Name: **B. SAI CHARAN**

Roll No: **2203A51L72**

Batch No: **21CSBTB12**

ASSIGNMENT – 7

Question:

Implement and train Seq2Seq models for machine translation.

Answer:

Here's an implementation of Sequence-to-Sequence (Seq2Seq) models for machine translation using PyTorch. This example will focus on training a model to translate simple sentences from English to French. The Seq2Seq model consists of an **encoder** (which processes the input sequence) and a **decoder** (which generates the output sequence).

Steps to Implement and Train the Seq2Seq Model

1. **Data Preparation**: Prepare paired sentence data for translation from English to French.
2. **Encoder and Decoder Models**: Define separate models for encoding the input sentence and decoding to generate the translation.
3. **Training Loop**: Train the Seq2Seq model on the dataset and evaluate the output.

Explanation

1. **Vocabulary Encoding**:
    - We define eng_vocab and fre_vocab to map each character to an integer. This helps the model work with numeric data rather than text.
2. **Encoder Model**:
    - The Encoder takes a sequence of characters in the source language and outputs a hidden state representation using a GRU layer.
    - **Embedding Layer**: Maps input characters to dense vectors, enabling better generalization.

    - **GRU Layer**: Processes embedded input sequence and generates a hidden state that summarizes the sentence.

3. **Decoder Model**:
    - The Decoder takes the hidden state from the Encoder and generates the translated sequence step-by-step.
    - **Embedding Layer**: Transforms each input character index into a dense vector.
    - **GRU Layer**: Generates a hidden state for each character in the output sequence.
    - **Fully Connected Layer**: Maps hidden state to the output vocabulary, predicting the next character.
    - **LogSoftmax**: Produces log probabilities for each character in the vocabulary.
4. **Seq2Seq Model**:
    - Combines the Encoder and Decoder to build the complete Seq2Seq model.
    - **Teacher Forcing**: A technique to speed up training by feeding the actual target as the next input during training rather than the predicted output.
5. **Training Loop**:
    - For each epoch, the model is trained on all sentence pairs in pairs.
    - **Loss Calculation**: We accumulate the loss for each character in the target sequence to compute the overall sentence loss.
    - **Backpropagation**: The loss is backpropagated to update model weights.
6. **Translation**:
    - To translate a new sentence, we pass it through the encoder, then feed the decoder step-by-step.
    - **Decoder Prediction**: The decoder produces one character at a time, and the output is appended to the translated sentence until reaching an end token or maximum length.

This code demonstrates a basic Seq2Seq model. For better translation performance, more data, a larger vocabulary, and attention mechanisms (like Bahdanau or Luong attention) could be added.

Code Implementation:

```
import torch
import torch.nn as nn
import torch.optim as optim
import random

# Sample data - English to French pairs
pairs = [
    ("i am a student", "je suis un étudiant"),
    ("she is a teacher", "elle est une enseignante"),
    ("he is a doctor", "il est un médecin"),
    ("they are engineers", "ils sont des ingénieurs"),
]

# Vocabulary dictionaries
eng_vocab = sorted(set(" ".join([pair[0] for pair in pairs])))
fre_vocab = sorted(set(" ".join([pair[1] for pair in pairs])))

eng_vocab = {word: i for i, word in enumerate(eng_vocab)}
fre_vocab = {word: i for i, word in enumerate(fre_vocab)}
eng_vocab_size = len(eng_vocab)
fre_vocab_size = len(fre_vocab)

# Encode sentences as index sequences
def encode_sentence(sentence, vocab):
    return [vocab[char] for char in sentence]

# Hyperparameters
hidden_size = 256
learning_rate = 0.01
num_epochs = 1000

# Encoder
class Encoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Encoder, self).__init__()
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size, batch_first=True)
```

Seq2Seq:

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

    def forward(self, src, trg, teacher_forcing_ratio=0.5):
        encoder_hidden = self.encoder(src)
        decoder_input = trg[:, 0]  # Start token for decoding
        decoder_hidden = encoder_hidden
        outputs = torch.zeros(trg.size(0), trg.size(1), fre_vocab_size)

        for t in range(1, trg.size(1)):
            output, decoder_hidden = self.decoder(decoder_input, decoder_hidden)
            outputs[:, t, :] = output
            top1 = output.argmax(1)
            decoder_input = trg[:, t] if random.random() < teacher_forcing_ratio else top1
        return outputs

# Instantiate models
encoder = Encoder(eng_vocab_size, hidden_size)
decoder = Decoder(hidden_size, fre_vocab_size)
model = Seq2Seq(encoder, decoder)

# Loss and optimizer
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training function
def train(model, pairs, num_epochs=1000):
    for epoch in range(num_epochs):
        total_loss = 0
        for pair in pairs:
            # Prepare data
            src_sentence, trg_sentence = pair
            src_indices = torch.tensor([encode_sentence(src_sentence, eng_vocab)], dtype=torch.long)
            trg_indices = torch.tensor([encode_sentence(trg_sentence, fre_vocab)], dtype=torch.long)
```

Training the Model:

```
# Train the model
train(model, pairs, num_epochs)

# Translation function
def translate(model, sentence):
    src_indices = torch.tensor([encode_sentence(sentence, eng_vocab)], dtype=torch.long)
    encoder_hidden = model.encoder(src_indices)
    decoder_input = torch.tensor([fre_vocab[" "]], dtype=torch.long)  # Start token
    decoder_hidden = encoder_hidden
    translated_sentence = ""

    for _ in range(20):
        output, decoder_hidden = model.decoder(decoder_input, decoder_hidden)
        top1 = output.argmax(1).item()
        if top1 == fre_vocab.get(" "):
            break
        translated_sentence += idx_to_char_fre[top1]
        decoder_input = torch.tensor([top1], dtype=torch.long)

    return translated_sentence

# Translate a sentence
print("Translation:", translate(model, "i am a student"))
```

Output:

```
••• Epoch [0/1000], Loss: 58.3907
    Epoch [100/1000], Loss: 0.0165
    Epoch [200/1000], Loss: 0.0054
    Epoch [300/1000], Loss: 0.0027
    Epoch [400/1000], Loss: 0.0016
```