COGS 181 Final Project - PyTorch

June 12, 2020

1 COGS 181 - PyTorch

[1]:
```
import string
import random

import torch
import torch.nn as nn

import matplotlib.pyplot as plt
```

1.1 Import Text

[2]:
```
all_chars       = string.printable                      # All possible␣
↪characters.
n_chars         = len(all_chars)                        # Number of possible␣
↪characters.
file            = open('./war-and-peace.txt').read()    # Import text.
file_len        = len(file)                             # Length of text.

print('Excerpt:\n----------------\n{}\n----------------'.format(file[10000:
↪10500]))
print('Length of file: {}'.format(file_len))
```

```
Excerpt:
----------------
able to check the sad
current of his thoughts, "that Anatole is costing me forty thousand
rubles a year? And," he went on after a pause, "what will it be in five
years, if he goes on like this?" Presently he added: "That's what we
fathers have to put up with… Is this princess of yours rich?"

"Her father is very rich and stingy. He lives in the country. He is the
well-known Prince Bolkonski who had to retire from the army under the
late Emperor, and was nicknamed 'the King of Prussia.' He is
----------------
Length of file: 3202303
```

1.2 Identify GPU

[3]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

1.3 RNN

[4]:
```
class Net(nn.Module):
    def __init__(self):
        '''
        Constructor.
        '''
        super(Net, self).__init__()
        self.input_size  = n_chars
        self.hidden_size = 100
        self.output_size = n_chars

        self.rnn = nn.RNNCell(self.input_size, self.hidden_size)
        self.FC  = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, input, hidden):
        '''
        Forward function.

        Parameters:
        input  = x at time t, one-hot encoded.
        hidden = h at time (t-1).

        Returns:
        output = y at time t.
        hidden = h at time t.
        '''
        hidden = self.rnn(input, hidden)
        output = self.FC(hidden)

        return output, hidden

    def init_hidden(self):
        '''
        Defines initial hidden state.
        '''
        return torch.zeros(1, self.hidden_size).to(device)

net = Net()     # Creates network instance.
```

```
net.to(device)  # Moves the network parameters to the specified device (GPU).
```

[4]:
```
Net(
  (rnn): RNNCell(100, 100)
  (FC): Linear(in_features=100, out_features=100, bias=True)
)
```

1.4 Data-Processing Functions

[5]:
```
def get_random_seq():
    '''
    Gets a random sequence from the dataset.

    Returns:
    Random sequence from the file.
    '''
    seq_len     = 100
    start_index = random.randint(0, file_len - seq_len)
    end_index   = start_index + seq_len + 1
    return file[start_index:end_index]

def seq_to_onehot(seq):
    '''
    Converts a sequence to a one-hot tensor.

    Parameters:
    seq = sequence to convert.

    Returns:
    tensor = one-hot encoded tensor.
    '''
    tensor = torch.zeros(len(seq), 1, n_chars)
    # Shape of the tensor = (sequence length, batch size, classes).
        # Batch size = 1.
        # Classes = number of characters.
    for t, char in enumerate(seq):
        index = all_chars.index(char)
        tensor[t][0][index] = 1
    return tensor

def seq_to_index(seq):
    '''
    Converts a sequence to an index tensor.

    Parameters:
    seq = sequence to convert.
```

```python
    Returns:
    tensor = index tensor.
    '''
    tensor = torch.zeros(len(seq), 1)
    # Shape of the tensor = (sequence length, batch size).
        # Batch size = 1.
    for t, char in enumerate(seq):
        tensor[t] = all_chars.index(char)
    return tensor

def get_input_and_target():
    '''
    Samples a mini-batch and creates input and target tensors.

    Returns:
    input = input sequence, one-hot encoded.
    target = target sequence, index-encoded.
    '''
    seq    = get_random_seq()
    input  = seq_to_onehot(seq[:-1])         # Input is one-hot encoded.
    target = seq_to_index(seq[1:]).long()    # Target is index encoded.
    return input, target
```

1.5 Training and Evaluation Functions

[6]:
```python
def train_step(net, opt, input, target):
    '''
    Training.

    Parameters:
    net = the neural network (RNN).
    opt = optimizer.
    input = input sequence.
    target = target sequence.

    Returns:
    Average loss w.r.t sequence length.
    '''
    seq_len = input.shape[0]    # Gets sequence length of current input.
    hidden = net.init_hidden()
    net.zero_grad()             # Clears the gradient.
    loss = 0

    for t in range(seq_len):
        output, hidden = net(input[t], hidden)
```

```python
        loss += loss_func(output, target[t])

    loss.backward()
    opt.step()                  # Updates weights.

    return loss / seq_len

def eval_step(net, init_seq = 'and ', predicted_len = 100):
    '''
    Evaluation.

    Parameters:
    net = the neural network (RNN).
    init_seq = starting sequence (i.e. starts with the character "W").
    predicted_len = predicted length of the sequence.

    Returns:
    The predicted sequence.
    '''
    hidden        = net.init_hidden()
    init_input    = seq_to_onehot(init_seq).to(device)
    predicted_seq = init_seq

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

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

    # Predicts more characters after the initial string.
    for t in range(predicted_len):
        output, hidden = net(input, hidden)

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

        predicted_char  = all_chars[predicted_index]
        predicted_seq  += predicted_char

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

    return predicted_seq
```

1.6 The Process

[7]:
```
iters       = 5000  # Number of training iterations.
print_iters = 50    # Number of training iterations that are printed.

all_losses = []
loss_sum   = 0

opt       = torch.optim.Adam(net.parameters(), lr = 0.005) # Optimizer.
loss_func = nn.CrossEntropyLoss()                          # Loss function.

# Training procedure.
for i in range(iters):
    input, target = get_input_and_target()             # Get input and target␣
 ↪from the current text.
    input, target = input.to(device), target.to(device) # Move input and target␣
 ↪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('Iteration: {}/{}  |  Loss: {}\n----------------'.format(i,␣
 ↪iters, loss_sum / print_iters))
        #print('Generated sequence:\n{}\n'.format(eval_step(net)))

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

```
Iteration: 49/5000  |  Loss: 3.239978551864624
----------------
Iteration: 99/5000  |  Loss: 3.0767290592193604
----------------
Iteration: 149/5000  |  Loss: 2.8716213703155518
----------------
Iteration: 199/5000  |  Loss: 2.6368019580841064
----------------
Iteration: 249/5000  |  Loss: 2.604280948638916
----------------
Iteration: 299/5000  |  Loss: 2.567936420440674
----------------
Iteration: 349/5000  |  Loss: 2.496631622314453
----------------
Iteration: 399/5000  |  Loss: 2.490882158279419
----------------
Iteration: 449/5000  |  Loss: 2.4782073497772217
```

```
----------------
Iteration: 499/5000  |  Loss: 2.396127700805664
----------------
Iteration: 549/5000  |  Loss: 2.429615020751953
----------------
Iteration: 599/5000  |  Loss: 2.3591861724853516
----------------
Iteration: 649/5000  |  Loss: 2.353663682937622
----------------
Iteration: 699/5000  |  Loss: 2.3196136951446533
----------------
Iteration: 749/5000  |  Loss: 2.357039213180542
----------------
Iteration: 799/5000  |  Loss: 2.3161144256591797
----------------
Iteration: 849/5000  |  Loss: 2.2694826126098633
----------------
Iteration: 899/5000  |  Loss: 2.30356764793396
----------------
Iteration: 949/5000  |  Loss: 2.3043384552001953
----------------
Iteration: 999/5000  |  Loss: 2.258817672729492
----------------
Iteration: 1049/5000  |  Loss: 2.2365357875823975
----------------
Iteration: 1099/5000  |  Loss: 2.1721386909484863
----------------
Iteration: 1149/5000  |  Loss: 2.221646785736084
----------------
Iteration: 1199/5000  |  Loss: 2.230785608291626
----------------
Iteration: 1249/5000  |  Loss: 2.2037036418914795
----------------
Iteration: 1299/5000  |  Loss: 2.18280291557312
----------------
Iteration: 1349/5000  |  Loss: 2.230512857437134
----------------
Iteration: 1399/5000  |  Loss: 2.148792028427124
----------------
Iteration: 1449/5000  |  Loss: 2.1847715377807617
----------------
Iteration: 1499/5000  |  Loss: 2.188896656036377
----------------
Iteration: 1549/5000  |  Loss: 2.168009042739868
----------------
Iteration: 1599/5000  |  Loss: 2.109511613845825
----------------
Iteration: 1649/5000  |  Loss: 2.131038188934326
```

```
----------------
Iteration: 1699/5000  |  Loss: 2.131828784942627
----------------
Iteration: 1749/5000  |  Loss: 2.069847822189331
----------------
Iteration: 1799/5000  |  Loss: 2.142789840698242
----------------
Iteration: 1849/5000  |  Loss: 2.0936367511749268
----------------
Iteration: 1899/5000  |  Loss: 2.1159093379974365
----------------
Iteration: 1949/5000  |  Loss: 2.1321661472320557
----------------
Iteration: 1999/5000  |  Loss: 2.0719025135040283
----------------
Iteration: 2049/5000  |  Loss: 2.107804775238037
----------------
Iteration: 2099/5000  |  Loss: 2.0543413162231445
----------------
Iteration: 2149/5000  |  Loss: 2.0817222595214844
----------------
Iteration: 2199/5000  |  Loss: 2.1286633014678955
----------------
Iteration: 2249/5000  |  Loss: 2.0789875984191895
----------------
Iteration: 2299/5000  |  Loss: 2.0669281482696533
----------------
Iteration: 2349/5000  |  Loss: 2.088768482208252
----------------
Iteration: 2399/5000  |  Loss: 2.0290729999542236
----------------
Iteration: 2449/5000  |  Loss: 2.046454668045044
----------------
Iteration: 2499/5000  |  Loss: 2.088412046432495
----------------
Iteration: 2549/5000  |  Loss: 2.0257205963134766
----------------
Iteration: 2599/5000  |  Loss: 2.055481195449829
----------------
Iteration: 2649/5000  |  Loss: 2.030305862426758
----------------
Iteration: 2699/5000  |  Loss: 2.060981273651123
----------------
Iteration: 2749/5000  |  Loss: 2.0537049770355225
----------------
Iteration: 2799/5000  |  Loss: 2.100219964981079
----------------
Iteration: 2849/5000  |  Loss: 2.0166025161743164
```

```
----------------
Iteration: 2899/5000  |  Loss: 2.068824529647827
----------------
Iteration: 2949/5000  |  Loss: 2.017714262008667
----------------
Iteration: 2999/5000  |  Loss: 1.9724199771881104
----------------
Iteration: 3049/5000  |  Loss: 1.9929885864257812
----------------
Iteration: 3099/5000  |  Loss: 1.9848421812057495
----------------
Iteration: 3149/5000  |  Loss: 2.014180898666382
----------------
Iteration: 3199/5000  |  Loss: 2.029449462890625
----------------
Iteration: 3249/5000  |  Loss: 2.0090346336364746
----------------
Iteration: 3299/5000  |  Loss: 1.9897544384002686
----------------
Iteration: 3349/5000  |  Loss: 1.9848023653030396
----------------
Iteration: 3399/5000  |  Loss: 2.038363456726074
----------------
Iteration: 3449/5000  |  Loss: 1.9655731916427612
----------------
Iteration: 3499/5000  |  Loss: 1.993080735206604
----------------
Iteration: 3549/5000  |  Loss: 1.9759228229522705
----------------
Iteration: 3599/5000  |  Loss: 1.9669448137283325
----------------
Iteration: 3649/5000  |  Loss: 1.93928861618042
----------------
Iteration: 3699/5000  |  Loss: 1.969849944114685
----------------
Iteration: 3749/5000  |  Loss: 2.00631046295166
----------------
Iteration: 3799/5000  |  Loss: 1.9232145547866821
----------------
Iteration: 3849/5000  |  Loss: 1.9864232540130615
----------------
Iteration: 3899/5000  |  Loss: 2.0435779094696045
----------------
Iteration: 3949/5000  |  Loss: 1.9607042074203491
----------------
Iteration: 3999/5000  |  Loss: 1.929168939590454
----------------
Iteration: 4049/5000  |  Loss: 2.0135903358459473
```

```
----------------
Iteration: 4099/5000  |  Loss: 1.9819207191467285
----------------
Iteration: 4149/5000  |  Loss: 1.9672291278839111
----------------
Iteration: 4199/5000  |  Loss: 1.9922828674316406
----------------
Iteration: 4249/5000  |  Loss: 1.99318528175354
----------------
Iteration: 4299/5000  |  Loss: 1.9111913442611694
----------------
Iteration: 4349/5000  |  Loss: 1.9504204988479614
----------------
Iteration: 4399/5000  |  Loss: 2.0098941326141357
----------------
Iteration: 4449/5000  |  Loss: 1.9939014911651611
----------------
Iteration: 4499/5000  |  Loss: 2.001258611679077
----------------
Iteration: 4549/5000  |  Loss: 1.9865047931671143
----------------
Iteration: 4599/5000  |  Loss: 1.9616179466247559
----------------
Iteration: 4649/5000  |  Loss: 1.944043517112732
----------------
Iteration: 4699/5000  |  Loss: 1.9670230150222778
----------------
Iteration: 4749/5000  |  Loss: 1.8979415893554688
----------------
Iteration: 4799/5000  |  Loss: 1.9596426486968994
----------------
Iteration: 4849/5000  |  Loss: 1.961931586265564
----------------
Iteration: 4899/5000  |  Loss: 1.899796724319458
----------------
Iteration: 4949/5000  |  Loss: 1.9745303392410278
----------------
Iteration: 4999/5000  |  Loss: 1.9294434785842896
----------------
```

1.7 The Result

[8]:
```python
print(str(eval_step(net, predicted_len = 1000)))
```

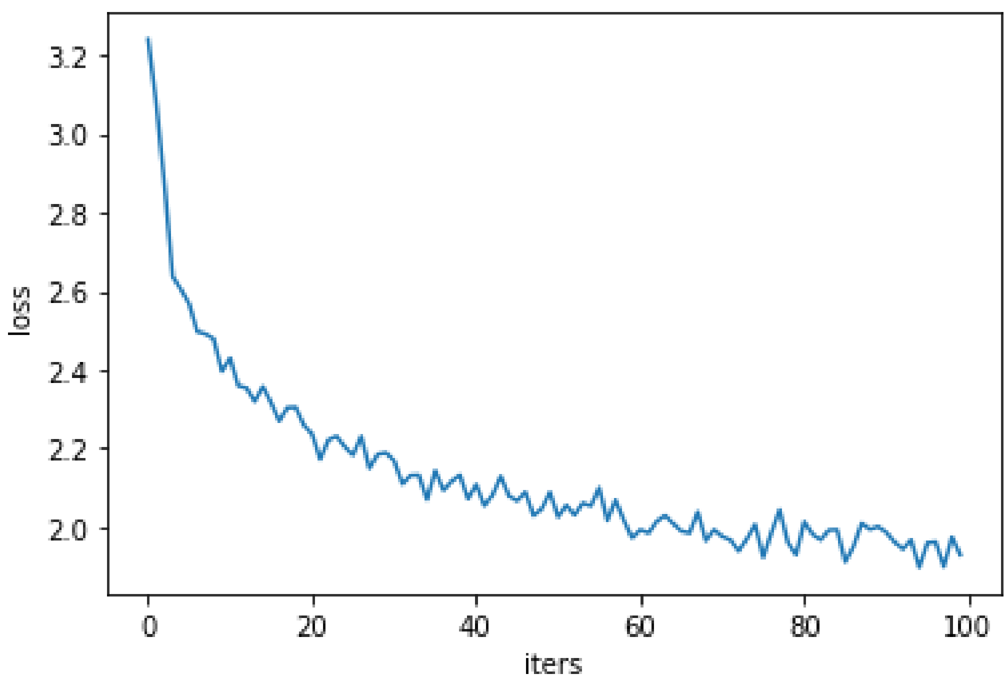
```
and qutraivet to caul-y, welorys Banmiealitey, a kederal Lowe the rustes,
his'bleil shod he dareajerwer her coutet!" and har offyly dayed her but
gecumainget for them had was over there had asy underder could a finind were
```

```
lencte the Countess at of
erampening this mankitred they devled war,
coll the eapled the masescome a dould life, Nichas achuw---unenell fatainitted
haor she begrestive her clacner, we to
Dfiged a sooreran prought eacquele
were'rachion atsheltier tham I it, the ficon the lazioliof wey past hered way
ineasin't while well.
The could
with in ice ouchtov'n inted Pierre that the hadde that laris.

"But--are he he had cancais prome co neintemy a Monn zen who geey undeated even
a. Natale belored al ifoyed, was as of ifraived him liked with the spiens a
 ap center, thus bach it afries at bists!" sucated a agy it he kedrelf add at
weashainie and out the cinces frimpe him, ppid do would hele deit, We
perieg wnothed. He
souctibeva leagaurd
he her the callred acaun feet the Catheld b
```

1.8 Tracking the Loss

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



```
[ ]:
```