Training Helper

Create a Training Helper object for training the decoder.

Chapter Goals:

- Learn about the decoding process during training
- Create a `TrainingHelper` object to use for decoding

A. Decoding during training

During training, we have access to both the input and output sequences of a training pair. This means that we can use the output sequence's ground truth tokens as input for the decoder.

We'll get into the specifics of TensorFlow decoding, i.e. using the decoder to generate model outputs, in later chapters. However, before we perform any decoding, we need to set up a `Helper` object. For training, the `Helper` object instance we use is the `TrainingHelper`.

Below, we create a `TrainingHelper` object for decoding during training.

```
import tensorflow as tf

# Placeholder representing the
# batch of (embedded) input sequences for the decoder
# Shape: (batch_size, max_seq_len, embed_dim)
decoder_embeddings = tf.placeholder(
    tf.float32, shape=(None, None, 12)
)

# Placeholder representing the
# individual lengths of each input sequence in the batch
decoder_seq_lens = tf.placeholder(tf.int32, shape=(None,))

helper = tf.contrib.seq2seq.TrainingHelper(
    decoder_embeddings, decoder_seq_lens)
```

The `TrainingHelper` object is initialized with the (embedded) ground truth

sequences and the lengths of the ground truth sequences. Note that we use

separate embedding models for the encoder input and the decoder input (i.e. ground truth tokens). This is because there are different word relationships in the input and output sequences for a seq2seq task, and sometimes the sequences can be completely different (e.g. machine translation).

Time to Code!

In this chapter you'll be filling in part of the `create_decoder_helper` function, which creates the Helper object for the decoder. Specifically, in this chapter you'll be focusing on creating the `TrainingHelper` object.

We use the decoder differently depending on whether we're in training or inference mode. If we're in training mode, we use the `decoder_inputs` as the ground truth tokens.

Before using the ground truth tokens, we need to convert them into embeddings. We provide a function called `get_embeddings`, which returns the embeddings and sequence lengths. The function takes in the input sequences and the name of the embedding model as required arguments.

Inside the `if` block, set the tuple `dec_embeddings, dec_seq_lens` equal to `self.get_embeddings` applied with `decoder_inputs` and `'decoder_emb'` as the required arguments.

During training, we use the `TrainingHelper` object. It takes in the ground truth embeddings and sequence lengths as required arguments.

Inside the `if` block, set `helper` equal to `tf_s2s.TrainingHelper` applied with `dec_embeddings` and `dec_seq_lens` as the required arguments.

```
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)

# Convert sequences to embeddings
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 helper for decoding
def create_decoder_helper(self, decoder_inputs, is_training, batch_size):
    if is_training:
        # CODE HERE
        pass
    else:
        # IGNORE FOR NOW
        pass
    return helper, dec_seq_lens
```