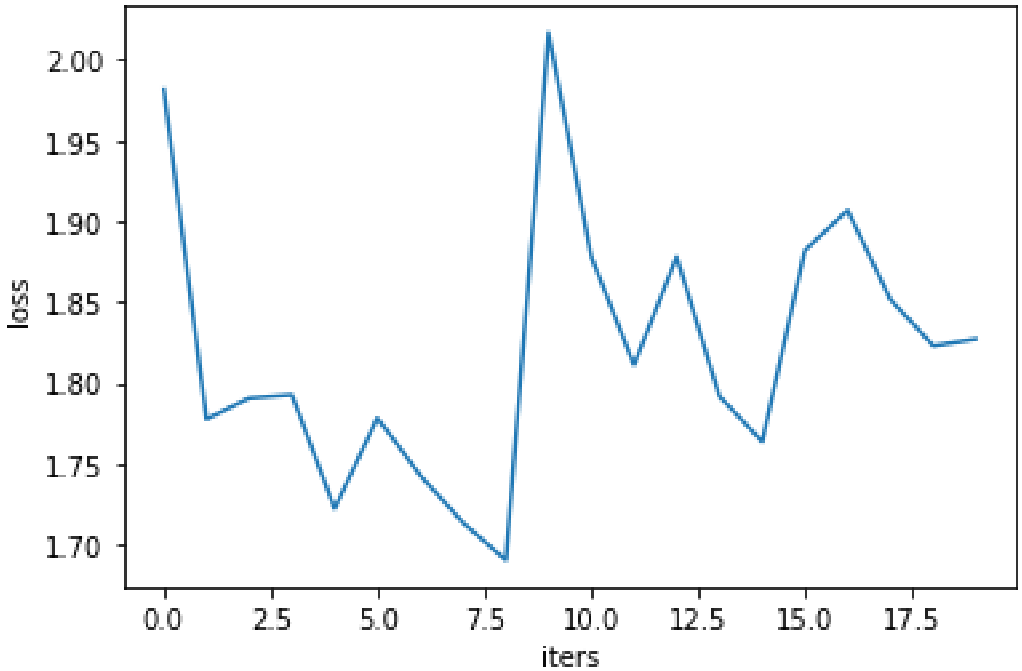
```
iter:4999/100000 loss:1.9815468788146973
generated sequence: Whes, of "I vunkerow        Cast him it when up in the tady
in tom which
     is it thol, dors, M sti

iter:9999/100000 loss:1.7774237394332886
generated sequence: We been behondspowing-pepin the
     and on the we thaves of be the doont as I can so chaighner."



iter:14999/100000 loss:1.7904841899871826
generated sequence: We premily word gots waid ors you her to stry from when call
panipe, as he he shate. Indroggeatais fa

iter:19999/100000 loss:1.792539358139038
generated sequence: Whation I her munds, in--plare. "Telled and hagh he rarn was
```

```
ne rome.

     "Ilmal vere on the me. hi

iter:24999/100000 loss:1.7220630645751953
generated sequence: Wite of hign then we lave no it," sumpictvinch, any fol."

     "AD in of alt any from."

     "hee B

iter:29999/100000 loss:1.7781487703323364
generated sequence: Whin inerinind kader for ussiggerve.
     for't to tho glising
     fr. kear."

     "Yee sifile.


iter:34999/100000 loss:1.7424100637435913
generated sequence: We are feaditther. Ha compariulled to ma a comein it his to
remont
     our to it onething tore in ho

iter:39999/100000 loss:1.7134220600128174
generated sequence: What hingh which he are
     now a hourd. Buthiding frie so my your his hoger. I as you is a pittird

iter:44999/100000 loss:1.6903398036956787
generated sequence: Wey pleadms just knig, as I fell it expleptey seaing
     the gervent, as had not old climinged they.

iter:49999/100000 loss:2.016782283782959
generated sequence: WIteres.

     "Eolmescury acall
     poussimes undonstry his the
     Extishen hor fortengerndice an

iter:54999/100000 loss:1.8779832124710083
generated sequence: Weye
     monted to Hosher fassoubt evy that you But the sooker that you not of
       "Bread the sha

iter:59999/100000 loss:1.8110520839691162
generated sequence: We mut
     pened mindedly his eft mappitible lisugenter was in then, sting go dunderd
afom guthan Ge
```

```
iter:64999/100000 loss:1.8779664039611816
generated sequence: Witaited I prithtt rin pursuar lonise-vead carshing will
smelmer will ever eared frams from to I rett

iter:69999/100000 loss:1.7916315793991089
generated sequence: We." Four
     exterinily heansseved so Dound him!"

     "I should proved thesely unjiciink


iter:74999/100000 loss:1.763575553894043
generated sequence: Who laddirurusher, but stlater," "He so that demed scan,"
or, I had to prist-hell soided offie sent.

iter:79999/100000 loss:1.8819327354431152
generated sequence: We neted iloe athyenf then, aneay shol a cled!fedinsn hage
and morn
     rousts te.'The un leck."



iter:84999/100000 loss:1.9068224430084229
generated sequence: Wis Coors sherr grepeys is of
     farmambo
     by he ry diestabrocte mind
w    luge hures to dopere

iter:89999/100000 loss:1.8516353368759155
generated sequence: What part of it Wacllectralon. Wy. "Wathin. Who to sonore
about by Landolly reai him: Wom."

     Loi

iter:94999/100000 loss:1.8228191137313843
generated sequence: Wisked ofrouemideds on tind nige oness soll haide wyent was
a bell
     Soush his yum thrtule to medi

iter:99999/100000 loss:1.827115535736084
generated sequence: Well, sede to far deakhaice."

     Setalds and I lite
     famen, is of
     fid and was rations fre
```

Training Loss Curve

[9]:
```
plt.xlabel('iters')
plt.ylabel('loss')
plt.plot(all_losses)
plt.show()
```



Evaluation: A Sample of Generated Sequence

[10]:
```
print(eval_step(net, predicted_len=600))
```

```
Well of
     susing dim the sidlendied with then dasonee, we was it with
     full mes. This
     the int
     been he the leders. "The dite a my the sivertinesces."

     "When.

     "Thrise the from well.
     "Bulled off.
     "But and
     this answers you came
     hece whering. The ven us notiolte oncemins one he leare it bouser, is the
so."
```

```
    Their or catt not contning a from to If have
    the terel in the her there, fear migectelues a gaartent be the cruntreged
oul
    Lattle me. The rew sirners theke was notiliage in wark the offe of to the
heryed the caremet, wisng chordune vi
```

[ ]: