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

**Ans : Stateless RNNs can only capture patterns whose length is less than, or equal to, the size of the windows the RNN is trained on. Conversely, stateful RNNs can capture longer-term patterns. However, implementing a stateful RNN is much harder— especially preparing the dataset properly. Moreover, stateful RNNs do not always work better, in part because consecutive batches are not independent and identically distributed (IID). Gradient Descent is not fond of non-IID datasets.**

1. Why do people use Encoder–Decoder RNNs rather than plain sequence-to-sequence RNNs for automatic translation?

**Ans : In general, if you translate a sentence one word at a time, the result will be terrible. For example, the French sentence “Je vous en prie” means “You are welcome,” but if you translate it one word at a time, you get “I you in pray.” Huh? It is much better to read the whole sentence first and then translate it. A plain sequence-to-sequence RNN would start translating a sentence immediately after reading the first word, while an Encoder– Decoder RNN will first read the whole sentence and then translate it. That said, one could imagine a plain sequence-to-sequence RNN that would output silence whenever it is unsure about what to say next (just like human translators do when they must translate a live broadcast).**

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

**Ans : Variable-length input sequences can be handled by padding the shorter sequences so that all sequences in a batch have the same length, and using masking to ensure the RNN ignores the padding token. For better performance, you may also want to create batches containing sequences of similar sizes. Ragged tensors can hold sequences of variable lengths, and tf.keras will likely support them eventually, which will greatly simplify handling variablelength input sequences (at the time of this writing, it is not the case yet). Regarding variable-length output sequences, if the length of the output sequence is known in advance (e.g., if you know that it is the same as the input sequence), then you just need to configure the loss function so that it ignores tokens that come after the end of the sequence. Similarly, the code that will use the model should ignore tokens beyond the end of the sequence. But generally the length of the output sequence is not known ahead of time, so the solution is to train the model so that it outputs an end-of-sequence token at the end of each sequence.**

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

**Ans : Beam search is a technique used to improve the performance of a trained Encoder–Decoder model, for example in a neural machine translation system. The algorithm keeps track of a short list of the k most promising output sentences (say, the top three), and at each decoder step it tries to extend them by one word; then it keeps only the k most likely sentences. The parameter k is called the beam width: the larger it is, the more CPU and RAM will be used, but also the more accurate the system will be. Instead of greedily choosing the most likely next word at each step to extend a single sentence, this technique allows the system to explore several promising sentences simultaneously. Moreover, this technique lends itself well to parallelization. You can implement beam search fairly easily using TensorFlow Addons.**

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

**Ans : Beam search is a technique used to improve the performance of a trained Encoder–Decoder model, for example in a neural machine translation system. The algorithm keeps track of a short list of the k most promising output sentences (say, the top three), and at each decoder step it tries to extend them by one word; then it keeps only the k most likely sentences. The parameter k is called the beam width: the larger it is, the more CPU and RAM will be used, but also the more accurate the system will be. Instead of greedily choosing the most likely next word at each step to extend a single sentence, this technique allows the system to explore several promising sentences simultaneously. Moreover, this technique lends itself well to parallelization. You can implement beam search fairly easily using TensorFlow Addons.**

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

**Ans : Beam search is a technique used to improve the performance of a trained Encoder–Decoder model, for example in a neural machine translation system. The algorithm keeps track of a short list of the k most promising output sentences (say, the top three), and at each decoder step it tries to extend them by one word; then it keeps only the k most likely sentences. The parameter k is called the beam width: the larger it is, the more CPU and RAM will be used, but also the more accurate the system will be. Instead of greedily choosing the most likely next word at each step to extend a single sentence, this technique allows the system to explore several promising sentences simultaneously. Moreover, this technique lends itself well to parallelization. You can implement beam search fairly easily using TensorFlow Addons.**

1. When would you need to use sampled softmax?

**Ans : Sampled softmax is used when training a classification model when there are many classes (e.g., thousands). It computes an approximation of the cross-entropy loss based on the logit predicted by the model for the correct class, and the predicted logits for a sample of incorrect words. This speeds up training considerably compared to computing the softmax over all logits and then estimating the cross-entropy loss. After training, the model can be used normally, using the regular softmax function to compute all the class probabilities based on all the logits.**