1. What are the pros and cons of using a stateful RNN versus a stateless RNN?

**Stateful RNN:**

**Pros:**

**Maintains state information across sequences, suitable for tasks with long-term dependencies.**

**Can learn and generate sequences with coherent context.**

**Cons:**

**Can be computationally expensive and memory-intensive due to the need to store and manage state information.**

**Less parallelization during training, potentially slowing down training time.**

**Stateless RNN:**

**Pros:**

**Simpler to implement and train since each input sequence is treated independently.**

**Easier to parallelize, leading to faster training on hardware like GPUs.**

**Cons:**

**May struggle to capture long-term dependencies in sequences.**

**Less suitable for tasks where context and state information across sequences are essential.**

2. Why do people use Encoder–Decoder RNNs rather than plain sequence-to-sequence RNNs

for automatic translation?

**Encoder-Decoder RNNs are preferred for automatic translation and similar tasks because:**

**They can handle variable-length input and output sequences, making them suitable for translation where sentences can vary in length.**

**The encoder processes the input sequence and generates a fixed-size context vector, which the decoder uses to generate the output sequence. This context vector helps capture semantic information from the input.**

**Encoder-Decoder models can align and translate words or subunits of different languages effectively.**

3. How can you deal with variable-length input sequences? What about variable-length output

sequences?

**Variable-Length Input Sequences: You can pad input sequences to a maximum length or use techniques like masking to handle variable-length inputs. Padding introduces extra values, and masking helps the model ignore padded elements.**

**Variable-Length Output Sequences: For output sequences, you can use special tokens to indicate the end of a sequence or employ techniques like beam search to generate sequences of variable lengths.**

4. What is beam search and why would you use it? What tool can you use to implement it?

**Beam search is a technique used in sequence generation tasks, such as machine translation or text generation. It explores multiple possible sequences by maintaining a "beam" of the top-k most likely candidates at each decoding step.**

**Beam search helps improve the quality of generated sequences by considering a broader range of possibilities, reducing the risk of getting stuck in local optima.**

**Tools like TensorFlow's tf.sequence\_beam\_search or external libraries can be used to implement beam search during sequence generation.**

5. What is an attention mechanism? How does it help?

**An attention mechanism is a component used in neural networks to selectively focus on different parts of the input sequence when generating each element of the output sequence.**

**It helps models attend to relevant parts of the input sequence, improving their ability to capture long-range dependencies and generate coherent and contextually accurate output sequences.**

**Attention mechanisms have significantly improved the performance of sequence-to-sequence tasks like machine translation.**

6. What is the most important layer in the Transformer architecture? What is its purpose?

**The most critical layer in the Transformer architecture is the "Multi-Head Self-Attention" layer.**

**Its purpose is to capture relationships and dependencies between different positions within the input sequence, allowing the model to weigh the importance of each element based on context.**

7. When would you need to use sampled softmax?

**Sampled softmax is used when dealing with large output vocabulary in tasks like language modeling or text generation.**

**Instead of calculating the full softmax over all possible output tokens, sampled softmax randomly selects a subset of tokens to compute the softmax over, making training more efficient.**

**Sampled softmax is used to approximate the full softmax, and it helps manage computational complexity in large-scale language models.**