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?

Answers:

1.

a. Applications for sequence-to-sequence RNN:

- Machine translation
- Speech recognition
- Text summarization
- Conversational agents
- Image captioning

b. Applications for sequence-to-vector RNN:

- Sentiment analysis
- Document classification
- Stock price prediction
- Image and video classification

c. Applications for vector-to-sequence RNN:

- Text generation
- Music generation
- Video captioning

2. People use encoder-decoder RNNs rather than plain sequence-to-sequence RNNs for automatic translation because encoder-decoder models have the ability to capture both the context of the source sentence and generate a corresponding target sentence. This is achieved by first encoding the input sequence into a fixed-length vector, which is then used to initialize the decoder to generate the output sequence. This allows the model to handle variable-length input and output sequences.
3. A common way to combine a convolutional neural network (CNN) with an RNN for video classification is to use the CNN to extract features from each frame of the video

and then use an RNN to process the sequence of features. The output of the RNN can then be used for classification. Another approach is to use a 3D CNN that takes the entire video as input and outputs a feature map that is fed into an RNN for further processing.

4. The advantages of building an RNN using dynamic_rnn() rather than static_rnn() are:

- Dynamic_rnn() allows for variable-length input sequences, whereas static_rnn() requires fixed-length sequences.
- Dynamic_rnn() is more memory efficient as it only processes the part of the input sequence that is required, whereas static_rnn() processes the entire sequence even if some parts are not needed.
- Dynamic_rnn() is easier to use with batched sequences, whereas static_rnn() requires more manual handling of batch dimensions.

5. To deal with variable-length input sequences, padding can be used to make all input sequences the same length. Another approach is to use dynamic_rnn() or bucketing to handle variable-length sequences. To deal with variable-length output sequences, the output can be truncated to a maximum length or a special end-of-sequence token can be used to indicate the end of the sequence.
6. A common way to distribute training and execution of a deep RNN across multiple GPUs is to use data parallelism. In this approach, each GPU processes a different batch of data and the gradients are averaged across all GPUs to update the model parameters. Another approach is to use model parallelism, where different parts of the model are run on different GPUs. However, model parallelism is more complex and requires careful partitioning of the model.