hw5q4

April 4, 2025

```python
[35]: import torch
      import torch.nn as nn
      import torch.optim as optim
      from torch.utils.data import DataLoader, Dataset

      # Device configuration
      device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
      print(f"Using device: {device}")
```

```
Using device: cuda
```

```python
[36]: # English to French translation dataset
      english_to_french = [
          ("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"),
        ("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")
]
```

[37]:
```python
# 1. Reverse the order of sentences in your dataset
reversed_dataset = [(target_sentence, input_sentence) for input_sentence,
 ↪target_sentence in english_to_french]

# Special tokens for the start and end of sequences
SOS_token = 0  # Start Of Sequence Token
EOS_token = 1  # End Of Sequence Token

# 2. Modify the word-to-index mapping
reversed_word_to_index = {"SOS": SOS_token, "EOS": EOS_token}
for pair in reversed_dataset:
    for word in pair[0].split() + pair[1].split():
        if word not in reversed_word_to_index:
            reversed_word_to_index[word] = len(reversed_word_to_index)

# 3. Adjust the TranslationDataset class to generate input and target tensors
 ↪in the reversed order
class ReversedTranslationDataset(Dataset):
    def __init__(self, dataset, word_to_index):
        self.dataset = dataset
        self.word_to_index = word_to_index

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

    def __getitem__(self, idx):
        target_sentence, input_sentence = self.dataset[idx]
        input_indices = [self.word_to_index[word] for word in input_sentence.
 ↪split()] + [EOS_token]
        target_indices = [self.word_to_index[word] for word in target_sentence.
 ↪split()] + [EOS_token]
```

```python
        return torch.tensor(input_indices, dtype=torch.long).to(device), torch.
 ↪tensor(target_indices, dtype=torch.long).to(device)

# Creating a DataLoader to batch and shuffle the reversed dataset
reversed_translation_dataset = ReversedTranslationDataset(reversed_dataset,␣
 ↪reversed_word_to_index)

# use batch size of 1
reversed_dataloader = DataLoader(reversed_translation_dataset, batch_size=1,␣
 ↪shuffle=True)

class Transformer(nn.Module):
    def __init__(self, input_vocab_size, target_vocab_size, hidden_size,␣
 ↪num_layers=2, num_heads=8, dropout=0.1):
        super(Transformer, self).__init__()
        self.embedding_input = nn.Embedding(input_vocab_size, hidden_size)
        self.embedding_target = nn.Embedding(target_vocab_size, hidden_size)
        self.transformer = nn.Transformer(
            d_model=hidden_size,
            nhead=num_heads,
            num_encoder_layers=num_layers,
            num_decoder_layers=num_layers,
            dim_feedforward=hidden_size * 4,
            dropout=dropout
        )
        self.fc = nn.Linear(hidden_size, target_vocab_size)

    def forward(self, input, target):
        input_embedded = self.embedding_input(input)
        target_embedded = self.embedding_target(target)

        input_pad_mask = self.generate_padding_mask(input)
        target_pad_mask = self.generate_padding_mask(target)
        target_subsequent_mask = self.generate_subsequent_mask(target)

        input_embedded = input_embedded.permute(1, 0, 2)
        target_embedded = target_embedded.permute(1, 0, 2)

        output = self.transformer(
            input_embedded,
            target_embedded,
            src_key_padding_mask=input_pad_mask,
            tgt_key_padding_mask=target_pad_mask,
            memory_key_padding_mask=input_pad_mask,
            tgt_mask=target_subsequent_mask
        )
```

```python
        output = self.fc(output)
        return output.permute(1, 0, 2)

    def generate_padding_mask(self, sequence):
        # Use EOS token for padding mask
        mask = (sequence == EOS_token)
        return mask.to(device)

    def generate_subsequent_mask(self, sequence):
        mask = (torch.triu(torch.ones(sequence.size(1), sequence.size(1))) ==
↪1).transpose(0, 1)
        mask = mask.float().masked_fill(mask == 0, float('-inf')).
↪masked_fill(mask == 1, float(0.0))
        return mask.to(device)

# Assuming all words in the dataset + 'SOS' and 'EOS' tokens are included in
↪word_to_index
input_size = len(reversed_word_to_index)
hidden_size = 64
output_size = len(reversed_word_to_index)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = Transformer(input_size, output_size, hidden_size).to(device)

# Set the learning rate for optimization
learning_rate = 0.0005

# Define the loss function and optimizer - use EOS token for ignore_index
criterion = nn.CrossEntropyLoss(ignore_index=EOS_token)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Set number of epochs for training
n_epochs = 20

# Training loop
for epoch in range(n_epochs):
    total_loss = 0
    total_correct = 0
    total_examples = 0

    model.train()

    for input_tensor, target_tensor in reversed_dataloader:
        input_tensor = input_tensor.to(device)
        target_tensor = target_tensor.to(device)
```

```python
        optimizer.zero_grad()

        output = model(input_tensor, target_tensor[:, :-1])  # Exclude EOS
↪token from target
        output_dim = output.shape[-1]

        output = output.contiguous().view(-1, output_dim)
        target_tensor = target_tensor[:, 1:].contiguous().view(-1)  # Exclude
↪SOS token from target

        loss = criterion(output, target_tensor)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

        _, predicted = torch.max(output, 1)
        correct = (predicted == target_tensor).sum().item()
        total_correct += correct
        total_examples += target_tensor.size(0)

    avg_loss = total_loss / len(reversed_dataloader)
    training_accuracy = total_correct / total_examples

    print(f"Epoch [{epoch+1}/{n_epochs}], Loss: {avg_loss:.4f}, Training
↪Accuracy: {training_accuracy:.4f}")
```

```
Epoch [1/20], Loss: 5.9875, Training Accuracy: 0.0187
Epoch [2/20], Loss: 5.1621, Training Accuracy: 0.0560
Epoch [3/20], Loss: 4.6023, Training Accuracy: 0.1120
Epoch [4/20], Loss: 4.1411, Training Accuracy: 0.1627
Epoch [5/20], Loss: 3.6967, Training Accuracy: 0.2640
Epoch [6/20], Loss: 3.3140, Training Accuracy: 0.3280
Epoch [7/20], Loss: 2.9373, Training Accuracy: 0.3867
Epoch [8/20], Loss: 2.6070, Training Accuracy: 0.4373
Epoch [9/20], Loss: 2.3243, Training Accuracy: 0.4933
Epoch [10/20], Loss: 2.0018, Training Accuracy: 0.5653
Epoch [11/20], Loss: 1.7610, Training Accuracy: 0.6080
Epoch [12/20], Loss: 1.4833, Training Accuracy: 0.6533
Epoch [13/20], Loss: 1.2405, Training Accuracy: 0.6933
Epoch [14/20], Loss: 1.0578, Training Accuracy: 0.7200
Epoch [15/20], Loss: 0.9185, Training Accuracy: 0.7413
Epoch [16/20], Loss: 0.7611, Training Accuracy: 0.7653
Epoch [17/20], Loss: 0.6126, Training Accuracy: 0.7840
Epoch [18/20], Loss: 0.5211, Training Accuracy: 0.7733
Epoch [19/20], Loss: 0.4451, Training Accuracy: 0.7867
Epoch [20/20], Loss: 0.3724, Training Accuracy: 0.7867
```

```
[38]: # Define the evaluation function outside the training loop
      def evaluate_model(model, dataloader, criterion):
          total_loss = 0
          total_correct = 0
          total_examples = 0

          model.eval()  # Set the model to evaluation mode

          with torch.no_grad():  # No gradient calculation during evaluation
              for input_tensor, target_tensor in dataloader:
                  input_tensor = input_tensor.to(device)
                  target_tensor = target_tensor.to(device)

                  output = model(input_tensor, target_tensor[:, :-1])  # Exclude EOS
      ↪token from target
                  output_dim = output.shape[-1]

                  output = output.contiguous().view(-1, output_dim)
                  target_tensor = target_tensor[:, 1:].contiguous().view(-1)  #
      ↪Exclude SOS token from target

                  loss = criterion(output, target_tensor)

                  total_loss += loss.item()

                  _, predicted = torch.max(output, 1)
                  correct = (predicted == target_tensor).sum().item()
                  total_correct += correct
                  total_examples += target_tensor.size(0)  # Use target tensor size
      ↪for total examples

          avg_loss = total_loss / len(dataloader)
          accuracy = total_correct / total_examples  # Calculate accuracy

          return avg_loss, accuracy

      eval_loss, eval_accuracy = evaluate_model(model, reversed_dataloader, criterion)
      print(f"Evaluation Loss: {eval_loss:.4f}, Evaluation Accuracy: {eval_accuracy:.
      ↪4f}")
```

Evaluation Loss: 0.1495, Evaluation Accuracy: 0.7920

[ ]: