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?

**Ans : Here are a few RNN applications:**

* **For a sequence-to-sequence RNN: predicting the weather (or any other time series), machine translation (using an encoder–decoder architecture), video captioning, speech to text, music generation (or other sequence generation), identifying the chords of a song.**
* **For a sequence-to-vector RNN: classifying music samples by music genre, ana‐ lyzing the sentiment of a book review, predicting what word an aphasic patient is thinking of based on readings from brain implants, predicting the probabil‐ ity that a user will want to watch a movie based on her watch history (this is one of many possible implementations of collaborative filtering).**
* **For a vector-to-sequence RNN: image captioning, creating a music playlist based on an embedding of the current artist, generating a melody based on a set of parameters, locating pedestrians in a picture (e.g., a video frame from a self-driving car’s camera).**

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 terri‐ ble. 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-tosequence 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 could you combine a convolutional neural network with an RNN to classify videos?

**Ans : To classify videos based on the visual content, one possible architecture could be to take (say) one frame per second, then run each frame through a convolutional neural network, feed the output of the CNN to a sequence-to-vector RNN, and finally run its output through a softmax layer, giving you all the class probabili‐ ties. For training you would just use cross entropy as the cost function. If you wanted to use the audio for classification as well, you could convert every second of audio to a spectrograph, feed this spectrograph to a CNN, and feed the output of this CNN to the RNN (along with the corresponding output of the other CNN).**

1. What are the advantages of building an RNN using dynamic\_rnn() rather than static\_rnn()?

**Ans : Building an RNN using dynamic\_rnn() rather than static\_rnn() offers several advantages:**

* **It is based on a while\_loop() operation that is able to swap the GPU’s memory to the CPU’s memory during backpropagation, avoiding out-of-memory errors.**
* **It is arguably easier to use, as it can directly take a single tensor as input and output (covering all time steps), rather than a list of tensors (one per time step). No need to stack, unstack, or transpose.**
* **It generates a smaller graph, easier to visualize in TensorBoard.**

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

**Ans : To handle variable length input sequences, the simplest option is to set the sequence\_length parameter when calling the static\_rnn() or dynamic\_rnn() functions. Another option is to pad the smaller inputs (e.g., with zeros) to make them the same size as the largest input (this may be faster than the first option if the input sequences all have very similar lengths). To handle variable-length out‐ put sequences, if you know in advance the length of each output sequence, you can use the sequence\_length parameter (for example, consider a sequence-tosequence RNN that labels every frame in a video with a violence score: the output sequence will be exactly the same length as the input sequence). If you don’t know in advance the length of the output sequence, you can use the padding trick: always output the same size sequence, but ignore any outputs that come after the end-of-sequence token (by ignoring them when computing the cost function).**

1. What is a common way to distribute training and execution of a deep RNN across multiple GPUs?

**Ans : To distribute training and execution of a deep RNN across multiple GPUs, a common technique is simply to place each layer on a different GPU**