nlp-assignment

September 22, 2024

SAMEER KHATWANI
PRN: 22070126099

[1]:
```
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
import numpy as np
from tqdm import tqdm
import time
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\samee\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[1]: True

[2]:
```
# Check if CUDA is available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Using device: cuda
```

[3]:
```
# Load and preprocess data
df = pd.read_csv(r'C:\Users\samee\OneDrive\Documents\SEM5\NLP\Assignment␣
 ↪3\Hindi_English_Truncated_Corpus.csv')  # Replace with your dataset path
df.dropna(inplace=True)
df = df.sample(frac=1, random_state=42)  # Shuffle the data
src_lang = df['english_sentence'].astype(str).tolist()
tgt_lang = df['hindi_sentence'].astype(str).tolist()
```

[4]:
```
def create_vocab(sentences):
    vocab = set()
    for sentence in sentences:
        vocab.update(str(sentence).split())
    return vocab
```

[5]:
```
src_vocab = create_vocab(src_lang)
tgt_vocab = create_vocab(tgt_lang)
src_vocab_size = len(src_vocab) + 1  # Add 1 for padding
tgt_vocab_size = len(tgt_vocab) + 1  # Add 1 for padding
```

[6]:
```
# Create word to index mappings
src_word2idx = {word: idx + 1 for idx, word in enumerate(src_vocab)}  # Start␣
 ↪indexing from 1
tgt_word2idx = {word: idx + 1 for idx, word in enumerate(tgt_vocab)}  # Start␣
 ↪indexing from 1
src_word2idx['<PAD>'] = 0  # Padding index
tgt_word2idx['<PAD>'] = 0  # Padding index

src_idx2word = {idx: word for word, idx in src_word2idx.items()}
tgt_idx2word = {idx: word for word, idx in tgt_word2idx.items()}
```

[7]:
```
# Convert sentences to indices
def sentence_to_indices(sentence, word2idx):
    return [word2idx.get(word, 0) for word in str(sentence).split()]
```

[8]:
```
src_indices = [sentence_to_indices(sentence, src_word2idx) for sentence in␣
 ↪src_lang]
tgt_indices = [sentence_to_indices(sentence, tgt_word2idx) for sentence in␣
 ↪tgt_lang]
```

[9]:
```
# Pad sequences
max_src_len = max(len(s) for s in src_indices)
max_tgt_len = max(len(s) for s in tgt_indices)

src_indices = [s + [0] * (max_src_len - len(s)) for s in src_indices]
tgt_indices = [s + [0] * (max_tgt_len - len(s)) for s in tgt_indices]
```

[10]:
```
# Create dataset
class TranslationDataset(Dataset):
    def __init__(self, src, tgt):
        self.src = src
        self.tgt = tgt

    def __len__(self):
        return len(self.src)

    def __getitem__(self, idx):
        return torch.tensor(self.src[idx]), torch.tensor(self.tgt[idx])
```

[11]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(src_indices, tgt_indices,␣
 ↪test_size=0.2, random_state=42)
```

[12]:
```
# Create dataloaders
train_dataset = TranslationDataset(X_train, y_train)
test_dataset = TranslationDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

[13]:
```
# Define the Seq2Seq model using LSTM with embedding layers
class Seq2SeqLSTM(nn.Module):
    def __init__(self, src_vocab_size, tgt_vocab_size, hidden_size):
        super(Seq2SeqLSTM, self).__init__()
        self.src_embedding = nn.Embedding(src_vocab_size, hidden_size)
        self.tgt_embedding = nn.Embedding(tgt_vocab_size, hidden_size)
        self.encoder = nn.LSTM(hidden_size, hidden_size, batch_first=True)
        self.decoder = nn.LSTM(hidden_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, tgt_vocab_size)

    def forward(self, src, tgt):
        src_embedded = self.src_embedding(src)
        tgt_embedded = self.tgt_embedding(tgt)
        _, (hidden, cell) = self.encoder(src_embedded)
        output, _ = self.decoder(tgt_embedded, (hidden, cell))
        return self.fc(output)
```

[14]:
```
# Initialize the Seq2Seq model
model = Seq2SeqLSTM(src_vocab_size, tgt_vocab_size, hidden_size=256).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss(ignore_index=0)  # Use padding index
```

[15]:
```
# Training loop
num_epochs = 2
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    progress_bar = tqdm(enumerate(train_loader), total=len(train_loader),␣
 ↪desc=f"Epoch {epoch+1}/{num_epochs}")

    for batch_idx, (src, tgt) in progress_bar:
        src, tgt = src.to(device), tgt.to(device)
        optimizer.zero_grad()

        # Forward pass
        output = model(src, tgt[:, :-1])
        loss = criterion(output.reshape(-1, tgt_vocab_size), tgt[:, 1:].
 ↪reshape(-1))
        loss.backward()
        optimizer.step()
```

```
        total_loss += loss.item()
        avg_loss = total_loss / (batch_idx + 1)

        # Update progress bar
        progress_bar.set_postfix({
            'Loss': f'{avg_loss:.4f}',
            'Batch': f'{batch_idx+1}/{len(train_loader)}'
        })

    print(f"Epoch {epoch+1}/{num_epochs} completed. Average Loss: {avg_loss:.
↪4f}")

    # Validation
    model.eval()
    val_loss = 0
    with torch.no_grad():
        for src, tgt in test_loader:
            src, tgt = src.to(device), tgt.to(device)
            output = model(src, tgt[:, :-1])
            loss = criterion(output.reshape(-1, tgt_vocab_size), tgt[:, 1:].
↪reshape(-1))
            val_loss += loss.item()

    val_loss /= len(test_loader)
    print(f"Validation Loss: {val_loss:.4f}")

    # Save the best model
    torch.save(model.state_dict(), 'best_translation_model.pth')
    print("Model saved!")
```

```
Epoch 1/2: 100%|      | 6381/6381 [6:06:21<00:00,  3.44s/it, Loss=6.4746,
Batch=6381/6381]

Epoch 1/2 completed. Average Loss: 6.4746
Validation Loss: 5.8262
Model saved!

Epoch 2/2: 100%|      | 6381/6381 [5:21:41<00:00,  3.02s/it, Loss=5.1379,
Batch=6381/6381]

Epoch 2/2 completed. Average Loss: 5.1379
Validation Loss: 5.3763
Model saved!
```

[16]:
```
# Inference function for translation
def translate(model, test_loader, src_idx2word, tgt_idx2word, device,␣
↪max_tgt_len):
    model.eval()
```

```
    all_translations = []
    all_references = []

    for src, tgt in tqdm(test_loader, desc="Translating"):
        src, tgt = src.to(device), tgt.to(device)
        for i in range(len(src)):
            src_sentence = ' '.join([src_idx2word.get(idx.item(), "") for idx␣
 ↪in src[i] if idx.item() != 0])
            tgt_sentence = ' '.join([tgt_idx2word.get(idx.item(), "") for idx␣
 ↪in tgt[i] if idx.item() != 0])

            src_tensor = torch.tensor([src[i].tolist()], device=device)
            with torch.no_grad():
                _, (hidden, cell) = model.encoder(model.
 ↪src_embedding(src_tensor))
                tgt_tensor = torch.zeros(1, 1, dtype=torch.long, device=device)

                output_sentence = []
                for _ in range(max_tgt_len):
                    output, (hidden, cell) = model.decoder(model.
 ↪tgt_embedding(tgt_tensor), (hidden, cell))
                    output = model.fc(output)
                    predicted = output.argmax(2).item()
                    if predicted == 0:
                        break
                    output_sentence.append(tgt_idx2word.get(predicted, ""))
                    tgt_tensor = torch.tensor([[predicted]], device=device)

            all_translations.append(' '.join(output_sentence))
            all_references.append(tgt_sentence)

    return all_translations, all_references
```

[17]:
```
# Translate and calculate BLEU score
from nltk.translate.bleu_score import corpus_bleu
model.load_state_dict(torch.load('best_translation_model.pth'))
translations, references = translate(model, test_loader, src_idx2word,␣
 ↪tgt_idx2word, device, max_tgt_len=20)
```

```
C:\Users\samee\AppData\Local\Temp\ipykernel_22028\3140131770.py:3:
FutureWarning: You are using `torch.load` with `weights_only=False` (the current
default value), which uses the default pickle module implicitly. It is possible
to construct malicious pickle data which will execute arbitrary code during
unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
```

```
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  model.load_state_dict(torch.load('best_translation_model.pth'))
Translating: 100%|      | 1596/1596 [13:17<00:00,  2.00it/s]
```

[79]:
```python
import nltk
import os
from urllib.request import urlretrieve

# Define the directory for nltk_data
nltk_data_dir = 'C:/nltk_data'
if not os.path.exists(nltk_data_dir):
    os.makedirs(nltk_data_dir)

# Redownload punkt manually if necessary
punkt_url = 'https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/packages/
 ↪tokenizers/punkt.zip'
punkt_path = os.path.join(nltk_data_dir, 'punkt.zip')
urlretrieve(punkt_url, punkt_path)

# Extract the punkt package
import zipfile
with zipfile.ZipFile(punkt_path, 'r') as zip_ref:
    zip_ref.extractall(nltk_data_dir)

# Load NLTK data path and tokenizer
nltk.data.path.append(nltk_data_dir)
nltk.download('punkt', download_dir=nltk_data_dir)

# Now perform tokenization
from nltk.tokenize import word_tokenize

processed_translations = [word_tokenize(t.lower()) for t in translations]
processed_references = [[word_tokenize(r.lower())] for r in references]
```

```
[nltk_data] Downloading package punkt to C:/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[80]:
```python
# Calculate BLEU score
bleu_score = corpus_bleu(processed_references, processed_translations)
print(f"BLEU Score: {bleu_score:.4f}")
```

```
BLEU Score: 0.0030
```

[81]:
```
# Print some example translations
num_examples = 5
print("\nExample Translations:")
for i in range(min(num_examples, len(translations))):
    print(f"Source: {references[i]}")
    print(f"Translation: {translations[i]}")
    print(f"Reference: {references[i]}")
    print()
```

```
Example Translations:
Source:     -


Translation:

Reference:     -



Source:
             ,                          .
Translation:

Reference:
                ,                          .

Source:
Translation:                            |
    |
Reference:

Source:                ,       ,      .
Translation:
563
Reference:               ,       ,      .

Source:
Translation:                            |
   .
Reference:
```