```
1   import tensorflow as tf
2   cell = tf.nn.rnn_cell.LSTMCell(7)
3   # Input sequences for the LSTM
4   # Shape: (batch_size, time_steps, embed_dim)
5   input_sequences = tf.placeholder(
6       tf.float32,
7       shape=(None, 10, 20)
8   )
9   output, final_state = tf.nn.dynamic_rnn(
10      cell,
11      input_sequences,
12      dtype=tf.float32
13  )
```

RUN   SAVE   RESET

The `tf.nn.dynamic_rnn` function returns a tuple containing the RNN outputs as well as the final state of the RNN. For now, we only need to focus on the RNN output.

You'll notice from the example that the output first and second dimensions are equal to the input batch. This is because the RNN calculates the output for each time step of each sequence in the input batch. The third dimension, however, is equal to the number of hidden units in the cell object. For RNNs with multiple cells (i.e. `MultiRNNCell` cell object), the third dimension is equal to the number of hidden units in the final cell.

B. Sequence lengths

Because each of the input sequences can have varying lengths, it is likely that many of them will contain padding. Since padding is essentially a sequence filler, and therefore adds nothing of value to the RNN, we don't really want the RNN to waste computation on the padded parts of a sequence.

Instead, we can use the `sequence_length` argument in `tf.nn.dynamic_rnn`. This argument takes in a 1-D integer tensor, specifying the non-padded lengths of each sequence in the input batch.

Below is an example that uses `sequence_length` in `tf.nn.dynmaic_rnn`. The `cell` and `input_sequences` variables are the same as ones from the previous example. In this case, the input batch size is 5.

LSTM Output

Run an LSTM model on input sequences and retrieve the output.

Chapter Goals:

- Compute the output of your LSTM model

A. TensorFlow implementation

In TensorFlow, the way we create and run an RNN is with the function `tf.nn.dynamic_rnn`. The function takes in two required arguments. The first is the cell object that is used to create the RNN (e.g. an `LSTMCell`, `MultiRNNCell`, etc.). The second is the batch of input sequences, which are usually first converted to word embedding sequences.

Of the keyword arguments for the function, it is required that either `initial_state` or `dtype` is set. The `initial_state` argument specifies the starting state for the input cell object. We'll use this argument in later parts of this section.

The `dtype` argument specifies the type of both the initial cell state and RNN output. Most of the time, we can just set this argument to `tf.float32`, since the RNN outputs are normally floating point numbers.

Below is an example demonstrating how to use `tf.nn.dynamic_rnn`. Note that the input sequences have maximum length 10 and embedding size 12.

```python
import tensorflow as tf
lens = [4, 9, 10, 5, 10]
cell = tf.nn.rnn_cell.LSTMCell(7)
input_sequences = tf.placeholder(
    tf.float32,
    shape=(None, 10, 20)
)
output, final_state = tf.nn.dynamic_rnn(
    cell,
    input_sequences,
    sequence_length=lens,
    dtype=tf.float32
)
```

RUN   SAVE   RESET

By using the `sequence_length` argument, the function can skip unnecessary computation for the padded parts of each sequence, which can greatly reduce training time.