final_project_catcher

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 = './catcher_in_the_rye.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: 380694
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', '{', '|', '}', '~', 'â', 'é']
Number of all possible characters: 102
```

[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(102, 100, bias=False)
  (linear): Linear(in_features=100, out_features=102, 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]:
```
# 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
```

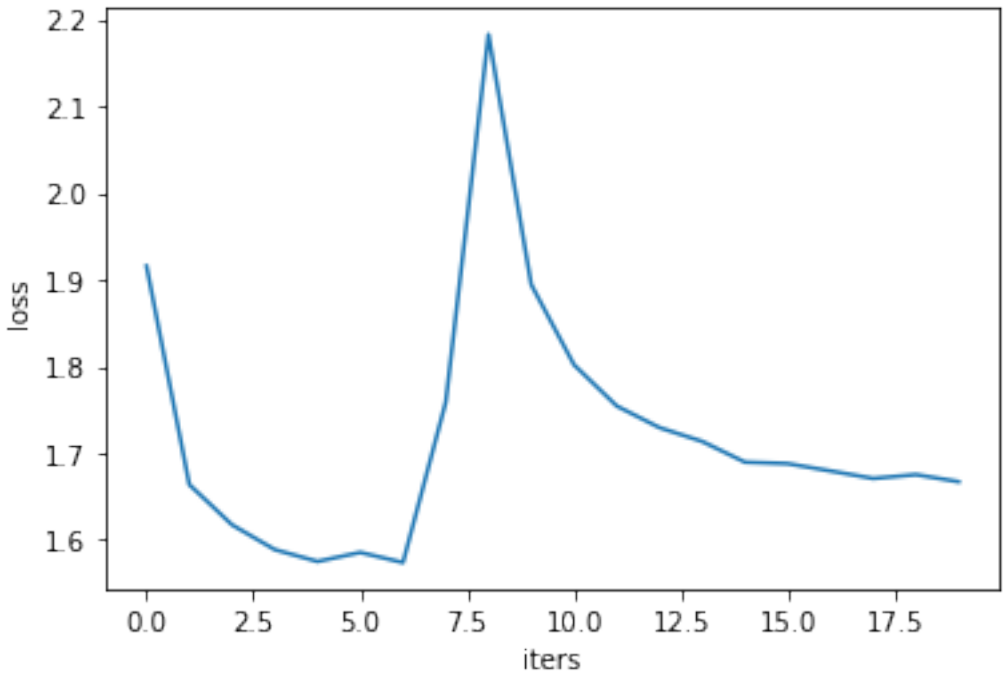
```
iter:4999/100000 loss:1.915701150894165
generated sequence: Why the sorderd I'd do you proot he was
at look and bit did. Straswine. No mart in a goddam
bres's
ge

iter:9999/100000 loss:1.663904070854187
generated sequence: Wht'vest. "It, so I meves he drot sude bebas starn
Rome about I'd buckne with hoor homman my cola onl

iter:14999/100000 loss:1.6174851655960083
generated sequence: Whooor on a mast, in her
a fingl in my pretty pucks. I dong it, even was arac old Lance it--"
"No."
"

iter:19999/100000 loss:1.5888748168945312
```

```
generated sequence: Who go buddann. It me. It drose, if Phat and hussank dadn
and even in these for you know. It cum out

iter:24999/100000 loss:1.5752642154693604
generated sequence: Well," al abe I derruder on a goddamns I at?"
"I dann thooch I and I wast near a could half my a give

iter:29999/100000 loss:1.5856388807296753
generated sequence: Whe otaing he atrines. I really all of the started out, "1
I couldn't wannlw ane," I got up you wante

iter:34999/100000 loss:1.5740139484405518
generated sequence: W'll she swinglen yo shed. Then
you oolr it tille then I was cart this fex pit of comcht clence we co

iter:39999/100000 loss:1.7588382959365845
generated sequence: Whicas little in the billed
herted's will she
wouldnering? I kidd I got
to come mweat do.
Thouched in

iter:44999/100000 loss:2.1826510429382324
generated sequence: We.--ers the tire her. All if or a got hute some, have he,
tampid, Lholk it thounty she dous ten.
He


iter:49999/100000 loss:1.8942697048187256
generated sequence: Whe the coums micasing at mid. I thought go. I'd there stere
nurou some ever fine that in. You or ove

iter:54999/100000 loss:1.8019583225250244
generated sequence: We of
a verentissed thessite clnpene, to mene fing hor you'rtissed?" I did iring liver
like the seret

iter:59999/100000 loss:1.7543872594833374
generated sequence: Whe Guse a mosking a Cabl. The wnorm, she, to shime all mant
a
theisendod of though."
"Mars somebody

iter:64999/100000 loss:1.7294727563858032
generated sequence: Whe sooovee
most the littialfissic sothing hack. Evech a me ciml a getter this crazy
probably down ho
```

```
iter:69999/100000 loss:1.713564157485962
generated sequence: We, and alle.
I sallost to go on my fers. We
and fatht be rapper on was aswee, book at Wendens and te

iter:74999/100000 loss:1.6896729469299316
generated sequence: Wh for dard'rld addn's facked the only dry was gon, I dly'd
with fict, but he's. Surk," she wher
lutt

iter:79999/100000 loss:1.6879678964614868
generated sequence: What's
of epaurry Pent
buclo a comica onay. I'm in the fabr didn't old Prica, ge funch get once the g

iter:84999/100000 loss:1.679552674293518
generated sequence: Wes so reall, but you was so old Tepleas Ic gore to go bece
Anly--C'lly deps way really so left trail

iter:89999/100000 loss:1.6709083318710327
generated sequence: What like and was over. I all little nood sell ficks oldl.
No in thoucide cigarad and I look Nothloth

iter:94999/100000 loss:1.675290584564209
generated sequence: Wes got obod in the was laor, nood old
"Goosarl to Gons. I'll she
goddam
so onhere's quef by I hame s

iter:99999/100000 loss:1.6671342849731445
generated sequence: Wes godd. So. I doull only it, but had a bigga in the?" I
swalling atarday,
bars tambel or lors when
```

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))
```

```
Whsy onas house meevid-dris,
and sent! You it, he saw it a glad whered anything do a gis nurd much and and
then the like feglle and littt or srandly seen at thisle," she out, when I hard
right thinking. She got I don't dod like with yaice, I dame did evet fuld in
feved wire," I so fin! And out he like thid at in the dough, the wasl "Whiding,
though, I drived in my picks now, lonell a blarn, ehind of cald all,"
"DoI'd, only didn't and little with me. Age did."
Fine sed in whing, She hot to grousr."
"Wusing thoubly goied like pascaw only. She fell she what was lifed, the should
ford, I was sce go
```

[ ]: