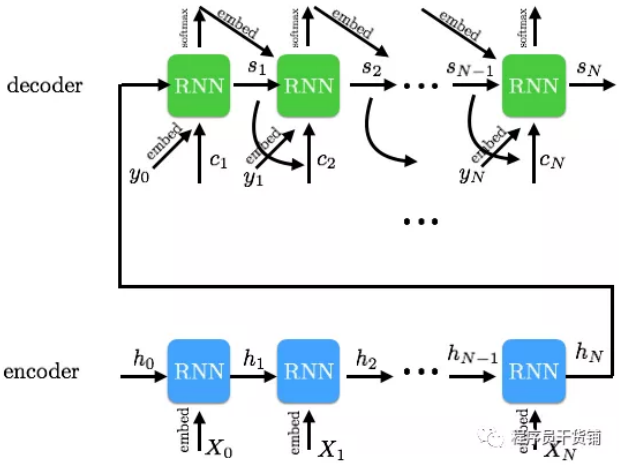
RNN成长记(三)：编码器和解码器

原创 刘训灼 程序员干货铺 2020-09-23

在本文中，我将介绍基本的编码器（encoder）和解码器（decoder），用于处理诸如机器翻译之类的 seq2seq 任务。我们不会在这篇文章中介绍注意力机制，而在下一篇文章中去实现它。

如下图所示，我们将输入序列输入给编码器，然后将生成一个最终的隐藏状态，并将其输入到解码器中。即编码器的最后一个隐藏状态就是解码器的新初始状态。我们将使用 softmax 来处理解码器输出，并将其与目标进行比较，从而计算我们的损失函数。这里的主要区别在于，我没有向编码器的输入添加 EOS（译注：句子结束符，end-of-sentence）token，同时我也没有让编码器对句子进行反向读取。



数据

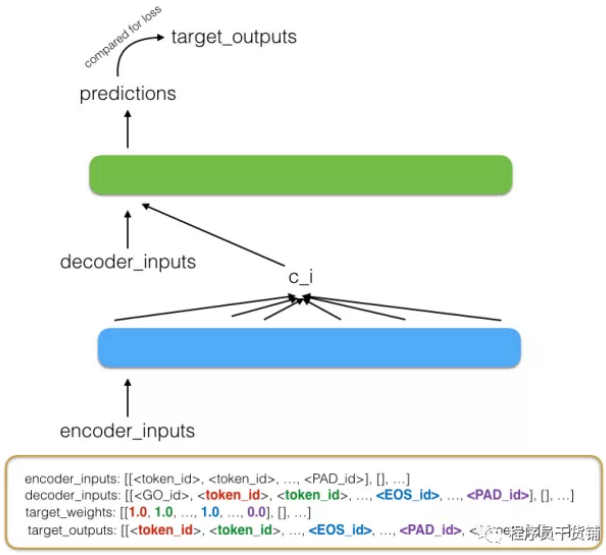
我想创建一个非常小的数据集来使用（20 个英语和西班牙语的句子）。本教程的重点是了解如何构建一个编码解码器系统，而不是去关注这个系统对诸如机器翻译和其他 seq2seq 处理等任务的处理。所以我自己写了几个句子，然后把它们翻译成西班牙语。这就是我们的数据集。

首先，我们将这些句子分隔为 token，然后将这些 token 转换为 token ID。在这个过程中，我们收集一个词汇字典和一个反向词汇字典，以便在 token 和 token ID 之间来回转换。对于我们的目标语言（西班牙语）来说，我们将添加一个额外的 EOS token。然后，我们会将源 token 和目标 token 都填充到（对应数据集中最长句子的）最大长度。这是我们模型的输入数据。对于编码器而言，我们将填充后的源内容直接进行输入，而对于目标内容做进一步处理，以获得我们的解码器输入和输出。

最后，输入结果是这个样子的：

```
encoder inputs:
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   4 141 163   8
 197  80   9 169 206   9  60 190  62  14  28 112 125  38]
decoder inputs
[  1  37 157 139  27   7 197 170  53 196   6   7 161 175   6  78 203   9
 158 140   4   2   0   0   0   0   0   0   0   0   0   0   0]
targets
[ 37 157 139  27   7 197 170  53 196   6   7 161 175   6  78 203   9 158
 140   4   2   0   0   0   0   0   0   0   0   0   0   0   0]
en_seq_lens
18
sp_seq_lens
21
```


这只是某个批次中的一个样本。其中 0 是填充的值，1 是 GO token，2 则是 EOS token。下图是数据变换更一般的表示形式。请无视目标权重，我们不会在实现中使用它们。



编码器

编码器只接受编码器的输入，而我们唯一关心的是最终的隐藏状态。这个隐藏的状态包含了所有输入的信息。我们不会像原始论文所建议的那样反转编码器的输入，因为我们使用的是 `dynamic_rnn` 的 `seq_len`。它会基于 `seq_len` 自动返回最后一个对应的隐藏状态。

```
with tf.variable_scope('encoder') as scope:

    # RNN 编码器单元
    self.encoder_stacked_cell = rnn_cell(FLAGS, self.dropout,
        scope=scope)

    # 嵌入 RNN 编码器输入
    W_input = tf.get_variable("W_input",
        [FLAGS.en_vocab_size, FLAGS.num_hidden_units])
    self.embedded_encoder_inputs = rnn_inputs(FLAGS,
        self.encoder_inputs, FLAGS.en_vocab_size, scope=scope)
```

```
    #initial_state = encoder_stacked_cell.zero_state(FLAGS.batch_size,
tf.float32)

    # RNN 编码器的输出
    self.all_encoder_outputs, self.encoder_state = tf.nn.dynamic_rnn(
        cell=self.encoder_stacked_cell,
        inputs=self.embedded_encoder_inputs,
        sequence_length=self.en_seq_lens, time_major=False,
        dtype=tf.float32)
```

我们将使用这个最终的隐藏状态作为解码器的新初始状态。

解码器

这个简单的解码器将编码器的最终的隐藏状态作为自己的初始状态。我们还将接入解码器的输入，并使用 RNN 解码器来处理它们。输出的结果将通过 softmax 进行归一化处理，然后与目标进行比较。注意，解码器输入从一个 GO token 开始，从而用来预测第一个目标 token。解码器输入的最后一个对应的 token 则是用来预测 EOS 目标 token 的。

```
with tf.variable_scope('decoder') as scope:

    # 初始状态是编码器的最后一个对应状态
    self.decoder_initial_state = self.encoder_state

    # RNN 解码器单元
    self.decoder_stacked_cell = rnn_cell(FLAGS, self.dropout,
        scope=scope)

    # 嵌入 RNN 解码器输入
    W_input = tf.get_variable("W_input",
        [FLAGS.sp_vocab_size, FLAGS.num_hidden_units])
    self.embedded_decoder_inputs = rnn_inputs(FLAGS, self.decoder_inputs,
        FLAGS.sp_vocab_size, scope=scope)

    # RNN 解码器的输出
    self.all_decoder_outputs, self.decoder_state = tf.nn.dynamic_rnn(
        cell=self.decoder_stacked_cell,
```

```
        inputs=self.embedded_decoder_inputs,
        sequence_length=self.sp_seq_lens, time_major=False,
        initial_state=self.decoder_initial_state)
```

那填充值会发生什么呢？它们也会预测一些输出目标，而我们并不关心这些内容，但如果我们把它们考虑进去，它们仍然会影响我们的损失函数。接下来我们将屏蔽掉这些损失以消除对目标结果的影响。

损失屏蔽

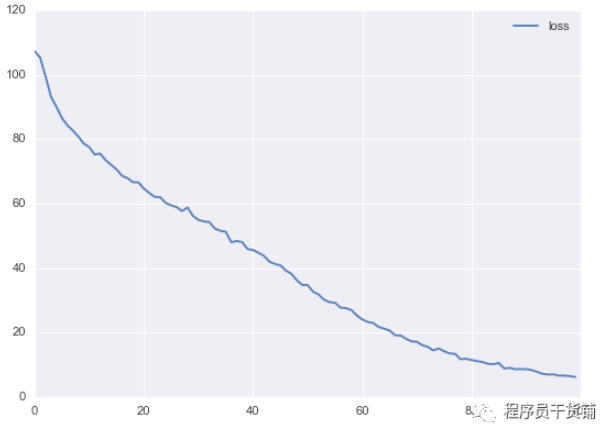
我们会检查目标，并将目标中被填充的部分屏蔽为 0。因此，当我们获得最后一个有关的解码器 token 时，目标就会是表示 EOS 的 token ID。而对于下一个解码器的输入而言，目标就会是 PAD ID，这也就是屏蔽开始的地方。

```
# Logit
self.decoder_outputs_flat = tf.reshape(self.all_decoder_outputs,
    [-1, FLAGS.num_hidden_units])
self.logits_flat = rnn_softmax(FLAGS, self.decoder_outputs_flat,
    scope=scope)

# 损失屏蔽
targets_flat = tf.reshape(self.targets, [-1])
losses_flat = tf.nn.sparse_softmax_cross_entropy_with_logits(
    self.logits_flat, targets_flat)
mask = tf.sign(tf.to_float(targets_flat))
masked_losses = mask * losses_flat
masked_losses = tf.reshape(masked_losses,  tf.shape(self.targets))
self.loss = tf.reduce_mean(
    tf.reduce_sum(masked_losses, reduction_indices=1))
```

注意到可以使用 PAD ID 为 0 这个事实作为屏蔽手段，我们便只需计算（一个批次中样本的）每一行损失之和即可，然后取所有样本损失的平均值，从而得到一个批次的损失。这时，我们就可以通过最小化这个损失函数来进行训练了。

以下是训练结果：



我们不会在这里做任何的模型推断，但是你可以在接下来的关于注意力机制的文章中看到。如果你真的想在这里实现模型推断，使用相同的模型就可以了，但你还得将预测目标的结果作为输入接入下一个 RNN 解码器单元。同时你还要将相同的权重集嵌入解码器中，并将其作为 RNN 的另一个输入。这意味着对于初始的 GO token 而言，你得嵌入一些伪造的 token 进行输入。

结论

这个编码解码器模型非常简单，但是在理解 seq2seq 实现之前，它是一个必要的基础。在下一篇 RNN 教程中，我们将涵盖 Attention 模型及其在编码解码器模型结构上的优势。