# 2.1

(1) WordPiece 算法是 tokenization 的一种方法，其核心流程可分为以下三个主要阶段：

1. **Token 化序列**
   (a) 首先，选定一种语种作为训练语料库，本次例子使用英语。
   (b) 接下来分词方面，WordPiece 要求明确每个子单元的序列关系。即以英语单词"peking" 为例，需分割为：

```
p ##e ##k ##i ##n ##g
```

   (c) 因此，具体步骤为：
      i. 对英语等以空格分隔的语言，先按空格切分单词；
      ii. 对每个单词进一步分割：首字符保留原形式，后续字符添加 ## 前缀。
2. **学习合并规则**
   (a) WordPiece 的合并规则基于以下评分函数：

$$\text{score} = \frac{\text{freq\_of\_pair}}{\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element}}$$

   (b) 该公式表示：两个单元合并出现的频率，除以它们各自独立出现的频率的乘积。
   (c) 分数越高，表明该组合在语料中共现频率越高，且各自独立频率相对较低。算法会优先合并词汇表中单独出现频率较低但共同出现频率高的单元对。
   (d) 示例：

```
("hug", 10), ("pug", 5), ("buns", 12)
```

   (e) 分割后的形式为：

```
("h", "##u", "##g", 10),
("p", "##u", "##g", 5),
("b", "##u", "##n", "##s", 12)
```

   (f) 合并决策说明：

$$\text{score of hu} = \frac{10}{10 \times 27} = \frac{1}{27} \quad \text{score of ug} = \frac{15}{15 \times 27} = \frac{1}{27},$$

$$\text{score of pu} = \frac{5}{5 \times 27} = \frac{1}{27}, \quad \text{score of bu} = \frac{12}{12 \times 27} = \frac{1}{27}$$
$$\text{score of un} = \frac{12}{27 \times 12} = \frac{1}{27}, \quad \text{score of ns} = \frac{12}{12 \times 12} = \frac{1}{12}$$

(g) 按照以上计算过程，如果是一般按照频率选择的算法如 Bpe 将会选择合并 $\#\#ug$，而 Wordpiece 倾向于选择 $\#\#ns$。这样的挑选机制可能是为了把常常出现在一起的配对起来，如英文里的冠词、介词、专有名词等。

(h) 按照网上的资料来源，这样的表示方法也可以用信息熵进行解释，即设对于每一个配对 **pq**，p 的信息熵可以表示为 $-log\frac{\text{freq\_p}}{total}$，对 **q** 和 **pq** 也是同理。则信息差可以表示为分别单独存在时带来的信息减去合并后的信息：

$$\begin{aligned}
&(-log\frac{\text{freq_p}}{total} + -log\frac{\text{freq_q}}{total}) - (-log\frac{\text{freq_pq}}{total}) \\
&= -(log\frac{\text{freq_p} \cdot \text{freq_q}}{total^2}) + log\frac{\text{freq_pq}}{total} \\
&\approx log\frac{\text{freq_pq}}{\text{freq_p} \cdot \text{freq_q}} \\
&\approx \frac{\text{freq_pq}}{\text{freq_p} \cdot \text{freq_q}}
\end{aligned}$$

由于 total 为常数和 log 为单增函数，因此不影响计算结果的意义，所以这样可以得到同样的公式。

3. **合并操作**：

(a) 合并前，如果前一个单元是带前缀的，合并后将再次添加。例如合并 ("$\#\#n$", " $\#\#s$") 后，新 token "$\#\#ns$" 将被加入词汇表。

(b) 随后，这个合并规则将在语料库遍历查找进行合并

```
("h", "##u", "##g", 10),
("p", "##u", "##g", 5),
("b", "##u", "##ns", 12)
```

**算法设计-实现 WordPiece**

1. 首先，需要将语料库预分词为单词，即对于 training corpus 里的每一句话（text）有以下处理

```
word_freqs = defaultdict(int)
for text in corpus:
    words_with_offsets = tokenizer.backend_tokenizer.
       pre_tokenizer.pre_tokenize_str(text)
    new_words = [word for word, offset in
       words_with_offsets]
    for word in new_words:
        word_freqs[word] += 1
```

以上代码为使用对应模型库封装的 tokenizer 如 Bert-base-cased，进行对每一句 text 以空格为分隔符的分离，得到相应的结果为单词序列。接着，对每个单词统计词频，生成字典序列，每个字典元素为 {单词：词频}。

2. 接着，需要构造词汇表，即对每个单词的字母按照序列规则进行相应的处理。如果目前遍历的字母不是开头字母，在加入词汇表前要加上前缀 ##，即以下代码内容：

```
alphabet = []
for word in word_freqs.keys():
    if word[0] not in alphabet:
        alphabet.append(word[0])
    for letter in word[1:]:
        if f"##{letter}" not in alphabet:
            alphabet.append(f"##{letter}")
```

3. 接下来，就能进行对应的合并操作，过程为先按照原本 text 顺序，在字母级别上进行两两配对计算频率比，代码为

```
scores = {
    pair: freq / (letter_freqs[pair[0]] * letter_freqs[
       pair[1]])
    for pair, freq in pair_freqs.items()
```

```
}
```

- `freq` 为配对的频率（出现次数）；
- `letter_freqs[pair[0]]` 为配对中第一个字母的频率；
- `letter_freqs[pair[1]]` 为配对中第二个字母的频率；
- `for pair, freq in pair_freqs.items()` 遍历所有字母配对及其频率。

4. 以下为计算结果展示

```
('p', '##e'): 0.045454545454545456
('##e', '##k'): 0.045454545454545456
('##k', '##i'): 0.05555555555555555
('##i', '##n'): 0.013888888888888888
('##n', '##g'): 0.05
```

5. 接下来，找出得分最高的配对作为合并目标，然后把这个结果更新到词汇表

```
while i < len(split) - 1:
    if split[i] == a and split[i + 1] == b:
        merge = a + b[2:] if b.startswith("##") else a +
            b
        split = split[:i] + [merge] + split[i + 2 :]
```

6. 最后，按照预设的词汇表大小循环完成词汇表的生成。

(2) 代码已完成在附件 *wpalg.py*

(3) Tokenization 的结果为

```
Tokenization result: ['n', '##o', '##u', '##s', 'e', '##tud', '##i',
   '##o', '##n', '##s', 'a', 'l', 'univ', '##e', '##rsi', '##t', '##e',
    'd', '##e', 'p', '##e', '##ki', '##n']
```

(4) 回答问题

(a) 使用语句为**'i love go to zoo, . ?'**

```
Tokenization result: ['i', 'lo', '##v', '##e', '[UNK]', 't', '##o',
    '[UNK]', '[UNK]', '[UNK]', '[UNK]']
```

(b) Llama 的 tokenizer 不需要 [UNK] 的原因为 llama 的 tokenizer 是基于 Bpe 算法的，Bpe 会将所有的输入先转为 Unicode 编码，再基于编码数进行合并。因此，在对训练集里从未出现的输入进行转换时，不需要 [UNK] 进行标注，token 将使用一个抽象的编码直接表示，这也是导致 Bpe 可能对非英语语言表现较差的原因之一，因为表达在不同的语言表达同义词时，英语的 tokens 数总是小于其他语言的。

## 2.2

1. (a) 训练方法：
   - 首先，使用 Hugging Face 提供的 `models.WordPiece` 初始化一个 WordPiece 分词器。
   - 接着，调用 `pre_tokenizer.Whitespace()`，将输入文本按空格进行初步切分。
   - 最后，使用 `trainers.WordPieceTrainer` 定义训练参数并调用 `tokenizer.train_from_iterator(corpus, trainer)`，传入语料数据 `corpus` 和训练器 `trainer` 进行训练。
   - 代码存放在 `tokenizer.py`

   (b) 训练参数：
   - `vocab_size = 50000`
   - `special_tokens = ["[UNK]", "[CLS]", "[SEP]", "[PAD]", "[MASK]"]`

   (c) 最后的结果存在 `biometical_tokenizer_50000.json`，词汇表大小为 50000。

2. 挑选 token 的方法为

```python
numbers = {'0','1','2','3','4','5','6','7','8','9'}
unique_bio_tokens = [
    token for token in bio_vocab
```

```
    if (token not in bert_vocab) and
        (not token.startswith('#')) and
        (not token[-1] in numbers) and
        (len(token) >= 6)
]
```

- 选择本身没有出现在原本 bert 词汇表里的 token
- 选择完整的词，不以 ## 为开头的
- 选择最后一个不为数字的 token
- 选择长度大于 6 的 token

以下为 50 个 tokens：

```
['Claude', 'Doctors', 'neurotrophin', 'Phenol', 'Carbone', 'Burrows', '
   Southampton', 'Detected', 'Compression', 'Marina', 'ijrobp', 'Trolox
   ', 'paralogous', 'microdiss', 'thiols', 'Lisboa', 'Originally', '
   Abbasi', 'Hayden', 'boxplots', 'Organomet', 'polymerized', '
   mesophyll', 'Signatures', 'Laboratorio', 'Intermittent', 'Inspection
   ', 'decarb', 'Albuquerque', 'Ficoll', 'recalcitr', 'gravit', '
   energetics', 'Embryo', 'Amplitude', 'Searching', 'Polyethylene', '
   confounder', 'Equilibrium', 'Advancement', 'workstation', '
   involution', 'Witten', 'Tsuchiya', 'discoloration', 'Lemesh', '
   Xiamen', 'transfections', 'lymphoproliferative', 'plasmacytoid']
```

这样选择的词大多为生物，化学领域的专业单词，适合用于在相关领域上做预测任务。

3. 句子一

Interestingly , the restoration of contact inhibition in HB-19 treated G401 cells is concomitant with marked reduction of transcripts coding the Wilms ' tumor 1 gene , matrix metalloproteinase-2 , epithelial isoform of CD44 , and vascular endothelial growth factor , whereas no apparent modification is detected for transcripts coding the proto-oncogene c-Myc , anti-apoptotic Bcl-2 , pro-apoptotic Bax , tissue inhibitor of metalloproteinase TIMP-1 , angiogenesis inhibitor TSP-1 , and growth factor Midkine .

原始 Bert：

```
interesting ##ly , the restoration of contact inhibition in h ##b - 19
   treated g ##40 ##1 cells is con ##com ##itan ##t with marked
   reduction of transcript ##s coding the wil ##ms ' tumor 1 gene ,
   matrix metal ##lo ##pro ##tein ##ase - 2 , ep ##ith ##elial iso ##
   form of cd ##44 , and vascular end ##oth ##elial growth factor ,
   whereas no apparent modification is detected for transcript ##s
   coding the proto - on ##co ##gen ##e c - my ##c , anti - ap ##op ##
   to ##tic bc ##l - 2 , pro - ap ##op ##to ##tic ba ##x , tissue
   inhibitor of metal ##lo ##pro ##tein ##ase tim ##p - 1 , ang ##io ##
   genesis inhibitor ts ##p - 1 , and growth factor mid ##kin ##e .
```

扩展 Bert：

```
interestingly , the restoration of contact inhibition in h ##b - 19
   treated g ##40 ##1 cells is concomitant with marked reduct ion of
   transcripts coding the wil ##ms ' tumor 1 gene , matrix metalloprote
    ina ##se - 2 , epitheli al iso ##form of cd ##44 , and vascular
   endoth eli ##al growth factor , whereas no apparent mod ##if ##i
   cation is detected for transcripts coding the proto - on ##co ##gen
   ##e c - my ##c , anti - apoptotic bc ##l - 2 , pro - apoptotic ba ##
   x , tissue inhibitor of metalloprote ina ##se tim ##p - 1 , ang ##io
    ##genesis inhibitor ts ##p - 1 , and growth factor mid ##kin ##e .
```

句子二

The cellobiose- and cellulose-responsive induction of the FIII-avicelase ( cbhI ) , FII-carboxymethyl cellulase ( cmc2 ) , and FIa-xylanase ( xynIa ) genes is not regulated by XlnR in Aspergillus aculeatus , which suggests that this fungus possesses an unknown cellulase gene-activating pathway .

原始 Bert：

```
the cello ##bio ##se - and cell ##ulo ##se - responsive induction of
   the fi ##ii - av ##ice ##lase ( cb ##hi ) , fi ##i - car ##box ##yme
    ##thy ##l cell ##ula ##se ( cm ##c ##2 ) , and fia - x ##yla ##nas
   ##e ( x ##yn ##ia ) genes is not regulated by xl ##nr in as ##per ##
   gill ##us ac ##ule ##atus , which suggests that this fungus
   possesses an unknown cell ##ula ##se gene - act ##ivating pathway .
```

扩展 Bert：

```
the cello ##bio ##se - and cell ##ulo ##se - responsive induction of
   the fi ##ii - av ##ice ##lase ( cb ##hi ) , fi ##i - carboxymethyl
   cellulase ( cm ##c ##2 ) , and fia - xylanase ( x ##yn ##ia ) genes
   is not regulated by xl ##nr in asperg ill ##us ac ##ule ##atus ,
   which suggests that this fungus possesses an unknown cellulase gene
   - activating pathway .
```

句子三

Compound C significantly inhibited the receptor tyrosine phosphorylation and the activity of downstream signaling molecules , such as p85 phosphoinositide 3-kinase , phospholipase C- 1 , and extracellular signal-regulated kinase 1/2 , induced by platelet-derived growth factor ( PDGF ) but not by epidermal growth factor- and insulin-like growth factor .

原始 Bert：

```
compound c significantly inhibit ##ed the receptor ty ##ros ##ine ph ##
   os ##ph ##ory ##lation and the activity of downstream signaling
   molecules , such as p ##85 ph ##os ##ph ##oin ##osi ##ti ##de 3 -
```

```
kinase , ph ##os ##ph ##oli ##pas ##e c -  ##1 , and extra ##
cellular signal - regulated kinase 1 / 2 , induced by plate ##let -
derived growth factor ( pd ##gf ) but not by ep ##ider ##mal growth
factor - and insulin - like growth factor .
```

扩展 Bert：

```
compound c significantly inhibit ##ed the receptor tyrosine phospho ry
##lation and the activity of down stream signaling molecules , such
as p ##85 phosphoinositide 3 - kinase , phospho lip ##ase c -  ##1
, and extrac el ##lu ##lar signal - regulated kinase 1 / 2 , induced
 by plate ##let - derived growth factor ( pd ##gf ) but not by
epidermal growth factor - and insulin - like growth factor .
```

差异总结：

| 句子 | 原始 BERT 长度 | 扩展 BERT 长度 | 新增合并 token |
|---|---|---|---|
| 一 | 135 | 122 | cation, ##if, transcripts, concomitant, ##se, ##i, interestingly, endoth, eli, ##al, ion, mod, al, epitheli, metalloprote, reduct, apoptotic, ina |
| 二 | 87 | 74 | ill, activating, carboxymethyl, cellulase, asperg, xylanase |
| 三 | 88 | 76 | ##ase, el, tyrosine, down, epidermal, ##lar, lip, phospho, ##lu, stream, extrac, phosphoinositide, ry |

- 原始 Bert token 平均长度 = 67.40
- 扩展 Bert token 平均长度 = 65.86

4. (1)

```
原始BERT词汇量: 30522
扩展BERT词汇量: 35522
BERT嵌入维度 = 768
```

因此，新增参数为

$$(35522 - 30522) \times 768 = 3,840,000$$

(2) 初始化代码为

```
with torch.no_grad():
    mean_embedding = model.embeddings.word_embeddings.
        weight[:len(bert_vocab)].mean(dim=0)
    model.embeddings.word_embeddings.weight[len(
        bert_vocab):] = mean_embedding
```

以上代码为使用了原始词汇表所有 token 嵌入的均值初始化。使用均值向量能保留原始词汇的总体分布特征，避免新 token 的嵌入偏离过远。

5. 使用的代码在 `Bert.py`，也保存了 `expanded_bert_tokenizer`
对比结果：

| **tokenizer 类型** | **Accuracy** | **F1-Macro** | **F1-Micro** |
|---|---|---|---|
| 原始词汇表 | 0.7738 | 0.7662 | 0.7738 |
| 扩展词汇表（不做任何筛选，正 5000） | 0.6167 | 0.4213 | 0.6167 |
| 扩展词汇表（不做任何筛选，倒 5000） | 0.6738 | 0.5249 | 0.6738 |
| 扩展词汇表 (仅使用不带有 ## 为开头的 token，正 5000) | 0.5428 | 0.4077 | 0.5429 |
| 扩展词汇表 (仅使用不带有 ## 为开头的 token，倒 5000) | 0.7381 | 0.6406 | 0.7381 |
| 扩展词汇表 (仅使用带有 ## 为开头的 token，正 5000) | 0.7786 | 0.7733 | 0.7786 |
| 扩展词汇表 (仅使用带有 ## 为开头的 token，倒 5000) | 0.7762 | 0.7252 | 0.7762 |

结论：

- 实验使用的是 hugging face 提供的包，但是生成后的 token 似乎进行了另外的排序，导致没有了 WordPiece 合并多频被优先选择的特点，因此实验只能以正 5000，倒 5000 来取
- 其他三项过滤是必须的，不做任何过滤是不行的，会混入大量的无用 token
- 使用扩展词汇表并没有比原始模型带来显著的提升，推测可能是 Hoc 训练集太小且与验证集有一些差距
- 如果要使用不带 ## 为开头的扩展词汇表，取倒 5000 个 token 效果会更好，正 5000 个的 token 大量为通用词，并非专用词
- 使用不带 ## 为开头的扩展词汇表使得 F1 Macro 下降，表明模型对少数类别的分类能力变差，可能是加入的完整单词使得混入低频或无领域相关性的子词。
- 如果使用带 ## 为开头的扩展词汇表表现都与原始词汇表相同