Encoder-Decoder

Learn about the encoder-decoder model architecture.

Chapter Goals:

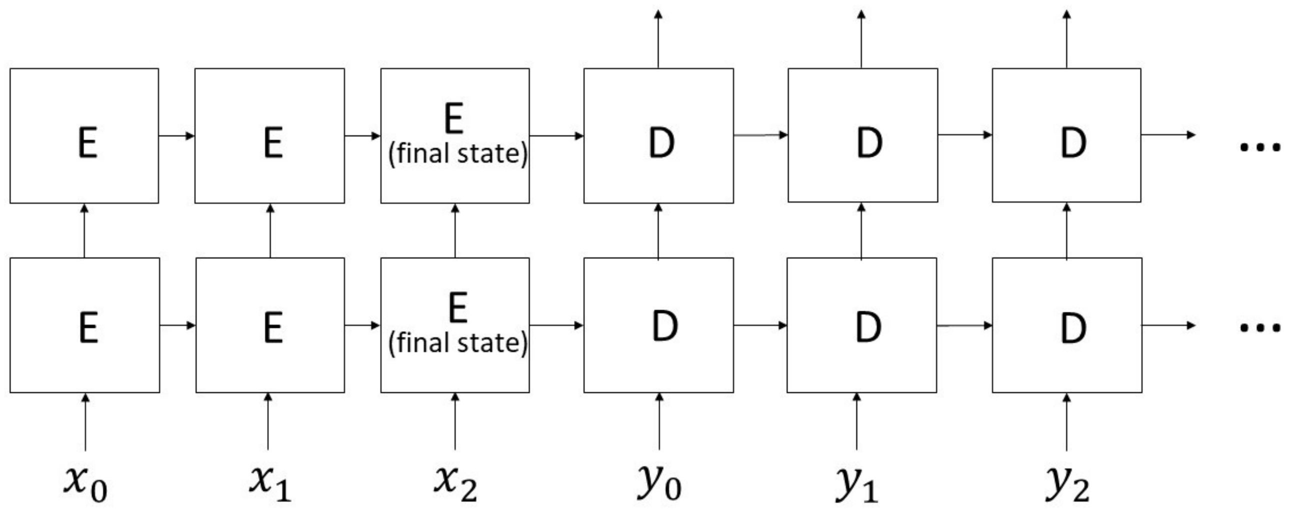
- Learn about the encoder-decoder model architecture

A. Model architecture

As previously mentioned, the encoder portion of the encoder-decoder model in this section is a BiLSTM. The decoder portion is just a regular forward LSTM, with the same number of LSTM layers as the encoder.

What makes an encoder-decoder model so powerful is that the decoder uses the final state of the encoder as its initial state. This gives the decoder access to the information that the encoder extracted from the input sequence, which is crucial for good sequence to sequence modeling.

In the case of multiple layers for the encoder-decoder model, each layer's final state in the encoder acts as the initial state for the corresponding layer in the decoder.



An unrolled encoder-decoder model with two LSTM layers. The encoder cells at each time step are labeled with "E", while the decoder's are labeled with "D".

In the above diagram, the final state of the LSTM encoder for each layer is used as the starting state for the corresponding decoder layer. Note that the encoder does not return outputs. This is because, in an encoder-decoder model, we ignore the encoder's outputs and only use the outputs of the decoder.

The input tokens for the encoder represent the input sequence for the encoder-decoder model. The input tokens for the decoder represent the ground truth tokens for the given input sequence. The decoder's output is equivalent to the output of a language model, i.e. each time step's output is based on the ground truth tokens from the previous time steps.

Note that the encoder in this diagram is a regular LSTM, rather than a BiLSTM. In the case of a BiLSTM, the starting state for each decoder layer is the combined final state from the corresponding BiLSTM layer (which you computed in the previous chapter).

B. Training vs. inference

During training, we have access to both the input and output sequences for each training pair. This gives the decoder access to the ground truth tokens.

However, during inference (i.e. real-time predictions), the model only has access to the input sequence. This means that the decoder doesn't have access to ground truth tokens. We handle this by using the *decoder's* word predictions from previous time steps to replace the ground truth tokens.

This will be covered in greater detail in chapter 11, when we discuss methods for decoding during inference.

Time to Code!

In this chapter you'll be writing some code in the `create_decoder_cell` function (which you'll subsequently complete in the next chapter). The function creates stacked LSTM cells for the decoder.

Each of the decoder LSTM cells contains twice the number of hidden units as the encoder LSTM cells to match the concatenated size of the encoder's forward and backward final states. We provide a function called `stacked_lstm_cells` which creates a (multi-layered) LSTM cell.

The `stacked_lstm_cells` function takes in the `is_training` boolean as the first argument and the number of hidden units as the second argument.

Set `num_decode_units` equal to two times `self.num_lstm_units`.

Set `dec_cell` equal to `self.stacked_lstm_cells` with `is_training` and `num_decode_units` as the arguments.

In the **Text Classification** section, we combined the forward and backward outputs of a BiLSTM in order to use them. We provide a utility function called `combine_enc_outputs` which does this for you.

Set `combined_enc_outputs` equal to `self.combine_enc_outputs` applied to `enc_outputs`.

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

# 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 cells
    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

    # Helper funtion to combine BiLSTM encoder outputs
    def combine_enc_outputs(self, enc_outputs):
        enc_outputs_fw, enc_outputs_bw = enc_outputs
        return tf.concat([enc_outputs_fw, enc_outputs_bw], -1)

    # Create the stacked LSTM cells for the decoder
    def create_decoder_cell(self, enc_outputs, input_seq_lens, is_training):
        # CODE HERE
        pass
```