RNN - TEXT CLASSIFICATION

```python
def model(FLAGS, train=True, sample=False):

    # Placeholders
    inputs_X = tf.placeholder(tf.int32, shape=[FLAGS.batch_size, None], name='inputs_X')
    targets_y = tf.placeholder(tf.float32, shape=[FLAGS.batch_size, FLAGS.num_classes], name='targets_y')

    # RNN cell
    if not train:
        FLAGS.dropout=0.0
    stacked_cell = rnn_cell(FLAGS)

    # Inputs to RNN
    with tf.variable_scope('rnn_inputs'):
        W_input = tf.get_variable("W_input", [FLAGS.en_vocab_size, FLAGS.num_hidden_units])

    inputs = rnn_inputs(FLAGS, inputs_X)
    initial_state = stacked_cell.zero_state(FLAGS.batch_size, tf.float32)

    # Outputs from RNN
    seq_lens = length(inputs_X)
    all_outputs, state = tf.nn.dynamic_rnn(cell=stacked_cell, inputs=inputs,
        initial_state=initial_state, sequence_length=seq_lens)

    if train:

        index, outputs = last_relevant(all_outputs, seq_lens)

        # Process RNN outputs
        with tf.variable_scope('rnn_softmax'):
            W_softmax = tf.get_variable("W_softmax", [FLAGS.num_hidden_units, FLAGS.num_classes])
            b_softmax = tf.get_variable("b_softmax", [FLAGS.num_classes])

        # Logits
        logits = rnn_softmax(FLAGS, outputs)
        probabilities = tf.nn.softmax(logits)
        accuracy = tf.equal(tf.argmax(targets_y,1), tf.argmax(logits,1))

        # Loss
        loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits, targets_y))
        final_state = state

        # Optimization
        lr = tf.Variable(0.0, trainable=False)
        trainable_vars = tf.trainable_variables()
        grads, _ = tf.clip_by_global_norm(tf.gradients(loss, trainable_vars),
                                       FLAGS.max_gradient_norm) # glip the gradient to avoid vanishing or blowing up gradients
        optimizer = tf.train.AdamOptimizer(lr)
        train_optimizer = optimizer.apply_gradients(zip(grads, trainable_vars))

        return dict(inputs_X=inputs_X, targets_y=targets_y, seq_lens=seq_lens, index=index, all_outputs=all_outputs,
            lr=lr, outputs=outputs, logits=logits, accuracy=accuracy, loss=loss, train_optimizer=train_optimizer)
```

embeddings (could also pre-train using skip-gram architecture for better performance)

input lengths of each sentence. dynamic_rnn will not process beyond this point

only get the relevant outputs

no need to mask the loss, as with many-to-many because we already isolated our relevant output