Calculating Loss

Calculate the model's loss based on logits and sparse outputs.

Chapter Goals:

 Calculate the training loss based on the model's logits and final token sequences

A. Final token sequence

So far, we've used the input sequences and ground truth sequences for training the encoder-decoder model. The final token sequences are used when calculating the loss.

If we view the decoder as a language model, the ground truth sequences act as the language model's input while the final token sequences act as the "correct" output for the language model.

In a language model, we calculate the loss based on the model's logits and the "correct" output. For the encoder-decoder, the loss is calculated in the same way. We use a sparse softmax cross entropy loss based on the logits and final token sequences, and then zero-out the time steps that correspond to padding.

B. Sequence mask

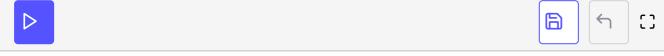
Previously, we used the <code>tf.sign</code> function to create binary sequences that are used to zero-out padding time steps. However, there is another way to create the binary sequences, also known as a *sequence mask*, using just the sequence lengths of the decoder's inputs.

The function that provides this utility is called tf.sequence_mask. It takes in a tensor of sequence lengths as its required argument, and returns the corresponding sequence mask.

```
seq_lens = tf.constant([3, 4, 2])
binary_sequences = tf.sequence_mask(seq_lens)
int_sequences = tf.sequence_mask(seq_lens, dtype=tf.int32)

with tf.Session() as sess:
    binary_array = sess.run(binary_sequences)
    int_array = sess.run(int_sequences)

print(repr(binary_array))
print(repr(int_array))
```



Creating sequence masks from sequence lengths using tf.sequence_mask.

Note that the default return type of tf.sequence_mask is a boolean tensor. We can set the dtype keyword argument to specify a different return type.

Time to Code!

In this chapter you'll be completing the calculate_loss function, which is
used to calculate the model's loss.

When calculating the loss, we need to zero-out any time steps that represent padding, i.e. time steps past the actual sequence length. We'll apply tf.sequence_mask to the decoder's input sequence lengths (dec_seq_lens) to obtain the necessary sequence mask.

Set binary_sequences equal to tf.sequence_mask applied with dec_seq_lens as the required argument. Also set the dtype keyword argument to tf.float32, so that binary_sequences has type tf.float32.

Just like in language modeling, we use sparse softmax cross entropy as the loss function. We also zero-out the loss at time steps that represent padding.

Set batch_loss equal to tf.nn.sparse_softmax_cross_entropy_with_logits applied with decoder_outputs and logits as the labels and logits keyword arguments, respectively.

Set unpadded_loss equal to batch_loss multiplied by binary_sequences.

The loss that we return will be the per-sequence loss, the average loss for each sequence in the training batch. This is equivalent to taking the entire loss and dividing it by the batch size.

Set per_seq_loss equal to tf.reduce_sum applied to unpadded_loss, divided by batch_size. Then return per_seq_loss.

```
import tensorflow as tf
                                                                                        C)
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)
   # Calculate the model loss
    def calculate_loss(self, logits, dec_seq_lens, decoder_outputs, batch_size):
        # CODE HERE
        pass
```









٢٦