Multiple Layers

Stack multiple LSTM cell layers for added performance.

Chapter Goals:

- Learn how to stack multiple cell layers in an RNN

A. Stacking layers

Similar to how we can stack hidden layers in an MLP, we can also stack cell layers in an RNN. Adding cell layers allows the model to pick up on more complex features from the input sequence and therefore improve performance when trained on a large enough dataset.

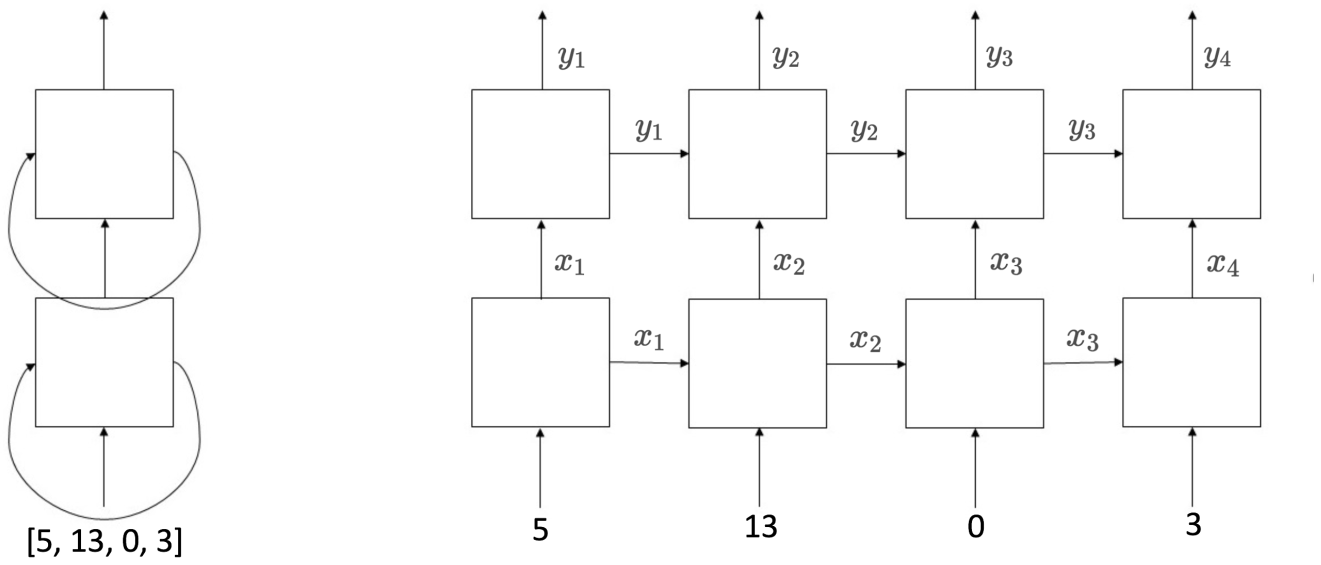

Diagram of an RNN with 2 cell layers. On the left is the rolled RNN, on the right is the unrolled RNN with 4 time steps.

In the diagram above, the RNN contains 2 cell layers (which is just two cells). At each time step, the first cell's output becomes the input for the second cell, and the second cell's output is the overall RNN's output for that particular time step. One thing to note is that the cell states remain separate, i.e. there aren't any recurrent connections between the two cells.

The input in this example is `[5, 13, 0, 3]`, which represents the tokenized sequence for some four word sentence. x_1, x_2, x_3, and x_4 represent the

outputs for the first layer. y_1, y_2, y_3, and y_4 represent the outputs for the second layer.
As with all other neural networks, adding layers can improve performance on larger datasets but also run the risk of overfitting the data. This makes regularization techniques such as dropout more important as we increase the size of our model.

Time to Code!

In this chapter, you'll be modifying the `stacked_lstm_cells` function to create a multi-layer LSTM model.

To create a multi-layer LSTM model, we need to set up a list containing the LSTM cells we want to use in our model. The number of cells will be equal to `self.num_lstm_layers`.

The dropout keep probability for each cell depends on whether we're training the model or not. If the model is training, we'll set the probability to 0.5, otherwise we'll set it to 1.0.

Set `dropout_keep_prob` equal to `0.5` if `is_training` is `True`, otherwise set it to `1.0`.

Create a list called `cell_list`, which contains `self.num_lstm_layers` number of LSTM cells. Each LSTM cell in the list should be the output of a distinct call to `self.make_lstm_cell`, with `dropout_keep_prob` as the only argument.

We can combine each of the LSTM cells into a usable TensorFlow object with the `tf.nn.rnn_cell.MultiRNNCell` function. The function takes in a list of cells as its required argument and returns a `MultiRNNCell` object representing the stacked cells.

Set `cell` equal to `tf.nn.rnn_cell.MultiRNNCell` applied with `cell_list` as the required argument. Then return `cell`.

```
import tensorflow as tf

# LSTM Language Model
class LanguageModel(object):
    # Model Initialization
    def __init__(self, vocab_size, max_length, num_lstm_units, num_lstm_layers):
        self.vocab_size = vocab_size
        self.max_length = max_length
        self.num_lstm_units = num_lstm_units
        self.num_lstm_layers = num_lstm_layers
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=vocab_size)

    # Create a cell for the LSTM
    def make_lstm_cell(self, dropout_keep_prob):
        cell = tf.nn.rnn_cell.LSTMCell(self.num_lstm_units)
        return tf.nn.rnn_cell.DropoutWrapper(
            cell, output_keep_prob=dropout_keep_prob)

    # Stack multiple layers for the LSTM
    def stacked_lstm_cells(self, is_training):
        # CODE HERE
        pass
```