Exercise 2

Neural machine translation with attention

By: Daniel Mehta

Imports and Config

In [1]:
```
import os
import re
import random
from pathlib import Path
from collections import Counter
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from tqdm import tqdm
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Setting up seed
SEED = 5501
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
```

In [3]:
```
# setting device
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {DEVICE}")
```

```
Using device: cuda
```

Dataset Path Setup

In [4]:
```python
# setting path to dataset
data_dir = Path("spa-eng")
data_path = data_dir / "spa.txt"
```

In [5]:
```python
if not data_path.exists():
    raise FileNotFoundError(f"Dataset not found at {data_path}")
print(f"Dataset located at: {data_path}")
```

```
Dataset located at: spa-eng\spa.txt
```

Data Exploration and Cleaning

In [6]:
```python
# reading the file and split into lines
with open(data_path, "r", encoding="utf-8") as f:
    lines = f.read().strip().split("\n")
```

In [7]:
```python
print(f"Total sentence pairs in file: {len(lines)}")
print("Sample lines:")
for i in range(5):
    print(lines[i])
```

```
Total sentence pairs in file: 142511
Sample lines:
Go.	Ve.	CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #4986655 (cueyayotl)
Go.	Vete.	CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #4986656 (cueyayotl)
Go.	Vaya.	CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #4986657 (cueyayotl)
Go.	Váyase.	CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #6586271 (arh)
Hi.	Hola.	CC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #431975 (Leono)
```

In [8]:
```
#Separating into English and Spanish
pairs = [line.split("\t") for line in lines]
english_sentences = [pair[0] for pair in pairs] #English (target)
spanish_sentences = [pair[1] for pair in pairs] #Spanish (source)

print("\nExample pair:")
print("EN:", english_sentences[0])
print("ES:", spanish_sentences[0])
```

```
Example pair:
EN: Go.
ES: Ve.
```

Tokenization & vocab building

In [9]:
```
#  start and end tokens to the English targets
START_TOKEN="<start>"
END_TOKEN="<end>"

english_sentences = [f"{START_TOKEN} {s} {END_TOKEN}" for s in english_sentences]
```

In [10]:
```
#Basic tokenization
#lowercase, split on spaces, strip punctuation
def tokenize(text):
    return text.lower().strip().split()
```

In [11]:
```
#Tokenize all the sentences
tokenized_es = [tokenize(s) for s in spanish_sentences]
tokenized_en = [tokenize(s) for s in english_sentences]
```

In [12]:
```
#Build vocabularies
def build_vocab(tokenized_sents,min_freq=1):
    counter = Counter(token for sent in tokenized_sents for token in sent)
    vocab = {token: idx+2 for idx, (token, freq) in enumerate(counter.items()) if freq >= min_freq}
    vocab["<pad>"] =0
```

```
    vocab["<unk>"] =1
    return vocab
```

In [13]:
```
src_vocab = build_vocab(tokenized_es)
tgt_vocab = build_vocab(tokenized_en)
```

In [14]:
```
# reverse look up for decoding
src_idx2word = {idx: word for word,idx in src_vocab.items()}
tgt_idx2word = {idx: word for word,idx in tgt_vocab.items()}
```

In [15]:
```
print(f"Source vocab size (ES): {len(src_vocab)}")
print(f"Target vocab size (EN): {len(tgt_vocab)}")
```

```
Source vocab size (ES): 46045
Target vocab size (EN): 25767
```

Convert sentences to index tensors

In [16]:
```
#Numericalize tokenized sentences
def numericalize(tokenized_sents,vocab):
    return [torch.tensor([vocab.get(tok,vocab["<unk>"]) for tok in sent],dtype=torch.long)
            for sent in tokenized_sents]
```

In [17]:
```
src_tensors =numericalize(tokenized_es, src_vocab)
tgt_input_tensors = numericalize([sent[:-1] for sent in tokenized_en], tgt_vocab)# without <end>
tgt_target_tensors = numericalize([sent[1:] for sent in tokenized_en], tgt_vocab)# without <start>
```

In [18]:
```
# Pad the sequences
src_tensors = pad_sequence(src_tensors,batch_first=True, padding_value=src_vocab["<pad>"])
tgt_input_tensors = pad_sequence(tgt_input_tensors,batch_first=True, padding_value=tgt_vocab["<pad>"])
tgt_target_tensors = pad_sequence(tgt_target_tensors,batch_first=True, padding_value=tgt_vocab["<pad>"])
```

In [19]:
```
print(f"Example ES tensor: {src_tensors[0]}")
print(f"Example EN input tensor: {tgt_input_tensors[0]}")
print(f"Example EN target tensor: {tgt_target_tensors[0]}")
```

```
Example ES tensor: tensor([2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
Example EN input tensor: tensor([2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
Example EN target tensor: tensor([3, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Dataloader Setup

In [20]:
```python
# Combineing into dataset
dataset=list(zip(src_tensors,tgt_input_tensors,tgt_target_tensors))
```

In [21]:
```python
# Train/validation split (80/20)
split_idx=int(len(dataset)*0.8)
train_data=dataset[:split_idx]
val_data=dataset[split_idx:]

BATCH_SIZE =64

train_loader = DataLoader(train_data,batch_size=BATCH_SIZE,shuffle=True)
val_loader = DataLoader(val_data,batch_size=BATCH_SIZE)

print(f"Train batches: {len(train_loader)}, Val batches: {len(val_loader)}")
```

```
Train batches: 1782, Val batches: 446
```

Model Architecture

In [22]:
```python
class Encoder(nn.Module):
    def __init__(self, input_vocab_size, embed_dim, hidden_dim):
        super().__init__()
```

```
        self.embedding = nn.Embedding(input_vocab_size, embed_dim)
        self.gru = nn.GRU(embed_dim, hidden_dim, batch_first=True, bidirectional=True)
        self.fc = nn.Linear(hidden_dim * 2, hidden_dim) # project bi GRU output to hidden_dim

    def forward(self, src_idxs):
        # src_idxs: (batch, src_len)
        embedded = self.embedding(src_idxs) # (batch, src_len, embed_dim)
        outputs, hidden = self.gru(embedded)# outputs: (batch, src_len, hidden_dim*2)

        # Mergeing the bidirectional hidden states
        hidden =torch.tanh(self.fc(torch.cat((hidden[-2, :, :], hidden[-1, :, :]), dim=1))) # (batch,hidden_dim)
        hidden =hidden.unsqueeze(0) # (1, batch, hidden_dim)

        return outputs,hidden #outputs for attention, hidden for decoder init


class LuongAttention(nn.Module):
    def __init__(self, enc_dim, dec_dim):
        super().__init__()
        self.attn = nn.Linear(enc_dim, dec_dim) #project encoder outputs to decoder dim

    def forward(self, decoder_hidden, encoder_outputs, src_mask=None):
        # decoder_hidden: (1, batch, dec_dim)
        # encoder_outputs: (batch, src_len, enc_dim)
        # src_mask: (batch, src_len) -> 1 for real tokens, 0 for PAD

        # Project encoder outputs to decoder hidden size
        proj_enc = self.attn(encoder_outputs) #(batch, src_len, dec_dim)

        # Repeat decoder hidden state across src_len
        decoder_hidden = decoder_hidden.permute(1, 0, 2) #(batch, 1, dec_dim)

        # score: batch matrix multiply
        scores = torch.bmm(proj_enc, decoder_hidden.transpose(1, 2)) #(batch, src_len, 1)

        if src_mask is not None:
            # mask pad positions before softmax
            scores = scores.masked_fill(src_mask.unsqueeze(2) == 0, float('-inf'))

        attn_weights = torch.softmax(scores, dim=1) #(batch, src_len, 1)
```

```
        # Context vector
        context = torch.bmm(attn_weights.transpose(1, 2), encoder_outputs) #(batch, 1, enc_dim)

        return context, attn_weights


class Decoder(nn.Module):
    def __init__(self,output_vocab_size, embed_dim,hidden_dim, enc_dim):
        super().__init__()
        self.embedding = nn.Embedding(output_vocab_size, embed_dim)
        self.attention = LuongAttention(enc_dim,hidden_dim)
        self.ctx_proj = nn.Linear(enc_dim,hidden_dim) #project context to decoder dim
        self.gru = nn.GRU(embed_dim+hidden_dim, hidden_dim, batch_first=True)
        self.fc_out = nn.Linear(hidden_dim*2,output_vocab_size)

    def forward(self,tgt_input_idxs, hidden, encoder_outputs):
        # tgt_input_idxs:(batch, tgt_len)
        embedded = self.embedding(tgt_input_idxs) #(batch,tgt_len,embed_dim)

        outputs=[]
        for t in range(embedded.size(1)): # step through target sequence
            input_t = embedded[:,t,:].unsqueeze(1) #(batch,1,embed_dim)

            # Attention context
            context, attn_weights = self.attention(hidden, encoder_outputs) # context:(batch,1,enc_dim)
            ctx_dec = self.ctx_proj(context) #(batch,1,hidden_dim)

            #Combine context with current input
            rnn_input = torch.cat((input_t, ctx_dec),dim=2) #(batch,1,embed_dim+hidden_dim)

            output, hidden = self.gru(rnn_input, hidden) # output:(batch,1,hidden_dim)

            # final output layer
            output_combined = torch.cat((output, ctx_dec),dim=2) #(batch,1,hidden_dim*2)
            prediction = self.fc_out(output_combined) #(batch,1,output_vocab_size)

            outputs.append(prediction)

        outputs = torch.cat(outputs,dim=1) #(batch,tgt_len,output_vocab_size)
        return outputs
```

```
    def step(self,input_token_idxs, hidden, encoder_outputs):
        # input_token_idxs:(batch,1)
        embedded = self.embedding(input_token_idxs) #(batch,1,embed_dim)

        context, attn_weights = self.attention(hidden, encoder_outputs) #(batch,1,enc_dim)
        ctx_dec = self.ctx_proj(context) #(batch,1,hidden_dim)

        rnn_input = torch.cat((embedded, ctx_dec),dim=2) #(batch,1,embed_dim+hidden_dim)

        output, hidden = self.gru(rnn_input, hidden) #(batch,1,hidden_dim)

        output_combined = torch.cat((output, ctx_dec),dim=2) #(batch,1,hidden_dim*2)
        prediction = self.fc_out(output_combined) #(batch,1,output_vocab_size)

        return prediction, hidden, attn_weights


class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src_idxs, tgt_input_idxs):
        encoder_outputs, hidden = self.encoder(src_idxs)
        outputs = self.decoder(tgt_input_idxs, hidden, encoder_outputs)
        return outputs
```

Config and Training Setup

In [23]:
```
# Hyperparameters
EMBED_DIM = 256 # size of word embeddings
HIDDEN_DIM = 256 # size of GRU hidden state
BATCH_SIZE = 64 # sentence per batch
EPOCHS = 15 # training passes over dataset
LR = 0.001 # learning rate
```

```
PAD_SRC = src_vocab["<pad>"]
PAD_TGT = tgt_vocab["<pad>"]
```

In [24]:
```
# Model
encoder = Encoder(input_vocab_size=len(src_vocab), embed_dim=EMBED_DIM, hidden_dim=HIDDEN_DIM)
decoder = Decoder(output_vocab_size=len(tgt_vocab),
                  embed_dim=EMBED_DIM,
                  hidden_dim=HIDDEN_DIM,
                  enc_dim=HIDDEN_DIM*2)
model = Seq2Seq(encoder, decoder, DEVICE).to(DEVICE)
```

In [25]:
```
# loss function (ignores the padding tokens in the targets)
criterion = nn.CrossEntropyLoss(ignore_index=PAD_TGT)
```

In [26]:
```
# label smoothing
criterion = nn.CrossEntropyLoss(ignore_index=PAD_TGT, label_smoothing=0.1)
```

In [27]:
```
# Adam optimizaer
optimizer = torch.optim.Adam(model.parameters(),lr=LR)
```

In [28]:
```
print(model)
```

```
Seq2Seq(
  (encoder): Encoder(
    (embedding): Embedding(46045, 256)
    (gru): GRU(256, 256, batch_first=True, bidirectional=True)
    (fc): Linear(in_features=512, out_features=256, bias=True)
  )
  (decoder): Decoder(
    (embedding): Embedding(25767, 256)
    (attention): LuongAttention(
      (attn): Linear(in_features=512, out_features=256, bias=True)
    )
    (ctx_proj): Linear(in_features=512, out_features=256, bias=True)
    (gru): GRU(512, 256, batch_first=True)
    (fc_out): Linear(in_features=512, out_features=25767, bias=True)
  )
)
```

Training and Validation Loops

In [29]:
```
def masked_accuracy(logits, targets, pad_idx):
    # logits: (B,T,V), targets: (B,T)
    with torch.no_grad():
        preds = logits.argmax(-1)
        mask = targets.ne(pad_idx)
        correct = (preds.eq(targets) & mask).sum().item()
        total = mask.sum().item()
        return correct/max(total, 1)
```

In [30]:
```
def run_epoch(loader, train=True):
    model.train() if train else model.eval()
    total_loss, total_acc, steps=0.0,0.0,0
    for src, tgt_in, tgt_out in tqdm(loader, desc="Train" if train else "Val", leave=False):
        src = src.to(DEVICE)
        tgt_in = tgt_in.to(DEVICE)
        tgt_out = tgt_out.to(DEVICE)

        if train:
            optimizer.zero_grad()

        # Forward
        logits =model(src, tgt_in) #(B,T,V)
        B, T, V = logits.shape
        loss = criterion(logits.view(B*T,V),tgt_out.view(B*T))

        if train:
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(),1.0)
            optimizer.step()

        acc = masked_accuracy(logits,tgt_out,PAD_TGT)
        total_loss+=loss.item()
        total_acc+=acc
        steps+=1
```

```
    return total_loss/steps, total_acc/steps
```

In [31]:
```
patience = 3
wait = 0
best_val = float("inf")
best_state = None

for epoch in range(1, EPOCHS+1):
    train_loss, train_acc = run_epoch(train_loader, train=True)
    val_loss, val_acc = run_epoch(val_loader, train=False)

    if val_loss<best_val:
        best_val=val_loss
        wait=0
        best_state ={k: v.detach().cpu().clone() for k, v in model.state_dict().items()}
    else:
        wait +=1
        if wait >= patience:
            print(f"Early stopping at epoch {epoch}")
            break

    print(f"Epoch {epoch:02d} | "
          f"train_loss {train_loss:.4f} acc {train_acc:.3f} | "
          f"val_loss {val_loss:.4f} acc {val_acc:.3f}")

# restore best weights
if best_state is not None:
    model.load_state_dict(best_state)
```

```
Epoch 01 | train_loss 4.1185 acc 0.511 | val_loss 4.7343 acc 0.408

Epoch 02 | train_loss 2.8188 acc 0.692 | val_loss 4.5096 acc 0.442

Epoch 03 | train_loss 2.3409 acc 0.776 | val_loss 4.4927 acc 0.452

Epoch 04 | train_loss 2.0795 acc 0.836 | val_loss 4.5228 acc 0.456
```

```
Epoch 05 | train_loss 1.9382 acc 0.873 | val_loss 4.5848 acc 0.454

Early stopping at epoch 6
```

In [32]:
```
torch.save(model.state_dict(), "nmt_luong_attn.pt")
print("Saved to nmt_luong_attn.pt")
```

```
Saved to nmt_luong_attn.pt
```

Inference (Greedy Decoding with Attention)

In [33]:
```
#helper to build a source tensor from Spanish text
def prepare_src_sentence_es(sentence, src_vocab, device):
    # no <start>/<end> on source
    tokens = sentence.lower().strip().split()
    idxs = [src_vocab.get(tok, src_vocab["<unk>"]) for tok in tokens]
    return torch.tensor(idxs,dtype=torch.long).unsqueeze(0).to(device) # (1,S)

# index to word for target vocab
tgt_idx2word = {idx: w for w, idx in tgt_vocab.items()}

# greedy decode
def translate_es_to_en(model, sentence_es, src_vocab, tgt_vocab, device, max_len= 40):
    model.eval()
    with torch.no_grad():
        src = prepare_src_sentence_es(sentence_es, src_vocab, DEVICE)
        encoder_outputs, hidden=model.encoder(src)

        start_id=tgt_vocab["<start>"]
        end_id=tgt_vocab["<end>"]
        next_token = torch.tensor([[start_id]],dtype=torch.long,device=DEVICE)

        decoded_ids = []

        for _ in range(max_len):
            logits, hidden, attn = model.decoder.step(next_token, hidden, encoder_outputs)
            next_id = int(logits[:, -1, :].argmax(dim=-1).item())
```

```
            if next_id == end_id:
                break
            decoded_ids.append(next_id)
            next_token = torch.tensor([[next_id]],dtype=torch.long,device=DEVICE)

    words = [tgt_idx2word[i] for i in decoded_ids if tgt_idx2word[i] not in ("<start>", "<end>")]
    return " ".join(words)
```

In [34]:

```
for _ in range(10):
    j = random.randint(0, len(spanish_sentences) - 1)
    src_es = spanish_sentences[j]# Spanish source
    ref_en = english_sentences[j]# English reference
    pred_en = translate_es_to_en(model, src_es, src_vocab, tgt_vocab, DEVICE)

    print(f"ES: {src_es}")
    print(f"PRED: {pred_en}")
    print(f"REF: {ref_en}")
    print("-" * 40)
```

```
ES: ¿Alguna vez comerías una cucaracha?
PRED: have you ever seen movies a day?
REF: <start> Would you ever eat a cockroach? <end>
----------------------------------------
ES: Encontramos el arma homicida.
PRED: the found gun found the weapon.
REF: <start> We found the murder weapon. <end>
----------------------------------------
ES: Estoy seguro de que hay una mejor manera de hacer eso.
PRED: i'm sure the one is doing that anymore.
REF: <start> I'm sure there's a better way to do that. <end>
----------------------------------------
ES: ¿Quién de ustedes no estuvo en el autobús?
PRED: which wasn't in tom's wife?
REF: <start> Which one of you wasn't on the bus? <end>
----------------------------------------
ES: Enseguida almuerzo.
PRED: i'll bring lunch soon.
REF: <start> I'll eat lunch soon. <end>
----------------------------------------
ES: Ese es un árbol limonero.
PRED: this is lemon tree.
REF: <start> That's a lemon tree. <end>
----------------------------------------
ES: Nos olvidamos.
PRED: we kissed each other out.
REF: <start> We forgot. <end>
----------------------------------------
ES: Si no tengo hambre.
PRED: if i don't have any hungry, if i were hungry.
REF: <start> I'm really not hungry. <end>
----------------------------------------
ES: Puede que él haya perdido el último tren.
PRED: he may have missed the train.
REF: <start> He may have missed the last train. <end>
----------------------------------------
ES: ¿Qué vas a hacer con mi foto?
PRED: what do you have? to do with you?
REF: <start> What are you going to do with my picture? <end>
----------------------------------------
```

In [ ]: