Problem_3

April 27, 2024

[ ]:
```
'''
Patrick Ballou
ID: 801130521
ECGR 4106
Homework 5
Problem 3
'''
```

[ ]:
```
'\nPatrick Ballou\nID: 801130521\nECGR 4106\nHomework 5\nProblem 3\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 23:50:18 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   64C    P0             31W /   60W |    1729MiB /   4096MiB |     56%
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     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     25972      C   …ri\anaconda3\envs\dl_env\python.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      |
+-------------------------------------------------------------------------------
----------+
```

[ ]:
```python
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"),
```

```python
    ("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
```

[ ]:
```python
# 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 = 'gelu'
```

[ ]:
```
# 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: 5.592086915845995
Epoch 2/50, Loss: 4.587467128580267
Epoch 3/50, Loss: 3.600221880070575
Epoch 4/50, Loss: 2.9157672772159824
Epoch 5/50, Loss: 2.3845219437952165
Epoch 6/50, Loss: 2.0404705512639763
Epoch 7/50, Loss: 1.726821508984287
Epoch 8/50, Loss: 1.5184345690460948
Epoch 9/50, Loss: 1.3232076490273723
Epoch 10/50, Loss: 1.133816942717735
Epoch 11/50, Loss: 0.9107692664997145
Epoch 12/50, Loss: 0.8286200464255624
Epoch 13/50, Loss: 0.6209167738916812
Epoch 14/50, Loss: 0.48454028053523657
Epoch 15/50, Loss: 0.4031976800315179
Epoch 16/50, Loss: 0.32079060859494396
Epoch 17/50, Loss: 0.24224924320330868
Epoch 18/50, Loss: 0.20441417028377581
Epoch 19/50, Loss: 0.17295378777984674
Epoch 20/50, Loss: 0.16498804605239398
Epoch 21/50, Loss: 0.15085802371190352
Epoch 22/50, Loss: 0.10973650374292553
Epoch 23/50, Loss: 0.09391042827205225
Epoch 24/50, Loss: 0.08778448489966331
Epoch 25/50, Loss: 0.07183856658589144
Epoch 26/50, Loss: 0.06552872309678948
Epoch 27/50, Loss: 0.06028724244082129
Epoch 28/50, Loss: 0.054431612470320294
Epoch 29/50, Loss: 0.048529400768411626
Epoch 30/50, Loss: 0.05027855064284492
Epoch 31/50, Loss: 0.04771107660800025
Epoch 32/50, Loss: 0.04279091093060258
Epoch 33/50, Loss: 0.03956577346309439
Epoch 34/50, Loss: 0.06072653215620425
Epoch 35/50, Loss: 0.050144447969248544
Epoch 36/50, Loss: 0.04787956537412746
Epoch 37/50, Loss: 0.036392105961000766
Epoch 38/50, Loss: 0.029908382442670984
Epoch 39/50, Loss: 0.02927132530815222
Epoch 40/50, Loss: 0.026540259758083076
Epoch 41/50, Loss: 0.02403189573291834
Epoch 42/50, Loss: 0.022314108726749948
Epoch 43/50, Loss: 0.022490420281983815
Epoch 44/50, Loss: 0.02220368350239156
Epoch 45/50, Loss: 0.020842568564845565
Epoch 46/50, Loss: 0.019099102543013825
Epoch 47/50, Loss: 0.018284767086000797
Epoch 48/50, Loss: 0.01769325253856066
Epoch 49/50, Loss: 0.017796896035214522
Epoch 50/50, Loss: 0.016353682185486926
```

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

```
Evaluation Loss: 0.0043, Accuracy: 0.7920
```

```
[ ]: torch.save(model.state_dict(), '../../Models/hw5_3.pth')
```