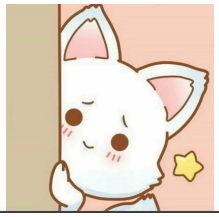
NLP Subword三大算法原理：BPE、WordPiece、ULM

夕小瑶的卖萌屋 2月16日



一只小狐狸带你解锁NLP/ML/DL秘籍

正文作者：Luke

正文来源：https://zhuanlan.zhihu.com/p/86965595

前言

Subword算法如今已经成为了一个重要的NLP模型性能提升方法。自从2018年BERT横空出世横扫NLP界各大排行榜之后，各路预训练语言模型如同雨后春笋般涌现，其中**Subword算法在其中已经成为标配**。且与传统空格分隔**tokenization技术**的对比有很大的优势~~

- 传统词表示方法无法很好的处理未知或罕见的词汇（OOV问题）
- 传统词tokenization方法不利于模型学习词缀之前的关系

  E.g. 模型学到的“old”, “older”, and “oldest”之间的关系无法泛化到“smart”, “smarter”, and “smartest”。
- Character embedding作为OOV的解决方法粒度太细
- Subword粒度在词与字符之间，能够较好的平衡OOV问题

话不多说，和小夕一起来看一下当下**最热最火三个subword算法叭**o(*￣▽￣*)ブ

Byte Pair Encoding

BPE(字节对)编码或二元编码是一种简单的数据压缩形式，其中最常见的一对连续字节数据被替换为该数据中不存在的字节。后期使用时需要一个替换表来重建原始数据。**OpenAI GPT-2 与Facebook RoBERTa均采用此方法构建subword vector.**

- 优点
  - 可以有效地平衡词汇表大小和步数(编码句子所需的token数量)。
- 缺点
  - 基于贪婪和确定的符号替换，不能提供带概率的多个分片结果。

算法

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

停止符"</w>"的意义在于表示subword是词后缀。举例来说："st"字词不加"</w>"可以出现在词首如"st ar"，加了"</w>"表明改字词位于词尾，如"wide st</w>"，二者意义截然不同。

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，同时原来在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。

BPE实现

```
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
```

```
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'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}
num_merges = 1000
for i in range(num_merges):
    pairs = get_stats(vocab)
    if not pairs:
        break
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)

# print output
# ('e', 's')
# ('es', 't')
# ('est', '</w>')
# ('l', 'o')
# ('lo', 'w')
# ('n', 'e')
# ('ne', 'w')
# ('new', 'est</w>')
# ('low', '</w>')
# ('w', 'i')
# ('wi', 'd')
# ('wid', 'est</w>')
# ('low', 'e')
# ('lowe', 'r')
# ('lower', '</w>')
```

编码

在之前的算法中，我们已经得到了subword的词表，对该词表按照子词长度由大到小排序。编码时，对于每个单词，遍历排好序的字词词表寻找是否有token是当前单词的子字符串，如果有，则该token是表示单词的tokens之一。

我们从最长的token迭代到最短的token，尝试将每个单词中的子字符串替换为token。最终，我们将迭代所有tokens，并将所有子字符串替换为tokens。如果仍然有子字符串没被替换但所有token都已迭代完毕，则将剩余的子词替换为特殊token，如<unk>。示例如下~~

```
# 给定单词序列
[“the</w>”, “highest</w>”, “mountain</w>”]

# 假设已有排好序的subword词表
[“errrr</w>”, “tain</w>”, “moun”, “est</w>”, “high”, “the</w>”, “a</w>”]
```

```
# 迭代结果
"the</w>" -> ["the</w>"]
"highest</w>" -> ["high", "est</w>"]
"mountain</w>" -> ["moun", "tain</w>"]
```

编码的计算量很大。在实践中，我们可以pre-tokenize所有单词，并在词典中保存单词tokenize的方式。如果我们看到字典中不存在的未知单词。我们应用上述编码方法对单词进行tokenize，然后将新单词的tokenization添加到字典中备用。

解码

将所有的tokens拼在一起，示例如下：

```
# 编码序列
[“the</w>”, “high”, “est</w>”, “moun”, “tain</w>”]

# 解码序列
“the</w> highest</w> mountain</w>”
```

WordPiece

WordPiece算法可以看作是BPE的变种。不同点在于，**WordPiece基于概率生成新的subword而不是下一最高频字节对。**

算法

1. 准备足够大的训练语料
2. 确定期望的subword词表大小
3. 将单词拆分成字符序列
4. 基于第3步数据训练语言模型
5. 从所有可能的subword单元中选择加入语言模型后能最大程度地增加训练数据概率的单元作为新的单元
6. 重复第5步直到达到第2步设定的subword词表大小或概率增量低于某一阈值

Unigram Language Model

ULM是另外一种subword分隔算法，它能够输出带概率的多个子词分段。它引入了一个假设：**所有subword的出现都是独立的，并且subword序列由subword出现概率的乘积产生。**WordPiece和ULM都利用语言模型建立subword词表。

算法

1. 准备足够大的训练语料
2. 确定期望的subword词表大小
3. 给定词序列优化下一个词出现的概率
4. 计算每个subword的损失
5. 基于损失对subword排序并保留前X%。为了避免OOV，建议保留字符级的单元
6. 重复第3至第5步直到达到第2步设定的subword词表大小或第5步的结果不再变化

总结

总结

1. subword可以平衡词汇量和对未知词的覆盖。极端的情况下，我们只能使用26个token（即字符）来表示所有英语单词。一般情况，建议使用16k或32k子词足以取得良好的效果，Facebook RoBERTa甚至建立的多达50k的词表。
2. 对于包括中文在内的许多亚洲语言，单词不能用空格分隔。因此，初始词汇量需要比英语大很多。

参考文献

[1] https://en.wikipedia.org/wiki/Byte_pair_encoding
[2] https://leimao.github.io/blog/Byte-Pair-Encoding/
[3] https://medium.com/@makcedward/how-subword-helps-on-your-nlp-model-83dd1b836f46
[4] Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates