Problem_4

April 27, 2024

[ ]:
```
'''
Patrick Ballou
ID: 801130521
ECGR 4106
Homework 5
Problem 4
'''
```

[ ]:
```
'\nPatrick Ballou\nID: 801130521\nECGR 4106\nHomework 5\nProblem 4\n'
```

[ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch import cuda
import math
import time
from torch.utils.data import DataLoader, Dataset
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
#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
Sat Apr 27 23:48:21 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   62C    P0             31W /   60W |    1701MiB /   4096MiB |     58%
Default |
|                                         |                        |
N/A |
+-----------------------------------------+----------------------+----------
----------+

+---------------------------------------------------------------------------
----------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name
GPU Memory |
|        ID   ID
Usage      |
|===========================================================================
==========|
|    0   N/A  N/A      1448    C+G   …ekyb3d8bbwe\PhoneExperienceHost.exe
N/A      |
|    0   N/A  N/A      1888    C+G   …5n1h2txyewy\ShellExperienceHost.exe
N/A      |
|    0   N/A  N/A      2444    C+G   …Brave-Browser\Application\brave.exe
N/A      |
|    0   N/A  N/A      4036    C+G   C:\Windows\explorer.exe
N/A      |
|    0   N/A  N/A      5740    C+G   …les\Microsoft OneDrive\OneDrive.exe
N/A      |
|    0   N/A  N/A     10916    C+G   …91.0_x64__8wekyb3d8bbwe\GameBar.exe
N/A      |
|    0   N/A  N/A     11388    C+G   …AppData\Roaming\Spotify\Spotify.exe
N/A      |
|    0   N/A  N/A     12940    C+G   …siveControlPanel\SystemSettings.exe
N/A      |
|    0   N/A  N/A     17204      C   …ri\anaconda3\envs\dl_env\python.exe
N/A      |
|    0   N/A  N/A     18068    C+G   …CBS_cw5n1h2txyewy\TextInputHost.exe
```

```
N/A      |
|    0   N/A  N/A     18388    C+G   …ta\Local\Programs\Notion\Notion.exe
N/A      |
|    0   N/A  N/A     21080    C+G   …aam7r\AcrobatNotificationClient.exe
N/A      |
|    0   N/A  N/A     21448    C+G   …e Stream\90.0.3.0\GoogleDriveFS.exe
N/A      |
|    0   N/A  N/A     21556    C+G   …b3d8bbwe\Microsoft.Media.Player.exe
N/A      |
|    0   N/A  N/A     22800    C+G   …1300.0_x64__8j3eq9eme6ctt\IGCC.exe
N/A      |
|    0   N/A  N/A     24160    C+G   …Programs\Microsoft VS Code\Code.exe
N/A      |
|    0   N/A  N/A     26360    C+G   …t.LockApp_cw5n1h2txyewy\LockApp.exe
N/A      |
|    0   N/A  N/A     27584    C+G   …Search_cw5n1h2txyewy\SearchApp.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
```

[ ]:
```
# 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)
```

[ ]:
```
class TranslationTransformer(nn.Module):
    def __init__(self, src_vocab_size, tgt_vocab_size, d_model, nhead,␣
↪num_encoder_layers, num_decoder_layers, dim_feedforward, dropout,␣
↪activation='relu'):
        super(TranslationTransformer, self).__init__()

        # Source and target embeddings
        self.src_embedding = nn.Embedding(src_vocab_size, d_model)
        self.tgt_embedding = nn.Embedding(tgt_vocab_size, d_model)

        # Positional Encoding (not learned)
        self.positional_encoding = PositionalEncoding(d_model, dropout)

        # Transformer Model
        self.transformer = nn.Transformer(d_model=d_model, nhead=nhead,␣
↪num_encoder_layers=num_encoder_layers,
```

```
                                                    ␣
↪num_decoder_layers=num_decoder_layers, dim_feedforward=dim_feedforward,␣
↪dropout=dropout, activation=activation, batch_first=True)

        # Output linear layer
        self.output_layer = nn.Linear(d_model, tgt_vocab_size)

    def forward(self, src, tgt):
        src = self.src_embedding(src) * math.sqrt(self.transformer.d_model)
        tgt = self.tgt_embedding(tgt) * math.sqrt(self.transformer.d_model)

        src = self.positional_encoding(src)
        tgt = self.positional_encoding(tgt)

        # Shift tgt input for decoder training: Skip the last token from the␣
↪target input
        tgt_input = tgt[:, :-1]  # Remove the last token for decoder input
        memory = self.transformer.encoder(src)
        outs = self.transformer.decoder(tgt_input, memory)

        return self.output_layer(outs)


class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * -(math.log(10000.0) /
↪ d_model))
        pe = torch.zeros(max_len, 1, d_model)
        pe[:, 0, 0::2] = torch.sin(position * div_term)
        pe[:, 0, 1::2] = torch.cos(position * div_term)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + self.pe[:x.size(0)]
        return self.dropout(x)
```

[ ]:
```
def train_transformer(model, dataloader, optimizer, criterion, epochs):
    model.train()
    for epoch in range(epochs):
        total_loss = 0  # Initialize total loss
        for input_tensor, target_tensor in dataloader:
            input_tensor, target_tensor = input_tensor.to(device),␣
↪target_tensor.to(device)
```

```
            optimizer.zero_grad()

            # Forward pass
            output = model(input_tensor, target_tensor)  # Notice target_tensor␣
 ↪is used directly

            # Flatten output and calculate loss based on the offset targets
            loss = criterion(output.view(-1, output.size(-1)), target_tensor[:,␣
 ↪1:].contiguous().view(-1))

            # Backward pass and optimization
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

        print(f'Epoch {epoch+1}/{epochs}, Loss: {total_loss / len(dataloader)}')

def evaluate(model, dataloader, criterion):
    model.eval()
    total_loss = 0
    total_correct = 0
    total_samples = 0

    with torch.no_grad():
        for input_tensor, target_tensor in dataloader:
            input_tensor, target_tensor = input_tensor.to(device),␣
 ↪target_tensor.to(device)

            # Forward pass through the transformer model
            output = model(input_tensor, target_tensor)  # Assuming your model␣
 ↪expects this slicing
            output_flat = output.view(-1, output.size(-1))
            target_flat = target_tensor[:, 1:].contiguous().view(-1)  # Flatten␣
 ↪target tensor

            # Calculate loss
            loss = criterion(output_flat, target_flat)
            total_loss += loss.item()

            # Calculate accuracy
            _, predictions = torch.max(output_flat, 1)
            correct = (predictions == target_flat).sum().item()
            total_correct += correct
            total_samples += target_flat.size(0)

    average_loss = total_loss / len(dataloader)
```

```
    accuracy = total_correct / total_samples
    print(f'Evaluation Loss: {average_loss:.4f}, Accuracy: {accuracy:.4f}')
```

[ ]:
```
# dim_model: The size of the input and output feature dimension in the model.␣
 ↪Also known as 'd_model'.
# This defines the size of the embedding layer as well as the hidden layers in␣
 ↪the model's multi-head
# attention mechanisms and the feedforward neural network. It affects the␣
 ↪model's capacity and the
# complexity of the relationships it can learn.
dim_model = 512

# nhead: The number of heads in the multi-head attention mechanisms. Each head␣
 ↪operates on a different
# part of the model's embedding vector, allowing the model to simultaneously␣
 ↪attend to information
# from different representation subspaces at different positions. A higher␣
 ↪number allows better learning
# of relationships but increases computational complexity.
nhead = 2

# num_layers: The number of sub-encoder and sub-decoder layers in the model.␣
 ↪Each layer consists of a
# multi-head attention mechanism and a feedforward neural network. More layers␣
 ↪allow the model to learn
# more complex representations but can make training more difficult and␣
 ↪increase the risk of overfitting.
num_layers = 4

# dim_feedforward: The dimension of the feedforward network model in each layer.
 ↪ This defines the
# size of the inner layer of the feedforward networks and affects the model's␣
 ↪capacity to learn
# complex functions within each layer.
dim_feedforward = 1024

# dropout: The dropout rate for layers during training. Dropout randomly zeros␣
 ↪some of the elements
# of the input tensor with probability equal to the dropout rate during␣
 ↪training. It is a regularization
# method to prevent overfitting by reducing the chance of complex␣
 ↪co-adaptations on training data.
dropout = 0.1

# epochs: The number of times the training data is iterated over. More epochs␣
 ↪can lead to better model
```

```
# learning, but also increase the risk of overfitting if not combined with␣
 ↪adequate regularization.
epochs = 50

# learning_rate: The step size used for each iteration of the weight update. If␣
 ↪set too high, training
# may diverge; if set too low, training can become too slow and possibly get␣
 ↪stuck in suboptimal solutions.
# This value affects how quickly the model learns and stabilizes in a␣
 ↪potentially optimal training configuration.
learning_rate = 0.00007

# activation: The activation function used in the feedforward neural network␣
 ↪layers. 'GELU' (Gaussian Error
# Linear Unit) provides smoother nonlinearities than 'ReLU', influencing how␣
 ↪effectively the network can
# learn complex patterns in the data.
activation = 'relu'
```

[ ]:
```
# Initialize the dataset and DataLoader
e2f_dataset = EngFrDataset(text)

from torch.nn.utils.rnn import pad_sequence

def collate_batch(batch):
    input_tensors, target_tensors = zip(*batch)
    input_tensors_padded = pad_sequence(input_tensors, batch_first=True,␣
 ↪padding_value=EOS_token)
    target_tensors_padded = pad_sequence(target_tensors, batch_first=True,␣
 ↪padding_value=EOS_token)

    return input_tensors_padded, target_tensors_padded


dataloader = DataLoader(e2f_dataset, batch_size=1, shuffle=True,␣
 ↪collate_fn=collate_batch)

# Model parameters
input_size = len(e2f_dataset.eng_vocab.word2index)
output_size = len(e2f_dataset.fr_vocab.word2index)
model = TranslationTransformer(input_size, output_size, dim_model, nhead,␣
 ↪num_layers, num_layers, dim_feedforward, dropout, activation).to(device)

# Optimizer and Loss Function
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

```
criterion = nn.CrossEntropyLoss(ignore_index=e2f_dataset.fr_vocab.
 ↪word2index["<EOS>"]).to(device)

# Train and evaluate the model
train_transformer(model, dataloader, optimizer, criterion, epochs)
```

```
Epoch 1/50, Loss: 5.416741854184634
Epoch 2/50, Loss: 5.003405595754648
Epoch 3/50, Loss: 4.491008928843907
Epoch 4/50, Loss: 3.7091021259109693
Epoch 5/50, Loss: 3.0237946154235247
Epoch 6/50, Loss: 2.471309867772189
Epoch 7/50, Loss: 2.0702714029844707
Epoch 8/50, Loss: 1.7732423303963303
Epoch 9/50, Loss: 1.4836126194371806
Epoch 10/50, Loss: 1.3246807734687607
Epoch 11/50, Loss: 1.082959336506856
Epoch 12/50, Loss: 0.9134462262902941
Epoch 13/50, Loss: 0.8298692788396563
Epoch 14/50, Loss: 0.7038294821977615
Epoch 15/50, Loss: 0.5341624849802488
Epoch 16/50, Loss: 0.4613327703305653
Epoch 17/50, Loss: 0.3573510224943037
Epoch 18/50, Loss: 0.27469468339310066
Epoch 19/50, Loss: 0.2337330138528502
Epoch 20/50, Loss: 0.1755601099946282
Epoch 21/50, Loss: 0.1455645002424717
Epoch 22/50, Loss: 0.11843608281055054
Epoch 23/50, Loss: 0.10669672244845273
Epoch 24/50, Loss: 0.08892064498035938
Epoch 25/50, Loss: 0.08235269267257157
Epoch 26/50, Loss: 0.07594343492543543
Epoch 27/50, Loss: 0.06669570304840416
Epoch 28/50, Loss: 0.05809499697106612
Epoch 29/50, Loss: 0.052386588308137735
Epoch 30/50, Loss: 0.04976699046500317
Epoch 31/50, Loss: 0.04885582272689064
Epoch 32/50, Loss: 0.04167155209057904
Epoch 33/50, Loss: 0.036581599842998885
Epoch 34/50, Loss: 0.04045485540643915
Epoch 35/50, Loss: 0.0848784450949593
Epoch 36/50, Loss: 0.087175820216343
Epoch 37/50, Loss: 0.06406824268981234
Epoch 38/50, Loss: 0.1966614941763994
Epoch 39/50, Loss: 0.1259897727422513
Epoch 40/50, Loss: 0.14111032750602667
Epoch 41/50, Loss: 0.14512585835265263
Epoch 42/50, Loss: 0.08468446658029184
```

```
Epoch 43/50, Loss: 0.04957139550091384
Epoch 44/50, Loss: 0.028475230886274343
Epoch 45/50, Loss: 0.021220288257562107
Epoch 46/50, Loss: 0.02008584327995777
Epoch 47/50, Loss: 0.01714297052013216
Epoch 48/50, Loss: 0.018487843497378097
Epoch 49/50, Loss: 0.04378235308671152
Epoch 50/50, Loss: 0.039821667143131616
```

[ ]:
```
evaluate(model, dataloader, criterion)
```

```
Evaluation Loss: 0.0047, Accuracy: 0.7800
```

[ ]:
```
torch.save(model.state_dict(), '../../Models/hw5_4.pth')
```