1. Can you think of a few applications for a sequence-to-sequence RNN? What about a sequence-to-vector RNN? And a vector-to-sequence RNN?
2. Why do people use encoder–decoder RNNs rather than plain sequence-to-sequence RNNs for automatic translation?
3. How could you combine a convolutional neural network with an RNN to classify videos?
4. What are the advantages of building an RNN using dynamic\_rnn() rather than static\_rnn()?
5. How can you deal with variable-length input sequences? What about variable-length output sequences?
6. What is a common way to distribute training and execution of a deep RNN across multiple GPUs?

ANSWER:

1. Some applications for a sequence-to-sequence RNN include machine translation, speech recognition, and chatbots. A sequence-to-vector RNN could be used for sentiment analysis or text classification tasks where the goal is to classify the input sequence into a fixed number of categories. A vector-to-sequence RNN could be used for image captioning, where the input is an image vector and the output is a sequence of words describing the image.
2. Encoder-decoder RNNs are used for automatic translation because they can handle variable-length input and output sequences. The encoder RNN reads in the input sequence and produces a fixed-length vector that summarizes the information in the sequence. The decoder RNN then takes this vector as input and generates the output sequence. This approach allows the model to handle variable-length sequences and generate translations of different lengths.
3. A convolutional neural network (CNN) can be used to extract features from individual frames of a video, and an RNN can be used to model the temporal relationships between these frames. One approach is to use a CNN to extract features from each frame, and then use an RNN to model the sequence of features. Another approach is to use a 3D CNN to directly process the video frames and then use an RNN to model the sequence of features extracted by the CNN.
4. The dynamic\_rnn() function in TensorFlow is designed to handle variable-length input sequences. It automatically calculates the sequence length and uses it to dynamically unroll the RNN. This can be more memory-efficient than the static\_rnn() function, which requires the entire input sequence to be stored in memory before the RNN is unrolled.
5. To handle variable-length input sequences, padding can be added to the shorter sequences so that all input sequences have the same length. The padding can be masked so that it does not affect the output of the RNN. Variable-length output sequences can be handled using techniques such as beam search or teacher forcing.
6. One common way to distribute training and execution of a deep RNN across multiple GPUs is to use a technique called data parallelism. In this approach, each GPU processes a subset of the input data and the gradients are combined across all GPUs to update the model parameters. Another approach is model parallelism, where different parts of the RNN are processed on different GPUs. However, this approach can be more difficult to implement and can introduce communication overhead between the GPUs.