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. How many dimensions must the inputs of an RNN layer have? What does each dimension represent? What about its outputs?
3. If you want to build a deep sequence-to-sequence RNN, which RNN layers should have return\_sequences=True? What about a sequence-to-vector RNN?
4. Suppose you have a daily univariate time series, and you want to forecast the next seven days. Which RNN architecture should you use?
5. What are the main difficulties when training RNNs? How can you handle them?
6. Can you sketch the LSTM cell’s architecture?
7. Why would you want to use 1D convolutional layers in an RNN?
8. Which neural network architecture could you use to classify videos?
9. Train a classification model for the SketchRNN dataset, available in TensorFlow Datasets.

Answer:

1.

* Sequence-to-sequence RNN: language translation, speech recognition, chatbot, image captioning, summarization.
* Sequence-to-vector RNN: sentiment analysis, document classification, stock price prediction, movie rating prediction.
* Vector-to-sequence RNN: image generation from text, video captioning, music generation.

The inputs of an RNN layer have three dimensions: batch size, sequence length, and input dimension. The batch size represents the number of samples in each batch, the sequence length represents the length of the input sequence, and the input dimension represents the number of features in each time step. The outputs of an RNN layer also have three dimensions: batch size, sequence length (if return\_sequences=True), and output dimension.

In a deep sequence-to-sequence RNN, all RNN layers should have return\_sequences=True, except for the last layer, which should have return\_sequences=False. This allows the intermediate layers to propagate information through the sequence. In a sequence-to-vector RNN, only the last RNN layer should have return\_sequences=False, since we only need the final output.

For a daily univariate time series, a simple RNN or LSTM model would be sufficient. We can use a sequence-to-sequence architecture, where the input sequence is the past daily values and the output sequence is the predicted next seven days.

The main difficulties when training RNNs are vanishing gradients, exploding gradients, and long-term dependencies. Vanishing gradients occur when the gradients become very small, making it hard for the network to learn. Exploding gradients occur when the gradients become very large, leading to instability during training. Long-term dependencies refer to the difficulty of RNNs to remember information from earlier time steps. To handle these difficulties, we can use techniques such as gradient clipping, batch normalization, and the use of alternative RNN cells like LSTM and GRU.

1. input gate

|

+-----------+

| |

+-|-\ |

| | | |

| +---|-------+

previous cell | | \

state | +-|-\ +-|-\

| | | | | | |

+---+-|---+---|-|-+

| +-------|-+

| |

| cell state

| |

| +-------+

| | |

+-|-------+

|

output gate

|

output

7. 1D convolutional layers can be used in an RNN to learn local patterns in the input sequence, allowing the RNN to better capture long-term dependencies. The 1D convolutional layer acts as a pre-processing step before the RNN layer, reducing the dimensionality of the input while preserving important features.

A common neural network architecture for video classification is a 3D convolutional neural network (CNN) followed by a recurrent neural network (RNN) or a fully connected layer. The 3D CNN processes the spatiotemporal information in the video frames, while the RNN or fully connected layer aggregates the information across time and produces the final classification.

9.

import tensorflow as tf

import tensorflow\_datasets as tfds

# Load the dataset

data, info = tfds.load('sketchrnn', split='train[:80%]', with\_info=True)

num\_classes = info.features['label'].num\_classes

# Preprocess the data

def preprocess\_data(sample):

image = tf.cast(sample['image'], tf.float32) /