# Problem_3

April 15, 2024

[ ]:
```
'''
Patrick Ballou
ID: 801130521
ECGR 4106
Homework 4
Problem 3
'''
```

[ ]:
```
'\nPatrick Ballou\nID: 801130521\nECGR 4106\nHomework 4\nProblem 3\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 14:04:41 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   44C    P8              6W /   60W |    1915MiB /   4096MiB |      9%
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 = [
    ("J'ai froid", "I am cold"),
    ("Tu es fatigué", "You are tired"),
    ("Il a faim", "He is hungry"),
    ("Elle est heureuse", "She is happy"),
    ("Nous sommes amis", "We are friends"),
    ("Ils sont étudiants", "They are students"),
    ("Le chat dort", "The cat is sleeping"),
    ("Le soleil brille", "The sun is shining"),
    ("Nous aimons la musique", "We love music"),
    ("Elle parle français couramment", "She speaks French fluently"),
    ("Il aime lire des livres", "He enjoys reading books"),
    ("Ils jouent au football chaque week-end", "They play soccer every␣
 ↪weekend"),
    ("Le film commence à 19 heures", "The movie starts at 7 PM"),
    ("Elle porte une robe rouge", "She wears a red dress"),
    ("Nous cuisinons le dîner ensemble", "We cook dinner together"),
    ("Il conduit une voiture bleue", "He drives a blue car"),
    ("Ils visitent souvent des musées", "They visit museums often"),
    ("Le restaurant sert une délicieuse cuisine", "The restaurant serves␣
 ↪delicious food"),
    ("Elle étudie les mathématiques à l'université", "She studies mathematics␣
 ↪at university"),
    ("Nous regardons des films le vendredi", "We watch movies on Fridays"),
    ("Il écoute de la musique en faisant du jogging", "He listens to music␣
 ↪while jogging"),
    ("Ils voyagent autour du monde", "They travel around the world"),
    ("Le livre est sur la table", "The book is on the table"),
    ("Elle danse avec grâce", "She dances gracefully"),
    ("Nous célébrons les anniversaires avec un gâteau", "We celebrate birthdays␣
 ↪with cake"),
```

```
    ("Il travaille dur tous les jours", "He works hard every day"),
    ("Ils parlent différentes langues", "They speak different languages"),
    ("Les fleurs fleurissent au printemps", "The flowers bloom in spring"),
    ("Elle écrit de la poésie pendant son temps libre", "She writes poetry in␣
↪her free time"),
    ("Nous apprenons quelque chose de nouveau chaque jour", "We learn something␣
↪new every day"),
    ("Le chien aboie bruyamment", "The dog barks loudly"),
    ("Il chante magnifiquement", "He sings beautifully"),
    ("Ils nagent dans la piscine", "They swim in the pool"),
    ("Les oiseaux gazouillent le matin", "The birds chirp in the morning"),
    ("Elle enseigne l'anglais à l'école", "She teaches English at school"),
    ("Nous prenons le petit déjeuner ensemble", "We eat breakfast together"),
    ("Il peint des paysages", "He paints landscapes"),
    ("Ils rient de la blague", "They laugh at the joke"),
    ("L'horloge tic-tac bruyamment", "The clock ticks loudly"),
    ("Elle court dans le parc", "She runs in the park"),
    ("Nous voyageons en train", "We travel by train"),
    ("Il écrit une lettre", "He writes a letter"),
    ("Ils lisent des livres à la bibliothèque", "They read books at the␣
↪library"),
    ("Le bébé pleure", "The baby cries"),
    ("Elle étudie dur pour les examens", "She studies hard for exams"),
    ("Nous plantons des fleurs dans le jardin", "We plant flowers in the␣
↪garden"),
    ("Il répare la voiture", "He fixes the car"),
    ("Ils boivent du café le matin", "They drink coffee in the morning"),
    ("Le soleil se couche le soir", "The sun sets in the evening"),
    ("Elle danse à la fête", "She dances at the party"),
    ("Nous jouons de la musique au concert", "We play music at the concert"),
    ("Il cuisine le dîner pour sa famille", "He cooks dinner for his family"),
    ("Ils étudient la grammaire française", "They study French grammar"),
    ("La pluie tombe doucement", "The rain falls gently"),
    ("Elle chante une chanson", "She sings a song"),
    ("Nous regardons un film ensemble", "We watch a movie together"),
    ("Il dort profondément", "He sleeps deeply"),
    ("Ils voyagent à Paris", "They travel to Paris"),
    ("Les enfants jouent dans le parc", "The children play in the park"),
    ("Elle se promène le long de la plage", "She walks along the beach"),
    ("Nous parlons au téléphone", "We talk on the phone"),
    ("Il attend le bus", "He waits for the bus"),
    ("Ils visitent la tour Eiffel", "They visit the Eiffel Tower"),
    ("Les étoiles scintillent la nuit", "The stars twinkle at night"),
    ("Elle rêve de voler", "She dreams of flying"),
    ("Nous travaillons au bureau", "We work in the office"),
    ("Il étudie l'histoire", "He studies history"),
    ("Ils écoutent la radio", "They listen to the radio"),
```

```
    ("Le vent souffle doucement", "The wind blows gently"),
    ("Elle nage dans l'océan", "She swims in the ocean"),
    ("Nous dansons au mariage", "We dance at the wedding"),
    ("Il gravit la montagne", "He climbs the mountain"),
    ("Ils font de la randonnée dans la forêt", "They hike in the forest"),
    ("Le chat miaule bruyamment", "The cat meows loudly"),
    ("Elle peint un tableau", "She paints a picture"),
    ("Nous construisons un château de sable", "We build a sandcastle"),
    ("Il chante dans le chœur", "He sings in the choir")
]

SOS_token = 0
EOS_token = 1
```

# 1 Problem 3A: French to English without attention

[ ]:
```
# 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):
    """The Encoder part of the seq2seq model."""
    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)

class Decoder(nn.Module):
    """The Decoder part of the seq2seq model."""
    def __init__(self, hidden_size, output_size):
        super(Decoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output, hidden = self.gru(embedded, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden

    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):
    # 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
    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(input_tensor[ei].unsqueeze(0),␣
↪encoder_hidden)

    # 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
```

```
    for di in range(target_length):
        decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
        # Choose top1 word from decoder's output
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # Detach from history as input

        # Calculate loss
        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):
    # 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 step
            for ei in range(input_length):
                encoder_output, encoder_hidden = encoder(input_tensor[ei].
↪unsqueeze(0), encoder_hidden)

            # 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(decoder_input,␣
↪decoder_hidden)
                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)
decoder = Decoder(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: 3.0847776353690977
Epoch 5, Loss: 2.391458537877896
Epoch 10, Loss: 1.2405039028171851
Epoch 15, Loss: 0.20794280146697305
Epoch 20, Loss: 0.05064842285716246
Epoch 25, Loss: 0.03017518640154293
Epoch 30, Loss: 0.021479093558837505
Epoch 35, Loss: 0.016620042057098543
Epoch 40, Loss: 0.01352441101706481
Epoch 45, Loss: 0.011378455714851927
Epoch 50, Loss: 0.009802569154806633
```

[ ]:
```
evaluate_and_show_examples(encoder, decoder, dataloader, criterion)
```

```
Input: Nous travaillons au bureau, Target: We work in the office, Predicted: We
work in the office
Input: Le soleil brille, Target: The sun is shining, Predicted: The sun is
shining
Input: Il peint des paysages, Target: He paints landscapes, Predicted: He paints
landscapes
Input: Nous jouons de la musique au concert, Target: We play music at the
concert, Predicted: We play music at the concert
Input: Nous parlons au téléphone, Target: We talk on the phone, Predicted: We
talk on the phone
Input: Elle écrit de la poésie pendant son temps libre, Target: She writes
poetry in her free time, Predicted: She writes poetry in her free time
Input: Il répare la voiture, Target: He fixes the car, Predicted: He fixes the
car
Input: Nous regardons des films le vendredi, Target: We watch movies on Fridays,
Predicted: We watch movies on Fridays
Input: Il attend le bus, Target: He waits for the bus, Predicted: He waits for
the bus
Input: Elle se promène le long de la plage, Target: She walks along the beach,
Predicted: She walks along the beach
Evaluation Loss: 0.0094, Accuracy: 100.00%
```

[ ]:
```
torch.save(encoder.state_dict(), '../../Models/hw4_3a_encoder.pth')
torch.save(decoder.state_dict(), '../../Models/hw4_3b_decoder.pth')
```

## 2 Problem 3B: French to English with attention

[ ]:
```
# 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)
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: 3.0278098774310176
Epoch 5, Loss: 2.410202463983974
Epoch 10, Loss: 1.2605623107820512
Epoch 15, Loss: 0.30540076114960707
```

```
Epoch 20, Loss: 0.06631536839498772
Epoch 25, Loss: 0.033590564190945145
Epoch 30, Loss: 0.021999660827998185
Epoch 35, Loss: 0.01609092903627002
Epoch 40, Loss: 0.012645105581302092
Epoch 45, Loss: 0.01031990974096739
Epoch 50, Loss: 0.008761231182459866
```

[ ]:
```
evaluate_and_show_examples(encoder, decoder, dataloader, criterion)
```

```
Input: Ils voyagent à Paris, Target: They travel to Paris, Predicted: They
travel to Paris
Input: Nous aimons la musique, Target: We love music, Predicted: We love music
Input: Tu es fatigué, Target: You are tired, Predicted: You are tired
Input: Il cuisine le dîner pour sa famille, Target: He cooks dinner for his
family, Predicted: He cooks dinner for his family
Input: Il conduit une voiture bleue, Target: He drives a blue car, Predicted: He
drives a blue car
Input: Il chante magnifiquement, Target: He sings beautifully, Predicted: He
sings beautifully
Input: Nous jouons de la musique au concert, Target: We play music at the
concert, Predicted: We play music at the concert
Input: Ils boivent du café le matin, Target: They drink coffee in the morning,
Predicted: They drink coffee in the morning
Input: Le chat miaule bruyamment, Target: The cat meows loudly, Predicted: The
cat meows loudly
Input: Il aime lire des livres, Target: He enjoys reading books, Predicted: He
enjoys reading books
Evaluation Loss: 0.0082, Accuracy: 100.00%
```

[ ]:
```
torch.save(encoder.state_dict(), '../../Models/hw4_3b_encoder.pth')
torch.save(decoder.state_dict(), '../../Models/hw4_3b_decoder.pth')
```