Line-by-Line Explanation of Transformer Seq2Seq Code

**Overview**  
This code implements a sequence-to-sequence Transformer model using PyTorch and TorchText for German-to-English translation. It covers data preprocessing (tokenization, vocabulary creation, and batching), model construction (with positional encodings and custom token embeddings), training/evaluation loops, and a greedy decode method for inference.  
  
**1. Data Preparation and Preprocessing**

* **Tokenization and Vocabulary:**
  + Uses SpaCy to create tokenizers for German and English
  + Iterates through the Multi30k dataset to tokenize sentences and build vocabularies with special tokens (<unk>, <pad>, <bos>, <eos>).
  + Sets the unknown token index to gracefully handle out-of-vocabulary tokens.
* **Text Transformation:**
  + The sequential\_transforms helper chains together tokenization, numerical conversion (using the vocabulary), and tensor conversion with added beginning-of-sentence (BOS) and end-of-sentence (EOS) markers.
  + The collate\_fn function pads sequences to create uniform batches, ensuring proper tensor dimensions for model input.

**2. Model Architecture**

* **Positional Encoding:**
  + Implements positional encodings to inject sequence order into token embeddings.
  + Uses sinusoidal functions over a defined maximum length.
* **Token Embedding:**
  + A simple module that transforms token indices into dense embeddings scaled by the square root of the embedding size.
* **Seq2Seq Transformer Model:**
  + The model comprises a Transformer with configurable numbers of encoder and decoder layers.
  + It includes separate token embedding layers for the source and target languages, as well as positional encoding applied before passing data to the Transformer.
  + The generator (final linear layer) projects the Transformer output to the target vocabulary size for token prediction.
* **Masking Functions:**
  + generate\_square\_subsequent\_mask: Creates a triangular mask to prevent the decoder from “peeking” at future tokens.
  + create\_mask: Constructs masks for source/target padding and subsequent masking, ensuring proper attention mechanics during training.

**3. Training and Evaluation**

* **Training Loop:**
  + Processes the Multi30k training dataset batch-by-batch.
  + For each batch, source and target sequences are sent to the device, and the target sequence is shifted to create input and output pairs.
  + Loss is computed via CrossEntropyLoss (ignoring padding tokens) and optimized using Adam with a custom learning rate and hyperparameters.
* **Evaluation Loop:**
  + Similar to training, but with the model set to evaluation mode.
  + Processes the validation set to compute the average loss (and by extension, perplexity) over batches.
* **Greedy Decoding & Translation:**
  + Implements greedy search to generate a target sentence given a source sentence.
  + The greedy\_decode function iteratively predicts the next token until an EOS token is generated or maximum length is reached.
  + The translate function transforms an input string into a tensor, applies greedy decoding, and converts the output indices back to human-readable text (after stripping BOS/EOS tokens).

**4. Hyperparameters & Setup**

* The model uses 3 encoder and 3 decoder layers with an embedding size of 512, 8 attention heads, and a feed-forward network dimension of 512.
* Data batching, tokenization, and learning rate settings are defined explicitly.
* Model param

**Conclusion**  
This implementation provides a complete pipeline—from data ingestion and preprocessing to model training, evaluation, and inference—using state-of-the-art Transformer architectures for machine translation. The documentation covers the core functionalities, making it easier to understand, extend, or integrate into larger projects.