Transformer Sequence-to-Sequence Model

Code Explanation

1. Dependencies and Setup

The code begins by installing and importing necessary libraries like `torch`, `torchtext`, and `spacy`. It also downloads English and German language models for tokenization.

2. Tokenization and Vocabulary

Spacy tokenizers are used for both English and German. A vocabulary is built from the training data using `torchtext.vocab.build_vocab_from_iterator`.

Special tokens `<unk>`, `<pad>`, `<bos>`, and `<eos>` are defined and added to the vocabulary. `set_default_index(UNK_IDX)` ensures unknown tokens are handled correctly.

3. Data Pipeline

A `collate_fn` function prepares batches:

- Tokenizes sentences
- Converts tokens to IDs
- Adds BOS/EOS tokens
- Pads sequences for uniform batch shape

Data is loaded using `torch.utils.data.DataLoader`.

4. Model Components

- **TokenEmbedding**: Converts token indices to dense vectors.
- **PositionalEncoding**: Adds position info using sinusoidal functions.
- **Seq2SeqTransformer**: Wraps PyTorch's `nn.Transformer`.
  - Applies embedding + positional encoding
  - Passes through Transformer encoder/decoder
  - Uses a final linear layer to predict token probabilities

5. Masking

- Target masks (causal masks) prevent attention to future tokens.
- Source/target padding masks prevent attention to padded elements.

6. Training and Evaluation

`train_epoch` runs a full pass over the training data with loss computation and backpropagation.
`evaluate` runs inference on validation data without updating weights.
Loss is computed using `CrossEntropyLoss`, ignoring padding tokens.

7. Greedy Decoding for Inference

`greedy_decode` generates the target sequence one token at a time by always picking the most likely next token.
Steps:

1. Encode source
2. Initialize decoder with BOS token
3. At each step, decode and pick the token with the highest probability
4. Stop at EOS or max length

8. Translation Utility

`translate`:

- Preprocesses input sentence (tokenizes, numericalizes, adds BOS/EOS)
- Generates output sequence using `greedy_decode`
- Converts output tokens to readable text

9. Training Loop

The training loop:

- Defines `NUM_EPOCHS`
- Trains model using `train_epoch`
- Evaluates each epoch using `evaluate`
- Uses Adam optimizer with a fixed learning rate

Summary

This code provides a full implementation of a Transformer-based machine translation system with:

- Spacy tokenization
- Vocabulary building
- PyTorch `nn.Transformer`
- Masked attention
- Training and inference logic
- Translation utility for real input