Combined State

Combine the final states for a BiLSTM into usable initial states.

Chapter Goals:

- Combine the final states for each BiLSTM layer

A. LSTMStateTuple initialization

We initialize an `LSTMStateTuple` object with a hidden state (`c`) and state output (`h`).

Below we show an example of initializing an `LSTMStateTuple` object using the final forward and backward states from a single layer BiLSTM encoder.

```
import tensorflow as tf

# Final states of single-layer BiLSTM
# Forward and backward cells both have 5 hidden units
state_fw, state_bw = final_states

# Concatenate along final axis
final_c = tf.concat([state_fw.c, state_bw.c], -1)
final_h = tf.concat([state_fw.h, state_bw.h], -1)

combined_state = tf.nn.rnn_cell.LSTMStateTuple(
    final_c, final_h)
print(combined_state)
```

In the above example, we combined the BiLSTM forward and backward states into a single `LSTMStateTuple` object, which can be passed into the decoder.

For BiLSTM encoders with multiple layers, we combine the states for each layer to create a tuple of `LSTMStateTuple` objects. The element at index i of the tuple is the i^{th} layer's combined final state.

Time to Code!

In this chapter you'll be finishing the `for` loop of the `encoder` function. This is on line **65** of the code editor.

For each BiLSTM layer, we create a combined state using the combined `c` and `h` properties from the previous chapter.

Inside the `for` loop, set `bi_lstm_state` equal to `tf.nn.rnn_cell.LSTMStateTuple`, initialized with `bi_state_c` and `bi_state_h`.

After creating the layer's final state, we append it to the end of `combined_state`.

Inside the `for` loop, append `bi_lstm_state` to the end of `combined_state`.

```
import tensorflow as tf
tf_fc = tf.contrib.feature_column
tf_s2s = tf.contrib.seq2seq

# Get c and h vectors for bidirectional LSTM final states
def get_bi_state_parts(state_fw, state_bw):
    bi_state_c = tf.concat([state_fw.c, state_bw.c], -1)
    bi_state_h = tf.concat([state_fw.h, state_bw.h], -1)
    return bi_state_c, bi_state_h

# Seq2seq model
class Seq2SeqModel(object):
    def __init__(self, vocab_size, num_lstm_layers, num_lstm_units):
        self.vocab_size = vocab_size
        # Extended vocabulary includes start, stop token
        self.extended_vocab_size = vocab_size + 2
        self.num_lstm_layers = num_lstm_layers
        self.num_lstm_units = num_lstm_units
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(
            num_words=vocab_size)

    def make_lstm_cell(self, dropout_keep_prob, num_units):
        cell = tf.nn.rnn_cell.LSTMCell(num_units)
        return tf.nn.rnn_cell.DropoutWrapper(cell, output_keep_prob=dropout_keep_prob)

    # Create multi-layer LSTM
    def stacked_lstm_cells(self, is_training, num_units):
        dropout_keep_prob = 0.5 if is_training else 1.0
        cell_list = [self.make_lstm_cell(dropout_keep_prob, num_units) for i in range(self.nu
        cell = tf.nn.rnn_cell.MultiRNNCell(cell_list)
        return cell

    # Get embeddings for input/output sequences
    def get_embeddings(self, sequences, scope_name):
        with tf.variable_scope(scope_name):
            cat_column = tf_fc.sequence_categorical_column_with_identity(
                'sequences',
                self.extended_vocab_size)
```

```
            embedding_column = tf.feature_column.embedding_column(
                cat_column,
                int(self.extended_vocab_size**0.25))
            seq_dict = {'sequences': sequences}
            embeddings, sequence_lengths = tf_fc.sequence_input_layer(
                seq_dict,
                [embedding_column])
            return embeddings, tf.cast(sequence_lengths, tf.int32)

    # Create the encoder for the model
    def encoder(self, encoder_inputs, is_training):
        input_embeddings, input_seq_lens = self.get_embeddings(encoder_inputs, 'encoder_emb')
        cell_fw = self.stacked_lstm_cells(is_training, self.num_lstm_units)
        cell_bw = self.stacked_lstm_cells(is_training, self.num_lstm_units)
        enc_outputs, final_states = tf.nn.bidirectional_dynamic_rnn(
            cell_fw,
            cell_bw,
            input_embeddings,
            sequence_length=input_seq_lens,
            dtype=tf.float32)
        states_fw, states_bw = final_states
        combined_state = []
        for i in range(self.num_lstm_layers):
            bi_state_c, bi_state_h = get_bi_state_parts(
                states_fw[i], states_bw[i]
            )
            # CODE HERE
        final_state = tuple(combined_state)
        return enc_outputs, input_seq_lens, final_state
```