Final States

Learn about the final state output of an LSTM and BiLSTM.

Chapter Goals:

- Learn about the final states for an LSTM and BiLSTM

A. The encoder

In an encoder-decoder model for seq2seq tasks, there are two components: the encoder and the decoder. The encoder is responsible for extracting useful information from the input sequence. For NLP applications, the encoder is normally an LSTM or BiLSTM.

B. LSTM final state

When using an LSTM or BiLSTM encoder, we need to pass the final state of the encoder into the decoder. The final state of an LSTM in TensorFlow is represented by the `LSTMStateTuple` object.

```
import tensorflow as tf

# Input sequences (embedded)
# Shape: (batch_size, max_seq_len, embed_dim)
input_embeddings = tf.placeholder(
    tf.float32, shape=(None, None, 4))

cell = tf.nn.rnn_cell.LSTMCell(5)
_, final_state = tf.nn.dynamic_rnn(
    cell,
    input_embeddings,
    dtype=tf.float32)

# final_state is the output of our LSTM encoder.
# it contains all the information about our input sequence,
# which in this case is just a tf.Placeholder object
print(final_state)
```

The final state of an LSTM, i.e. the second element of the returned tuple from tf.nn.dynamic_rnn.

An `LSTMStateTuple` object contains two important properties: the hidden state (`c`) and the state output (`h`). The hidden state represents the internal cell state (i.e. "memory") of the LSTM cell.

These two properties are represented by tensors with shape `(batch_size, hidden_units)`.

C. Multi-layer final states

For a multi-layer LSTM, the final state output of `dynamic_rnn` is a tuple containing the final state for each layer.

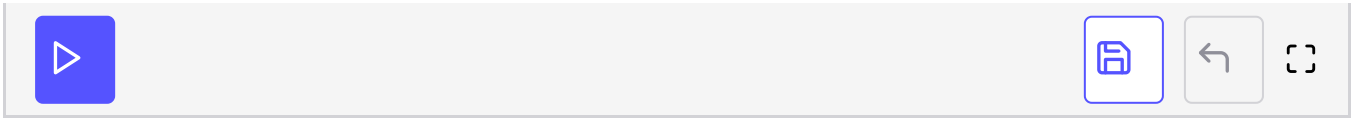
```
import tensorflow as tf

# Input sequences (embedded)
# Shape: (batch_size, max_seq_len, embed_dim)
input_embeddings = tf.placeholder(
    tf.float32, shape=(None, None, 4))

cell1 = tf.nn.rnn_cell.LSTMCell(5)
cell2 = tf.nn.rnn_cell.LSTMCell(8)
multi_cell = tf.nn.rnn_cell.MultiRNNCell([cell1, cell2])
_, final_state = tf.nn.dynamic_rnn(
    multi_cell,
    input_embeddings,
    dtype=tf.float32)

print(len(final_state))
print('\n')

print(final_state[0]) # layer 1
print('\n')

print(final_state[1]) # layer 2
print('\n')
```

The final state of a two-layer LSTM.

D. BiLSTM final state

The final state of a BiLSTM consists of the final state for the forward LSTM and the final state for the backward LSTM. This corresponds to a tuple containing two `LSTMStateTuple` objects.

```
import tensorflow as tf

# Input sequences (embedded)
# Shape: (batch_size, max_seq_len, embed_dim)
input_embeddings = tf.placeholder(
    tf.float32, shape=(None, None, 4))

cell_fw = tf.nn.rnn_cell.LSTMCell(5)
cell_bw = tf.nn.rnn_cell.LSTMCell(5)
_, final_states = tf.nn.bidirectional_dynamic_rnn(
    cell_fw,
    cell_bw,
    input_embeddings,
    dtype=tf.float32)

print(len(final_states))
print('\n')

print(final_states[0]) # forward
print('\n')

print(final_states[1]) # backward
print('\n')
```

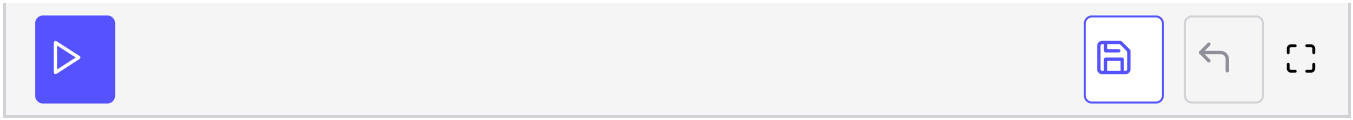


The final state of a BiLSTM, i.e. the second element of the returned tuple from tf.nn.bidirectional_dynamic_rnn.

When the BiLSTM has multiple layers, the `final_states` output of `bidirectional_dynamic_rnn` is a tuple containing the final forward and backward states for each layer.

```
import tensorflow as tf

# Input sequences (embedded)
# Shape: (batch_size, max_seq_len, embed_dim)
input_embeddings = tf.placeholder(
    tf.float32, shape=(None, None, 4))

cell_fw1 = tf.nn.rnn_cell.LSTMCell(5)
cell_fw2 = tf.nn.rnn_cell.LSTMCell(9)
multi_cell_fw = tf.nn.rnn_cell.MultiRNNCell(
    [cell_fw1, cell_fw2])

cell_bw1 = tf.nn.rnn_cell.LSTMCell(5)
cell_bw2 = tf.nn.rnn_cell.LSTMCell(9)
multi_cell_bw = tf.nn.rnn_cell.MultiRNNCell(
    [cell_bw1, cell_bw2])

_, final_states = tf.nn.bidirectional_dynamic_rnn(
    multi_cell_fw,
    multi_cell_bw,
    input_embeddings,
    dtype=tf.float32)
```

```
final_fw, final_bw = final_states


print(final_fw[0]) # forward layer 1
print('\n')

print(final_fw[1]) # forward layer 2
print('\n')

print(final_bw[0]) # backward layer 1
print('\n')

print(final_bw[1]) # backward layer 2
print('\n')
```



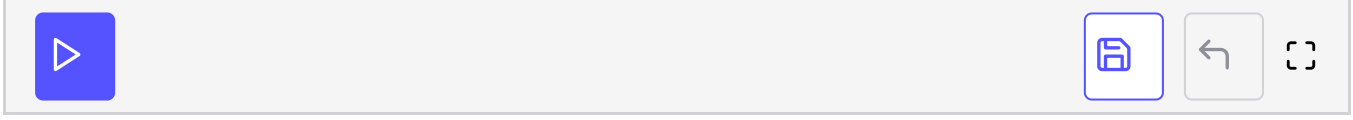
The final forward and backward state tuples for a two-layer BiLSTM.

E. Combining forward and backward

In order to use BiLSTM final states in an encoder-decoder model, we need to combine the forward and backward states. This is because the decoder portion utilizes a regular LSTM, which only has a forward direction.

Luckily, combining the forward and backward states is rather simple. The main thing we need to do is concatenate the hidden state and state output of both the forward and backward states.

```
import tensorflow as tf

# Forward state of single-layer BiLSTM final states
fw_c, fw_h = final_states[0].c, final_states[0].h

# Backward state of single-layer BiLSTM final states
bw_c, bw_h = final_states[1].c, final_states[1].h

# Concatenate along final axis
final_c = tf.concat([fw_c, bw_c], -1)
final_h = tf.concat([fw_h, bw_h], -1)
```



Concatenating the hidden state (c) and state output (h). Note that the concatenation is applied along the final axis.

We'll describe in the next chapter how to convert the combined values into an `LSTMStateTuple` object.

Time to Code!

In this chapter you'll be completing the `get_bi_state_parts` function, which combines the forward and backward final states for a layer in the BiLSTM. The function is used as a helper in the `Seq2Seq` object's `encoder` function, which creates the encoder portion of the model.

When combining the forward and backward states, we use the `tf.concat` function to concatenate along the final axis. We perform separate concatenations for the hidden state (`c`) and the state output (`h`).

Set `bi_state_c` equal to `tf.concat` applied with a list containing `state_fw.c` and `state_bw.c` (in that order), and `-1` as the axis.

Set `bi_state_h` equal to `tf.concat` applied with a list containing `state_fw.h` and `state_bw.h` (in that order), and `-1` as the axis.

Return a tuple containing `bi_state_c` as the first element and `bi_state_h` as the second.

```
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):
    # CODE HERE
    pass

# 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]
            )
```