BPE

2020年11月14日 21:08

一、为什么要使用Byte-Pair-Encoding这种技术

1.传统的词表无法很好地处理未知和罕见的词汇；

2.传统的Tokenization方法不利于模型学习词缀之间的关系；

e.g.模型学到old，older，oldest之间的关系，但是无法泛化到smart，smarter，smatest

3.character embedding解决OOV问题粒度太细；

4.平衡词粒度和字符粒度之间的gap

二、原理

总的来说就是把一对连续的字节数据替换成一个该数据中不存在的字节，后期使用的时候拿替换表来查询；

e.g.old，older，oldest，我们可以把old替换成其他的一个字节代替变成[rep],[rep]er,[rep]est

具体的实现过程：

1. 准备足够大的训练语料
2. 确定期望的subword词表大小
3. 将单词拆分为字符序列并在末尾添加后缀“ </ w>”，统计单词频率。 本阶段的subword的粒度是字符。 例如，“ low”的频率为5，那么我们将其改写为“ l o w </ w>”：5
4. 统计每一个连续字节对的出现频率，选择最高频者合并成新的subword
5. 重复第4步直到达到第2步设定的subword词表大小或下一个最高频的字节对出现频率为1

每次合并后词表可能出现3种变化：

- +1，表明加入合并后的新字词，同时原来的2个子词还保留（2个字词不是完全同时连续出现）
- +0，表明加入合并后的新字词，同时原来的2个子词中一个保留，一个被消解（一个字词完全随着另一个字词的出现而紧跟着出现）
- -1，表明加入合并后的新字词，同时原来的2个子词都被消解（2个字词同时连续出现）

实际上，随着合并的次数增加，词表大小通常先增加后减小。

例子

输入：

```
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w e s t </w>': 6, 'w i d e s t </w>': 3}
```

Iter 1, 最高频连续字节对"e"和"s"出现了6+3=9次，合并成"es"。输出：

```
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w es t </w>': 6, 'w i d es t </w>': 3}
```

Iter 2, 最高频连续字节对"es"和"t"出现了6+3=9次, 合并成"est"。输出：

```
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w est </w>': 6, 'w i d est </w>': 3}
```

Iter 3, 以此类推，最高频连续字节对为"est"和"</w>" 输出：

```
{'l o w </w>': 5, 'l o w e r </w>': 2, 'n e w est</w>': 6, 'w i d est</w>': 3}
```

.....

Iter n, 继续迭代直到达到预设的subword词表大小或下一个最高频的字节对出现频率为1。