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。

1. 与传统空格分隔tokenization技术的对比

- 传统词表示方法无法很好的处理**OOV**问题，对于一些罕见词，直接就会变成.
- 词表中的低频词/稀疏词在模型训练过程中无法得到充分训练，进而模型不能充分理解这些词的语义
- 一个单词因为不同的形态会产生不同的词，如由"look"衍生出的"looks", "looking", "looked"，显然这些词具有相近的意思，但是在词表中这些词会被当作不同的词处理，一方面增加了训练冗余，另一方面也造成了大词汇量问题。
- 词表可能非常之大。尤其是假如要训多种语言(例如mBERT)，不同语言的词语实在太多，根本无法存下。

一种解决思路是使用字符粒度来表示词表，虽然能够解决OOV问题，但单词被拆分成字符后，一方面丢失了词的语义信息，另一方面，模型输入会变得很长，这使得模型的训练更加复杂难以收敛。

针对上述问题，Subword(子词)模型方法横空出世。它的**划分粒度介于词与字符**之间，比如可以将"looking"划分为"look"和"ing"两个子词，而划分出来的"look"，"ing"又能够用来构造其它词，如"look"和"ed"子词可组成单词"looked"，因而Subword方法能够大大降低词典的大小，同时对相近词能更好地处理。

一般情况，建议使用16k或32k子词足以取得良好的效果，FacebookRoBERTa甚至建立的多达50k的词表。

2. Byte Pair Encoding (Sennrich et al., 2015)

BPE最早是一种数据压缩算法，由Sennrich等人于2015年引入到NLP领域并很快得到推广。该算法简单有效，因而目前它是最流行的方法。**GPT-2、RoBERTa、XLM使用的Subword算法都是BPE。**

BPE获得Subword的步骤如下：

1. 准备足够大的训练语料，并确定期望的Subword词表大小；
2. 将单词拆分为成最小单元。比如英文中26个字母加上各种符号，这些作为初始词表；
3. 在语料上统计单词内**相邻单元对的频数**，选取频数最高的单元对合并成新的Subword单元；
4. 重复第3步直到达到第1步设定的Subword词表大小或下一个最高频数为1.

下面以一个例子来说明:

假设有语料集经过统计后表示为{'low':5,'lower':2,'newest':6,'widest':3}，其中数字代表的是对应单词在语料中的频数。

1) 拆分单词成最小单元，并初始化词表。这里的最小单元即为字符。因而，可得到：

语料	词表
'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	'l','o','w', 'e','r','n', 's','t','i','d','</w>'

需要注意的是，在将单词拆分成最小单元时，要在单词序列后加上""(具体实现上也可以使用其它符号)来表示中止符。在subword解码时，中止符可以区分单词边界。

2) 在语料上统计相邻单元的频数。这里，最高频连续子词对"e"和"s"出现了6+3=9次，将其合并成"es"，有

语料	词表
'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	'l','o','w','e','r','n', '**s**','t','i','d','</w>','**es**'

由于语料中不存在's'子词了，因此将其从词表中删除。同时加入新的子词'es'。一增一减，词表大小保持不变。

3) 继续统计相邻子词的频数。此时，最高频连续子词对"es"和"t"出现了6+3=9次, 将其合并成"est"，有

语料	词表
'l o w </w>':5 'l o w e r </w>':2 'n e w est </w>':6 'w i d est </w>':3	'l','o','w','e','r','n', 't','i','d','</w>','es','est'

4) 接着，最高频连续子词对为"est"和""，有

语料	词表
'l o w </w>':5 'l o w e r </w>':2 'n e w est</w>':6 'w i d est</w>':3	'l','o','w','e','r','n', 'i','d','est','est</w>'

5) 继续上述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。

从上面的示例可以知道，每次合并后词表大小可能出现3种变化：

- +1，表明加入合并后的新子词，同时原来的2个子词还保留（2个字词分开出现在语料中）。
- +0，表明加入合并后的新子词，同时原来的2个子词中一个保留，一个被消解（一个子词完全随着另一个子词的出现而紧跟着出现）。
- -1，表明加入合并后的新子词，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。

在得到Subword词表后，针对每一个单词，我们可以采用如下的方式来进行编码：

1. 将词典中的所有subword按照长度由大到小进行排序；
2. 对于单词w，依次遍历排好序的词典。查看当前subword是否是该单词的子字符串，如果是，则输出当前子词，并对剩余单词字符串继续匹配。
3. 如果遍历完字典后，仍然有子字符串没有匹配，则将剩余字符串替换为特殊符号输出，如""。
4. 单词的表示即为上述所有输出子词。

例子：

```
# 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>"]
```

解码：

如果相邻subword间没有中止符，则将两subword直接拼接，否则两subword之间添加分隔符。

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

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

在Huggingface Transformer中，已经封装好了常见transformer模型的tokenizer，例如：

```
from transformers import XLMTokenizer
tokenizer = XLMTokenizer.from_pretrained('xlm-mlm-tlm-xnli15-1024')
tokens = tokenizer.tokenize("Hello, my dog is cute")
print(tokens)
### ['hello</w>', ',</w>', 'my</w>', 'dog</w>', 'is</w>', 'cute</w>']
```

3. WordPiece (Schuster et al., 2012)

Google的**Bert**模型在分词的时候使用的是WordPiece算法。与BPE算法类似，WordPiece算法也是每次从词表中选出两个subword合并成新的subword。与BPE的最大区别在于，如何选择两个subword进行合并：BPE选择频数最高的相邻subword合并，而WordPiece选择能够提升语言模型概率最大的相邻subword加入词表。

看到这里，你可能不清楚WordPiece是如何选取子词的。这里，通过形式化方法，能够清楚地理解WordPiece在合并这一步是如何作出选择的。假设句子 $S=(t_1,t_2,\dots,t_n)$ 由n个子词组成，t_i 表示子词，且假设各个子词之间是独立存在的，则句子 S 的语言模型似然值等价于所有子词概率的乘积，取log后变成加和：

$$logP(S)=\sum_{i=1}^{n}logP(t_i)$$

假设把相邻位置的 x 和 y 两个子词进行合并，合并后产生的子词记为 zz，此时句子 SS 似然值的变化可表示为：

$$logP(t_z)-(logP(t_x)+logP(t_y))=log(\frac{P(t_z)}{P(t_x)P(t_y)})$$

从上面的公式发现，似然值的变化就是**两个subword之间的互信息。**简而言之，WordPiece每次选择合并的两个subword，他们具有最大的互信息值，也就是两子词在语言模型上具有较强的关联性，它们经常在语料中以相邻方式同时出现。

```
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
tokens = tokenizer.tokenize("oooops, this is a missspelling")
print(tokens)
##['o', '##oo', '##ops', ',', 'this', 'is', 'a', 'miss', '##sp', '##elling']
```

另外，n-gram在现在也经常使用。例如淘宝2021年的文章Embedding-based Product Retrieval in Taobao Search 中提到的Multi-Granular Semantic Unit，其实就是利用中文的1-gram，2-gram和分词之后的结果，来把握不同粒度的语义。
