**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?**

* **Sequence-to-sequence RNN**:
  + Machine translation (e.g., translating English to French).
  + Text summarization.
  + Speech recognition.
* **Sequence-to-vector RNN**:
  + Sentiment analysis (e.g., classifying a sentence as positive or negative).
  + Time series forecasting (e.g., predicting the next value in a sequence).
* **Vector-to-sequence RNN**:
  + Image captioning (e.g., generating a description for an image).
  + Text generation (e.g., generating a sentence from a seed word).

**2. How many dimensions must the inputs of an RNN layer have? What does each dimension represent? What about its outputs?**

* **Inputs**: 3D tensor with shape (batch\_size, timesteps, input\_dim).
  + batch\_size: Number of sequences in the batch.
  + timesteps: Number of time steps in each sequence.
  + input\_dim: Number of features at each time step.
* **Outputs**: 3D tensor with shape (batch\_size, timesteps, units) if return\_sequences=True, or 2D tensor with shape (batch\_size, units) if return\_sequences=False.
  + units: Number of neurons in the RNN layer.

**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?**

* **Sequence-to-sequence RNN**:
  + All RNN layers except the last one should have return\_sequences=True.
* **Sequence-to-vector RNN**:
  + Only the last RNN layer should have return\_sequences=False; all others should have return\_sequences=True.

**4. Suppose you have a daily univariate time series, and you want to forecast the next seven days. Which RNN architecture should you use?**

* Use a **sequence-to-sequence RNN** with return\_sequences=True in all layers except the last one. The output layer should predict the next seven days (e.g., using a Dense layer with 7 units).

**5. What are the main difficulties when training RNNs? How can you handle them?**

* **Difficulties**:
  + **Vanishing/exploding gradients**: Gradients can become too small or too large, making training unstable.
  + **Long-term dependencies**: RNNs struggle to capture dependencies over long sequences.
* **Solutions**:
  + Use **LSTM** or **GRU** cells to handle long-term dependencies.
  + Use **gradient clipping** to prevent exploding gradients.
  + Use **batch normalization** or **layer normalization** to stabilize training.

**6. Can you sketch the LSTM cell's architecture?**

* An LSTM cell consists of:
  + **Input gate**: Controls what new information is stored in the cell state.
  + **Forget gate**: Controls what information is discarded from the cell state.
  + **Output gate**: Controls what information is output from the cell state.
  + **Cell state**: Stores long-term information.
  + **Hidden state**: Outputs information for the current time step.

**7. Why would you want to use 1D convolutional layers in an RNN?**

* **Advantages**:
  + **Efficiency**: 1D convolutions are faster to compute than RNNs.
  + **Local feature extraction**: Convolutions can capture local patterns in the sequence.
  + **Parallelization**: Convolutions can be parallelized, unlike RNNs.
* **Use cases**:
  + Text classification.
  + Time series analysis.

**8. Which neural network architecture could you use to classify videos?**

* Use a **3D CNN** or a **CNN-RNN hybrid**:
  + **3D CNN**: Applies 3D convolutions to capture spatial and temporal features.
  + **CNN-RNN**: Uses a CNN to extract spatial features from each frame and an RNN to model temporal dependencies.

**9. Train a classification model for the SketchRNN dataset, available in TensorFlow Datasets.**

* Example code:

import tensorflow as tf

import tensorflow\_datasets as tfds

# Load the SketchRNN dataset

dataset, info = tfds.load('sketchrnn', with\_info=True, as\_supervised=True)

train\_data, test\_data = dataset['train'], dataset['test']

# Preprocess the data

def preprocess(data):

sketches, labels = data['image'], data['label']

sketches = tf.image.resize(sketches, [28, 28])

sketches = tf.cast(sketches, tf.float32) / 255.0

return sketches, labels

train\_data = train\_data.map(preprocess).batch(32)

test\_data = test\_data.map(preprocess).batch(32)

# Build the model

model = tf.keras.Sequential([

tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

tf.keras.layers.MaxPooling2D((2, 2)),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(10, activation='softmax')

])

# Compile and train the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(train\_data, epochs=10, validation\_data=test\_data)