基于转移的短文本回复生成

June 28, 2017

1 短文本回复对和回复生成

短文本回复对是一种非常常见的对话结构。发言人就某话题发表简短的言论，其他人简短地回复自己的观点这样的“一发一回”的对话结构被称为短文本回复对。在微博上就有大量的短文本回复，因为微博用户可以通过文字对某一个话题发表短的言论，而其他用户可以以回复的方式来提出自己的观点。这样的回复对由两个部分构成：Post和Response。其中Post 是发言人发表的言论，Response则是某一个用户的回复内容。Table 1 中给出短文本回复对的例子。所谓的回复生成就是向模型输入Post，然后自动输出Response。下文将介绍如何通过基于转移的框架和神经网络抽取的特征来构建这样的模型。

Post	十点半主楼有XXX博士的关于深度学习的讲座。
Response	太悲催了，我有事没办法去了。
Post	我航学生实验室杂活——搬导弹，怎一个V5了得啊。
Response	啊！我没有抬过，亏了。
Post	想了一下，描述这棵树最贴切的两个字应该是：奇葩
Response	我倒觉得，相当惊艳呢。很像孔雀开屏呀。

Table 1: 短文本回复对的例子。

2 基于转移的框架

2.1 框架整体简介

基于转移的框架可以用于解决结构预测的问题，比如分词、分词词性标注、依存分析、短语结构分析、机器翻译等等任务。基于转移的框架有两个组成部分：状态和动作。其中状态表示处理过程中的部分结果，动作表示状态之间的转移规则。这个框架的状态从开始状态起，每次从候选动作中选择一个动作，再根据动作进行状态之间的转移，直到结束状态见Figure 1。整个过程实际上可以看成是一个搜索的过程，从诸多动作序列中搜索出最佳动作序列。对于回复生成来说，状态是Post 和部分生成好的Response；而动作表示下一个要生成的词；开始状态：Response 部分只有句子的开始符$< start >$；结束状态：Response 部分生成了句子结束符$< end >$。Table 2 反映了回复生成的过程。

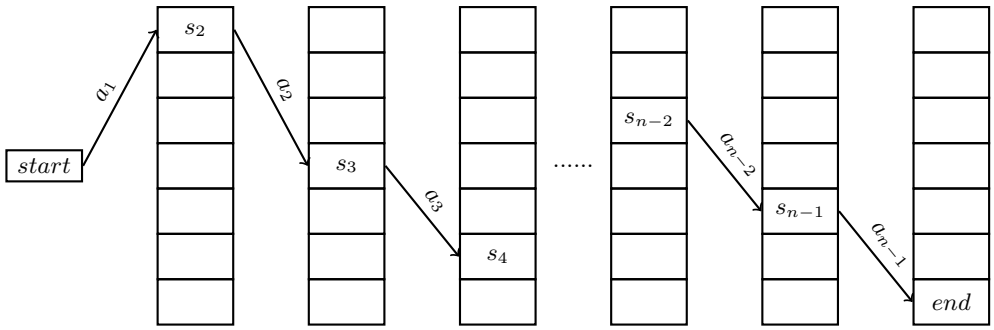

Figure 1: 状态根据动作进行转移，其中s_i表示状态，a_i表示动作

状态中的Response部分	动作候选池	下一个动作
	$< start >$	$< start >$
$< start >$	这、你、这个、好、太、...	太
$< start >$ 太	可爱、好、悲、美、牛、...	悲
$< start >$ 太 悲	催、了、过去、。不、！、...	催
$< start >$太 悲 催	了、啊、$< end >$、啦、。、，...	了
... 太 悲 催 了	。、$< end >$、，、！、吧、...	，
... 悲 催 了 ，	我、你、这、还、不、，就、好、...	我
... 催 了 ， 我	爱、只、很、就是、家、有事、...	有事
... 了 ， 我 有事	私密、想、要、找、没、...	没
... 我 有事 没	有、看、吃、想到、人、办法、了、...	办法
... 有事 没 办法	，、。、啊、了、！、把、去、...	去
... 没 办法 去	改变、看、了、。、的、一、看看、...	了
... 办法 去 了	。、，、？、！、啊、吗、吧、...	。
... 去 不 了 。	$< end >$、我、！、你、好、（、...	$< end >$
... 不 了 。 $< end >$		

Table 2: 基于转移的回复生成的过程

2.2 动作剪枝

在每次选择下一个动作时，在回复生成这个任务中会有大量的候选动作可供选择，这是由于语言本身的复杂性造成的，如此一来搜索最佳动作序列的时间会变得很漫长。 因此我们使用语言模型对候选动作进行删减来加快生成速度。通过语言模型，我们可以去除概率低的候选动作，比如当Response中已经生成了“$< start >$太 悲”，而候选池中的候选动作有“催、了、过去、。不、！、...”。 我们根据语言模型可以知道“$< start >$ 太 悲” 下一个词的概率分布，从中可以把概率低的词从候选池中删去。

2.3 搜索策略

训练完的模型在选择候选动作时，需要对每一个候选动作进行打分。然后再使用贪心策略，每次的动作选择只选择分数最高的。但是贪心策略无法选出最佳动作序列，这里之所以使用它是因为要提高速度，而且它很容易扩展成柱搜索。

2.4 动作分数

候选动作的分数来自两个部分：一个是候选动作本身，另一个是当前状态。我们需要抽取出这两个部分的特征来计算分数。这里的特征抽取并不局限于使用神经网络，也可以使用传统的特征模板。本文要介绍的是通过神经网络来对特征进行向量表示。计算候选动作分数的公式如下：

$$score(a_i) = h_{state_j} \cdot e(a_i)$$

其中$e(a_i)$表示a_i这个候选动作的向量表示；h_{state_j}表示状态的向量表示；$\cdot$表示矩阵乘法，故h_{state_j}和$e(a_i)$的维度一致。候选动作的向量表示$e(a_i)$ 可以随机初始化，然后微调即可，下面将说明如何通过神经网络来对状态进行向量表示。

2.4.1 状态的向量表示

状态的向量表示来自两个部分：其一，是Post的向量表示；其二，是已经生成的部分Response的向量表示，即已经生成的动作序列的向量表示。我们用下列公式直接将这两部分的向量进行拼接得到状态的向量表示：

$$h_{state_j} = h_{post} \oplus h_{act\_seq_j}$$

其中h_{state_j}表示状态的向量表示，h_{post}表示Post向量表示，$h_{act_seq_j}$表示动作序列的向量表示。

我们使用了一个卷积层和双向LSTM对Post文本内容进行向量表示，具体的结构见Figure 2。

动作序列的向量表示f_i则是由一个增量式的LSTM来表示，其中它的输入是上一个状态的最后两个动作的向量表示：

$$f_i = tanh(W_1 \cdot e_1(last\_action_1) + W_2 \cdot e_2(last\_action_2) + b)$$

其中f_i是最后两个动作的向量表示；W_1，W_2，b表示模型参数；$e_1(last\_action_1)$，$e_2(last_action_2)$表示上一个状态的最后两个向量表示，$tanh()$表示激活函数。具体的动作序列的向量表示见Figure 3，它随着状态的转移增量式地产生，并不是像h_{post}中的LSTM已知所有的输入向量，一次性完成向量表示。这是由于我们只能知道部分已经产生的动作序列，对没有产生的动作序列我们没有办法提前知道。至于为什么在已知的Response中只选择最后的两个动作的向量化表示作为增量式的LSTM的输入，是因为我们认为最后的这两个动作序列对候选动作的选择是一个重要的特征。比如Response中已经生成了“$< start >$太悲”，那我们认为“太”和“悲”对下一词的产生有大的参考价值。

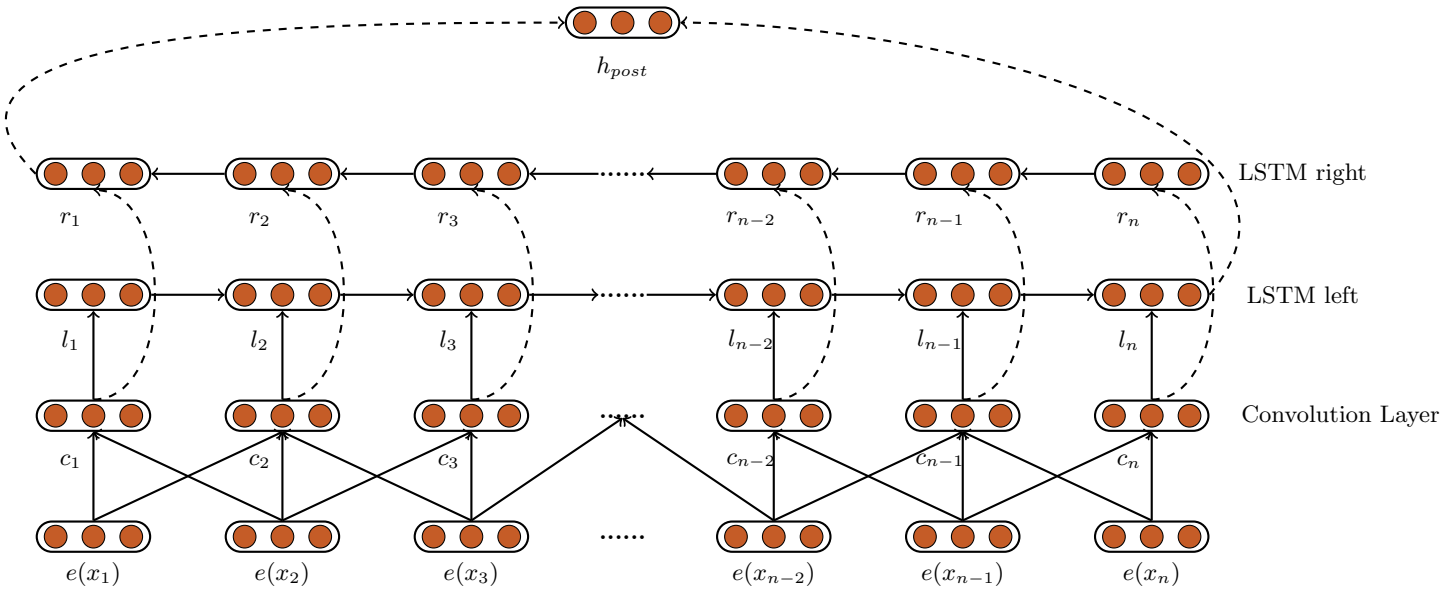

Figure 2: 图中h_{post}为Post的向量表示，$e(x_i)$为Post中词的向量表示，c_i为卷积层输出的向量表示，l_i和r_i为两层LSTM的输出向量表示。

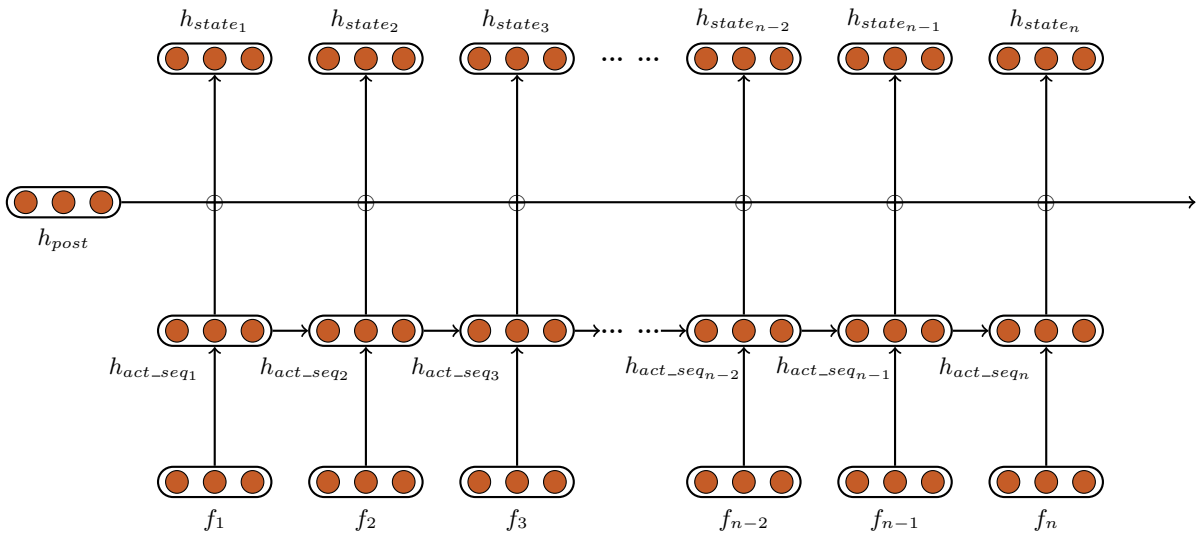

Figure 3: 图中h_{state_i} 为状态的向量表示，f_i为最后已生成的最后两个动作的向量表示，$h_{act\_seq_i}$ 为动作序列的向量表示。

3 模型训练

模型在训练时，已知Response中的内容，而且不会根据候选动作的分数来挑选动作，而是直接根据Response选择正确答案。模型的损失函数是交叉熵损失函数:

$$p_i = \frac{e^{score(a_i)}}{\Sigma e^{score(a_j)}}$$

$$loss(\theta) = -\Sigma g_i \cdot log(p_i)$$

其中$score(a_i)$ 是第i个候选动作的分数；g_i 表示第i个候选动作是否是正确的，1表示是，0表示否，这个信息从已知的Response中可以获得；θ表示模型参数。我们可以用梯度下降法来最小化损失函数。

用下面的例子可以看到最小化损失函数的过程，使模型对正确答案打出更高的分数。假设$h_{state} = (1, 1)$，两个候选动作$e(a_0) = (-1, -1)$和$e(a_1) = (1, 1)$，$g_0 = 1$，$g_1 = 0$。

根据候选动作分数的计算公式

$$score(a_i) = h_{state_j} \cdot e(a_i)$$

得$score(a_0) = -2$和$score(a_1) = 2$。再算出$p_0 = 1/(e^4 + 1)$和 $p_1 = e^4/(e^4 + 1)$。下面开始计算导数:

$$\frac{\delta loss(\theta)}{\delta score(a_0)} = p_0 - g_0 = \frac{-e^4}{e^4 + 1}$$

$$\frac{\delta loss(\theta)}{\delta score(a_1)} = p_1 - g_1 = \frac{e^4}{e^4 + 1}$$

接下来计算

$$\frac{\delta loss(\theta)}{\delta e(a_0)} = \frac{\delta loss(\theta)}{\delta score(a_0)} \cdot \frac{\delta score(a_0)}{\delta e(a_0)} = (\frac{-e^4}{e^4 + 1}, \frac{-e^4}{e^4 + 1})$$

$$\frac{\delta loss(\theta)}{\delta e(a_1)} = \frac{\delta loss(\theta)}{\delta score(a_1)} \cdot \frac{\delta score(a_1)}{\delta e(a_1)} = (\frac{e^4}{e^4 + 1}, \frac{e^4}{e^4 + 1})$$

为了快速看出效果，我们把步长 α 设到2，然后使用梯度下降法，这里我们只对$e(a_0)$ $e(a_1)$进行参数更新。因为h_{state}本身的计算太过复杂，所以假设它的参数学习地很好了，不进行调整。新的$e(a_0)$ $e(a_1)$计算如下:

$$e'(a_0) = e(a_0) - \alpha \cdot \frac{\delta loss(\theta)}{\delta e(a_0)} = (\frac{e^4 - 1}{e^4 + 1}, \frac{e^4 - 1}{e^4 + 1})$$

$$e'(a_1) = e(a_1) - \alpha \cdot \frac{\delta loss(\theta)}{\delta e(a_1)} = (\frac{1 - e^4}{e^4 + 1}, \frac{1 - e^4}{e^4 + 1})$$

此时$score'(a_0) = \frac{2(e^4-1)}{e^4+1}$，$score'(a_1) = \frac{2(1-e^4)}{e^4+1}$。$score'(a_0) > score'(a_1)$，如此一来，模型就可以对正确答案打出更高的分数。