Calculating Loss

Calculate the loss for your LSTM model.

Chapter Goals:

- Convert your LSTM model's outputs into logits
- Use a padding mask to calculate the overall loss

A. Logits & loss

As mentioned in earlier chapters, the task for a language model is no different from regular multiclass classification. Therefore, the loss function will still be the regular softmax cross entropy loss. We use a final fully-connected layer to convert model outputs into logits for each of the possible classes (i.e. vocabulary words).

```
import tensorflow as tf
# Output from an LSTM
# Shape: (batch_size, time_steps, cell_size)
lstm_outputs = tf.placeholder(tf.float32, shape=(None, 10, 7))

vocab_size = 100
logits = tf.layers.dense(lstm_outputs, vocab_size)

# Target tokenized sequences
# Shape: (batch_size, time_steps)
target_sequences = tf.placeholder(tf.int64, shape=(None, 10))
loss = tf.nn.sparse_softmax_cross_entropy_with_logits(
    labels=target_sequences,
    logits=logits)
```

Obtaining the loss for an LSTM model (with max sequence length of 5).

The function used to calculate the softmax cross entropy loss for feed-forward neural networks is `tf.nn.softmax_cross_entropy_with_logits`. However, we can only use this function if the `labels` and `logits` arguments both have the same shape.

In our example, `logits` has 3 dimensions while `labels` (`target_sequences`) only has 2. In this case, the `labels` are referred to as *sparse* (i.e. they represent class indexes rather than one-hot vectors), so we use the sparse version of the loss function.

B. Padding mask

When we calculate the loss based on the model's outputs, we don't want to include the logits for every time step in each sequence. Specifically, we want to exclude the loss calculated for the padded time steps, since those values are meaningless. Therefore, we use a *padding mask* to zero-out the loss at padded time steps.

The padding mask will have the same shape as the labels (i.e. target batch), but it will only contain 0's and 1's. Locations containing 0 represent padded time steps while locations containing 1 represent actual input sequence tokens. We multiply the padding mask by the loss to zero-out the padded time step locations.

The code below demonstrates an example usage of a padding mask, with batch size of 1 and max sequence length of 5. Note that we cast the padding mask to `tf.float32` so that it matches the type of the loss.

```
import tensorflow as tf
# loss: Softmax loss for LSTM
with tf.Session() as sess:
    print(repr(sess.run(loss)))

# Same shape as loss
pad_mask = tf.constant([
    [1., 1., 1., 1., 0.],
    [1., 1., 0., 0., 0.]
])

new_loss = loss * pad_mask
with tf.Session() as sess:
    print(repr(sess.run(new_loss)))
```

Time to Code!

In this chapter you'll be completing `calculate_loss` function, which calculates the model loss from the LSTM outputs.

First, we'll convert the outputs of the LSTM model into logits.

Set `logits` equal to `tf.layers.dense` applied with `lstm_outputs` and `self.vocab_size` as the two arguments.

Note that `logits` has shape `(batch_size, self.max_length, self.vocab_size)` while `output_sequences` (the batch of tokenized target sequences) has shape `(batch_size, self.max_length)`. Therefore, we use a sparse softmax cross entropy to compute the loss.

Set `batch_sequence_loss` equal to `tf.nn.sparse_softmax_cross_entropy_loss` applied with `output_sequences` and `logits` for the `labels` and `logits` keyword arguments, respectively.

To zero-out the loss for the padded time steps, we'll use `binary_sequences` (created in the previous chapter) as our padding mask. However, in order to use the padding mask, we need to first cast it to type `tf.float32` (to match the type of the loss).

Cast `binary_sequences` to type `tf.float32` and then multiply it by `batch_sequence_loss`. Store the output in the variable `unpadded_loss`.

The overall loss calculated by the function is the sum of the losses across every time step of each sequence.

Set `overall_loss` equal to `tf.reduce_sum` applied with `unpadded_loss` as the only argument. Then return `overall_loss`.

```
import tensorflow as tf

# LSTM Language Model
class LanguageModel(object):
    # Model Initialization
    def __init__(self, vocab_size, max_length, num_lstm_units, num_lstm_layers):
        self.vocab_size = vocab_size
        self.max_length = max_length
        self.num_lstm_units = num_lstm_units
        self.num_lstm_layers = num_lstm_layers
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=vocab_size)

    # Calculate model loss
    def calculate_loss(self, lstm_outputs, binary_sequences, output_sequences):
        # CODE HERE
        pass
```