Problem_2

April 15, 2024

[ ]:
```
'''
Patrick Ballou
ID: 801130521
ECGR 4106
Homework 4
Problem 2
'''
```

[ ]:
```
'\nPatrick Ballou\nID: 801130521\nECGR 4106\nHomework 4\nProblem 2\n'
```

[ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch import cuda
from torch.utils.data import DataLoader, Dataset
```

[ ]:
```
#check if GPU is available and set the device accordingly
#device = 'torch.device("cuda:0" if torch.cuda.is_available() else "cpu")'
device = 'cuda'
print("Using GPU: ", cuda.get_device_name())

gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Using GPU:  Quadro T2000
Mon Apr 15 13:50:12 2024
+-------------------------------------------------------------------------------
----------+
| NVIDIA-SMI 551.86                 Driver Version: 551.86         CUDA Version:
12.4     |
|-----------------------------------------+------------------------+------------
----------+
| GPU  Name                     TCC/WDDM  | Bus-Id          Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util
Compute M. |
|                                         |                        |
MIG M. |
|=========================================+========================+============
==========|
|   0  Quadro T2000                 WDDM  |   00000000:01:00.0  On |
N/A |
| N/A   53C    P0              13W /   60W |    1759MiB /   4096MiB |      0%
Default |
|                                         |                        |
N/A |
+-----------------------------------------+------------------------+------------
----------+

+-------------------------------------------------------------------------------
----------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|===============================================================================
==========|
|    0   N/A  N/A      2084    C+G   …t.LockApp_cw5n1h2txyewy\LockApp.exe
N/A      |
|    0   N/A  N/A      3148    C+G   …e Stream\89.0.2.0\GoogleDriveFS.exe
N/A      |
|    0   N/A  N/A      6212    C+G   …Programs\Microsoft VS Code\Code.exe
N/A      |
|    0   N/A  N/A     10052    C+G   …b3d8bbwe\Microsoft.Media.Player.exe
N/A      |
|    0   N/A  N/A     10604    C+G   …1300.0_x64__8j3eq9eme6ctt\IGCC.exe
N/A      |
|    0   N/A  N/A     12068    C+G   C:\Windows\explorer.exe
N/A      |
|    0   N/A  N/A     13612    C+G   …siveControlPanel\SystemSettings.exe
N/A      |
|    0   N/A  N/A     14436    C+G   …CBS_cw5n1h2txyewy\TextInputHost.exe
N/A      |
|    0   N/A  N/A     15644    C+G   …Search_cw5n1h2txyewy\SearchApp.exe
N/A      |
|    0   N/A  N/A     16524    C+G   …Brave-Browser\Application\brave.exe
N/A      |
|    0   N/A  N/A     17168    C+G   …aam7r\AcrobatNotificationClient.exe
N/A      |
|    0   N/A  N/A     18440    C+G   …ekyb3d8bbwe\PhoneExperienceHost.exe
```

```
N/A      |
|    0   N/A  N/A     19164    C+G   …5n1h2txyewy\ShellExperienceHost.exe
N/A      |
|    0   N/A  N/A     19280    C+G   …ta\Local\Programs\Notion\Notion.exe
N/A      |
|    0   N/A  N/A     20356    C+G   …91.0_x64__8wekyb3d8bbwe\GameBar.exe
N/A      |
|    0   N/A  N/A     21180    C+G   …AppData\Roaming\Spotify\Spotify.exe
N/A      |
|    0   N/A  N/A     21636    C+G   …les\Microsoft OneDrive\OneDrive.exe
N/A      |
|    0   N/A  N/A     24908      C   …ri\anaconda3\envs\dl_env\python.exe
N/A      |
+-------------------------------------------------------------------------------
----------+
```

[ ]:

```
text = [
    ("I am cold", "J'ai froid"),
    ("You are tired", "Tu es fatigué"),
    ("He is hungry", "Il a faim"),
    ("She is happy", "Elle est heureuse"),
    ("We are friends", "Nous sommes amis"),
    ("They are students", "Ils sont étudiants"),
    ("The cat is sleeping", "Le chat dort"),
    ("The sun is shining", "Le soleil brille"),
    ("We love music", "Nous aimons la musique"),
    ("She speaks French fluently", "Elle parle français couramment"),
    ("He enjoys reading books", "Il aime lire des livres"),
    ("They play soccer every weekend", "Ils jouent au football chaque␣
  ↪week-end"),
    ("The movie starts at 7 PM", "Le film commence à 19 heures"),
    ("She wears a red dress", "Elle porte une robe rouge"),
    ("We cook dinner together", "Nous cuisinons le dîner ensemble"),
    ("He drives a blue car", "Il conduit une voiture bleue"),
    ("They visit museums often", "Ils visitent souvent des musées"),
    ("The restaurant serves delicious food", "Le restaurant sert une délicieuse␣
  ↪cuisine"),
    ("She studies mathematics at university", "Elle étudie les mathématiques à␣
  ↪l'université"),
    ("We watch movies on Fridays", "Nous regardons des films le vendredi"),
    ("He listens to music while jogging", "Il écoute de la musique en faisant␣
  ↪du jogging"),
    ("They travel around the world", "Ils voyagent autour du monde"),
    ("The book is on the table", "Le livre est sur la table"),
    ("She dances gracefully", "Elle danse avec grâce"),
    ("We celebrate birthdays with cake", "Nous célébrons les anniversaires avec␣
  ↪un gâteau"),
    ("He works hard every day", "Il travaille dur tous les jours"),
    ("They speak different languages", "Ils parlent différentes langues"),
    ("The flowers bloom in spring", "Les fleurs fleurissent au printemps"),
    ("She writes poetry in her free time", "Elle écrit de la poésie pendant son␣
↪temps libre"),
    ("We learn something new every day", "Nous apprenons quelque chose de␣
↪nouveau chaque jour"),
    ("The dog barks loudly", "Le chien aboie bruyamment"),
    ("He sings beautifully", "Il chante magnifiquement"),
    ("They swim in the pool", "Ils nagent dans la piscine"),
    ("The birds chirp in the morning", "Les oiseaux gazouillent le matin"),
    ("She teaches English at school", "Elle enseigne l'anglais à l'école"),
    ("We eat breakfast together", "Nous prenons le petit déjeuner ensemble"),
    ("He paints landscapes", "Il peint des paysages"),
    ("They laugh at the joke", "Ils rient de la blague"),
    ("The clock ticks loudly", "L'horloge tic-tac bruyamment"),
    ("She runs in the park", "Elle court dans le parc"),
    ("We travel by train", "Nous voyageons en train"),
    ("He writes a letter", "Il écrit une lettre"),
    ("They read books at the library", "Ils lisent des livres à la␣
↪bibliothèque"),
    ("The baby cries", "Le bébé pleure"),
    ("She studies hard for exams", "Elle étudie dur pour les examens"),
    ("We plant flowers in the garden", "Nous plantons des fleurs dans le␣
↪jardin"),
    ("He fixes the car", "Il répare la voiture"),
    ("They drink coffee in the morning", "Ils boivent du café le matin"),
    ("The sun sets in the evening", "Le soleil se couche le soir"),
    ("She dances at the party", "Elle danse à la fête"),
    ("We play music at the concert", "Nous jouons de la musique au concert"),
    ("He cooks dinner for his family", "Il cuisine le dîner pour sa famille"),
    ("They study French grammar", "Ils étudient la grammaire française"),
    ("The rain falls gently", "La pluie tombe doucement"),
    ("She sings a song", "Elle chante une chanson"),
    ("We watch a movie together", "Nous regardons un film ensemble"),
    ("He sleeps deeply", "Il dort profondément"),
    ("They travel to Paris", "Ils voyagent à Paris"),
    ("The children play in the park", "Les enfants jouent dans le parc"),
    ("She walks along the beach", "Elle se promène le long de la plage"),
    ("We talk on the phone", "Nous parlons au téléphone"),
    ("He waits for the bus", "Il attend le bus"),
    ("They visit the Eiffel Tower", "Ils visitent la tour Eiffel"),
    ("The stars twinkle at night", "Les étoiles scintillent la nuit"),
    ("She dreams of flying", "Elle rêve de voler"),
    ("We work in the office", "Nous travaillons au bureau"),
    ("He studies history", "Il étudie l'histoire"),
    ("They listen to the radio", "Ils écoutent la radio"),
    ("The wind blows gently", "Le vent souffle doucement"),
    ("She swims in the ocean", "Elle nage dans l'océan"),
    ("We dance at the wedding", "Nous dansons au mariage"),
    ("He climbs the mountain", "Il gravit la montagne"),
    ("They hike in the forest", "Ils font de la randonnée dans la forêt"),
    ("The cat meows loudly", "Le chat miaule bruyamment"),
    ("She paints a picture", "Elle peint un tableau"),
    ("We build a sandcastle", "Nous construisons un château de sable"),
    ("He sings in the choir", "Il chante dans le chœur")
]

SOS_token = 0
EOS_token = 1
```

[ ]:

```
# Vocabulary class to handle mapping between words and numerical indices
class Vocabulary:
    def __init__(self):
        # Initialize dictionaries for word to index and index to word mappings
        self.word2index = {"<SOS>": SOS_token, "<EOS>": EOS_token}
        self.index2word = {SOS_token: "<SOS>", EOS_token: "<EOS>"}
        self.word_count = {}  # Keep track of word frequencies
        self.n_words = 2  # Start counting from 2 to account for special tokens

    def add_sentence(self, sentence):
        # Add all words in a sentence to the vocabulary
        for word in sentence.split(' '):
            self.add_word(word)

    def add_word(self, word):
        # Add a word to the vocabulary
        if word not in self.word2index:
            # Assign a new index to the word and update mappings
            self.word2index[word] = self.n_words
            self.index2word[self.n_words] = word
            self.word_count[word] = 1
            self.n_words += 1
        else:
            # Increment word count if the word already exists in the vocabulary
            self.word_count[word] += 1

# Custom Dataset class for English to French sentences
class EngFrDataset(Dataset):
    def __init__(self, pairs):
        self.eng_vocab = Vocabulary()
        self.fr_vocab = Vocabulary()
        self.pairs = []
        # Process each English-French pair
        for eng, fr in pairs:
            self.eng_vocab.add_sentence(eng)
            self.fr_vocab.add_sentence(fr)
            self.pairs.append((eng, fr))

        # Separate English and French sentences
        self.eng_sentences = [pair[0] for pair in self.pairs]
        self.fr_sentences = [pair[1] for pair in self.pairs]

    # Returns the number of pairs
    def __len__(self):
        return len(self.pairs)

    # Get the sentences by index
    def __getitem__(self, idx):
        input_sentence = self.eng_sentences[idx]
        target_sentence = self.fr_sentences[idx]
        input_indices = [self.eng_vocab.word2index[word] for word in␣
↪input_sentence.split()] + [EOS_token]
        target_indices = [self.fr_vocab.word2index[word] for word in␣
↪target_sentence.split()] + [EOS_token]

        return torch.tensor(input_indices, dtype=torch.long), torch.
↪tensor(target_indices, dtype=torch.long)

# Initialize the dataset and DataLoader
e2f_dataset = EngFrDataset(text)
dataloader = DataLoader(e2f_dataset, batch_size=1, shuffle=True)
```

[ ]:

```
class Encoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Encoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size) # Embedding layer
        self.gru = nn.GRU(hidden_size, hidden_size) # GRU layer

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output, hidden = self.gru(embedded, hidden)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)

# Decoder with attention
class AttentionDecoder(nn.Module):
    def __init__(self, hidden_size, output_size, max_length=16, dropout_p=0.1):
        super(AttentionDecoder, self).__init__()
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.dropout_p = dropout_p
        self.max_length = max_length

        self.embedding = nn.Embedding(self.output_size, self.hidden_size) #␣
↪Embedding layer
        self.attn = nn.Linear(self.hidden_size * 2, self.max_length) #␣
↪Attention layer
        self.attn_combine = nn.Linear(self.hidden_size * 2, self.hidden_size) #␣
↪Combining layer
        self.dropout = nn.Dropout(self.dropout_p)
        self.gru = nn.GRU(self.hidden_size, self.hidden_size)
        self.out = nn.Linear(self.hidden_size, output_size)

    def forward(self, input, hidden, encoder_outputs):
        embedded = self.embedding(input).view(1, 1, -1)
        embedded = self.dropout(embedded)

        # Calculating attention weights
        attn_weights = torch.softmax(self.attn(torch.cat((embedded[0],␣
↪hidden[0]), 1)), dim=1)
        attn_applied = torch.bmm(attn_weights.unsqueeze(0), encoder_outputs.
↪unsqueeze(0))

        # Combining embedded input with attention output
        output = torch.cat((embedded[0], attn_applied[0]), 1)
        output = self.attn_combine(output).unsqueeze(0)

        output = torch.relu(output)
        output, hidden = self.gru(output, hidden)

        output = torch.log_softmax(self.out(output[0]), dim=1)
        return output, hidden, attn_weights

    def initHidden(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

[ ]:

```
def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer,␣
↪decoder_optimizer, criterion, max_length=16):
    # Initialize encoder hidden state
    encoder_hidden = encoder.initHidden()

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()
    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    loss = 0

    # Encoding each character in the input
    encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)
    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(input_tensor[ei].unsqueeze(0), encoder_hidden)
        encoder_outputs[ei] = encoder_output[0, 0]

    # Decoder's first input is the SOS token
    decoder_input = torch.tensor([[SOS_token]], device=device)

    # Decoder starts with encoder's last hidden state
    decoder_hidden = encoder_hidden

   # Decoding loop with attention
    for di in range(target_length):
        decoder_output, decoder_hidden, decoder_attention = decoder(
            decoder_input, decoder_hidden, encoder_outputs)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # Detach from history as input

        loss += criterion(decoder_output, target_tensor[di].unsqueeze(0))
        if decoder_input.item() == EOS_token:
            break

    # Backpropagation
    loss.backward()

    # Update encoder and decoder
    encoder_optimizer.step()
    decoder_optimizer.step()

    # Return average loss
    return loss.item() / target_length

def train__loop(encoder, decoder, encoder_optimizer, decoder_optimizer, criterion, dataloader, epochs):
    for epoch in range(epochs):
        total_loss = 0
        for input_tensor, target_tensor in dataloader:
            input_tensor = input_tensor[0].to(device)
            target_tensor = target_tensor[0].to(device)

            # Perform a single training step and update loss
            loss = train(input_tensor, target_tensor, encoder, decoder,␣
↪encoder_optimizer, decoder_optimizer, criterion)
            total_loss += loss

        # Print loss every 5 epochs
        if epoch % 5 == 0:
            print(f'Epoch {epoch}, Loss: {total_loss / len(dataloader)}')
```

[ ]:

```
def evaluate_and_show_examples(encoder, decoder, dataloader, criterion,␣
↪n_examples=10, max_length=16):
    # Switch model to evaluation mode
    encoder.eval()
    decoder.eval()

    total_loss = 0
    correct_predictions = 0

    # No gradient calculation
    with torch.no_grad():
        for i, (input_tensor, target_tensor) in enumerate(dataloader):
            # Move tensors to the correct device
            input_tensor = input_tensor[0].to(device)
            target_tensor = target_tensor[0].to(device)

            encoder_hidden = encoder.initHidden()

            input_length = input_tensor.size(0)
            target_length = target_tensor.size(0)

            loss = 0

            # Encoding each character in the input
            encoder_outputs = torch.zeros(max_length, encoder.hidden_size,␣
↪device=device)
            for ei in range(input_length):
                encoder_output, encoder_hidden = encoder(input_tensor[ei].
↪unsqueeze(0), encoder_hidden)
                encoder_outputs[ei] = encoder_output[0, 0]

            # Decoding step
            decoder_input = torch.tensor([[SOS_token]], device=device)
            decoder_hidden = encoder_hidden

            predicted_indices = []
            for di in range(target_length):
                decoder_output, decoder_hidden, decoder_attention =␣
↪decoder(decoder_input, decoder_hidden, encoder_outputs)
                topv, topi = decoder_output.topk(1)
                predicted_indices.append(topi.item())
                decoder_input = topi.squeeze().detach()

                loss += criterion(decoder_output, target_tensor[di].
↪unsqueeze(0))
                if decoder_input.item() == EOS_token:
                    break

            # Calculate and print loss and accuracy for the evaluation
            total_loss += loss.item() / target_length
            if predicted_indices == target_tensor.tolist():
                correct_predictions += 1

            # Print some examples
            if i < n_examples:
                predicted_sentence = ' '.join([dataloader.dataset.fr_vocab.
↪index2word[index] for index in predicted_indices if index not in (SOS_token,␣
↪EOS_token)])
                target_sentence = ' '.join([dataloader.dataset.fr_vocab.
↪index2word[index.item()] for index in target_tensor if index.item() not in␣
↪(SOS_token, EOS_token)])
                input_sentence = ' '.join([dataloader.dataset.eng_vocab.
↪index2word[index.item()] for index in input_tensor if index.item() not in␣
↪(SOS_token, EOS_token)])

                print(f'Input: {input_sentence}, Target: {target_sentence},␣
↪Predicted: {predicted_sentence}')

        # Print overall evaluation results
        average_loss = total_loss / len(dataloader)
        accuracy = correct_predictions / len(dataloader)
        print(f'Evaluation Loss: {average_loss:.4f}, Accuracy: {100*accuracy:.
↪2f}%')
```

[ ]:

```
# Params
epochs = 51
learning_rate = 0.01
hidden_size = 1028
```

[ ]:

```
input_size = len(e2f_dataset.eng_vocab.word2index)
output_size = len(e2f_dataset.fr_vocab.word2index)
encoder = Encoder(input_size=input_size, hidden_size=hidden_size).to(device)
```

```python
decoder = AttentionDecoder(hidden_size=hidden_size, output_size=output_size).to(device)

# Optimizers
encoder_optimizer = optim.SGD(encoder.parameters(), lr=learning_rate)
decoder_optimizer = optim.SGD(decoder.parameters(), lr=learning_rate)

criterion = nn.NLLLoss()

train__loop(encoder, decoder, encoder_optimizer, decoder_optimizer, criterion, dataloader, epochs)
```

```
Epoch 0, Loss: 3.0707377460513547
Epoch 5, Loss: 2.4297287939455416
Epoch 10, Loss: 1.288507148475407
Epoch 15, Loss: 0.4999194042888087
Epoch 20, Loss: 0.1366252877687491
Epoch 25, Loss: 0.05094209939473329
Epoch 30, Loss: 0.02466687192989486
Epoch 35, Loss: 0.016949383764453062
Epoch 40, Loss: 0.013042975576501172
Epoch 45, Loss: 0.010492681922155682
Epoch 50, Loss: 0.008808551625705915
```

[ ]:

```python
evaluate_and_show_examples(encoder, decoder, dataloader, criterion)
```

```
Input: They speak different languages, Target: Ils parlent différentes langues,
Predicted: Ils parlent différentes langues
Input: He sleeps deeply, Target: Il dort profondément, Predicted: Il dort
profondément
Input: We build a sandcastle, Target: Nous construisons un château de sable,
Predicted: Nous construisons un château de sable
Input: The sun sets in the evening, Target: Le soleil se couche le soir,
Predicted: Le soleil se couche le soir
Input: The movie starts at 7 PM, Target: Le film commence à 19 heures,
Predicted: Le film commence à 19 heures
Input: They are students, Target: Ils sont étudiants, Predicted: Ils sont
étudiants
Input: The children play in the park, Target: Les enfants jouent dans le parc,
Predicted: Les enfants jouent dans le parc
Input: I am cold, Target: J'ai froid, Predicted: J'ai froid
Input: He waits for the bus, Target: Il attend le bus, Predicted: Il attend le
bus
Input: We cook dinner together, Target: Nous cuisinons le dîner ensemble,
Predicted: Nous cuisinons le dîner ensemble
Evaluation Loss: 0.0082, Accuracy: 100.00%
```

[ ]:

```
torch.save(encoder.state_dict(), '../../Models/hw4_2_encoder.pth')
torch.save(decoder.state_dict(), '../../Models/hw4_2_decoder.pth')
```