基于BPE的汉语tokenization

刘帅 2020212267 2020219111班

一、实验环境

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              48
On-line CPU(s) list: 0-47
Thread(s) per core:  2
Core(s) per socket:  24
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               85
Model name:          Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz
Stepping:            4
CPU MHz:             2499.998
BogoMIPS:            4999.99
Hypervisor vendor:   KVM
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            1024K
L3 cache:            33792K
NUMA node0 CPU(s):   0-47
```

本次实验在阿里云第四代ECS48核云服务器进行实验，型号为Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz。

二、算法代码：

2.1 导入相关模块&初始化

```
import re
import collections
import logging
import tqdm
logging.basicConfig(filename='mylog.log', level=logging.INFO)
with open('mylog.log', 'w'): #对之前的log先清空
    pass
```

其中，re模块提供了正则表达式操作的支持，可以使用它来对字符串进行模式匹配、查找、替换等操作。

collections模块提供了额外数据类型，便于后续建立defaultdict

logging为日志模块，用于在训练过程中进行log记录，从而进行时间、训练效率监控等。

tqdm记录里训练过程的iter次数及cpu处理iter的时间，从而起到可视化监控训练的目的。

2.2 统计子词对应的词频

```
def get_subwords(data):
    #统计子词以及对应的词频
    subwords = collections.defaultdict(int)
    for word, freq in data.items():
        for subword in word.split():
            subwords[subword] += freq
    return subwords
```

针对data中的每个词，将其分割成subword并将对应词频存入字典，从而返回子词及其对应频率的字典。

2.3 前处理

2.3.1 训练数据初始化(预处理)

```
def init_traindata(path):
    with open(path,'r')as f:
        lines=f.readlines()
        # for i in range(1000):
        #     lines.append(f.readline())
    train_data={}
    for data in lines: #针对每行句子
        for i in re.split(r"[,。、《》：“”()\[\]【】]", data): #提取出该行的子句
            if i not in train_data.keys():
                train_data[i]=0
            else:
                train_data[i]+=1
    return train_data
```

首先通过readlines以列表形式读入训练数据并存入lines，而后针对lines中的每行句子，利用正则表达式进行处理，将该行中包含,。、《》：“”()[]【】的字符进行删除，并且将最终结果存入字典，表示该行在整个数据集存在的频次。

2.3.2 验证数据初始化

```
def get_testdata(path):
    with open(path,'r')as f:
        lines=f.readlines()
        # lines=[f.readline()]
        # for i in range(10):
        #     lines.append(f.readline())
    results=[]
    for data in lines:
        results.append(data.replace(' ',''))
    return results
```

与之对应的则为验证数据的初始化。针对验证数据，我们将标点符号进行保留，因为在生成子词的整个过程中，不涉及标点符号作为子词的构建，因此在test处理时我们只需将每个汉字之间的空格取消。

```
trainpath="train_BPE.txt" #数据集路径
testpath="test_BPE.txt"

train_data=init_traindata(trainpath)

subwords = get_subwords(train_data)
# 获取初始化的子词词表
bpe_vocab = set(subwords.keys())
print("初始词表个数为",len(bpe_vocab))
```

指定数据集路径，并对训练集进行预处理，构建最初子词表。

2.4 获取子词集合并

```
def get_pair_with_frequency(data):
    """
    获取子词对以及子词集合
    """
    pairs = collections.defaultdict(int)
    for word, freq in data.items():
        sub_words = word.split()
        for i in range(len(sub_words)-1):
            pair = (sub_words[i],sub_words[i+1])#每两个相邻的词组成pair
            pairs[pair] += freq#结果存入字典
    return pairs
```

获取相邻的两子词组成pair，并将结果存于字典。

2.5 合并最高频子词

```
def merge_data_with_pair(pair, data):
    """
    将语料中的最高频子词对进行合并
    输入:
        - pair: 最高频子词词对
        - data: 字典形式，统计好的输入语料
    """
    result = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)') #pair中的两元素进行合并 例如("社
会","主义"),并且两者前后不能有其他穿插（保证连续）
    for word in data:
        merged_word = p.sub(''.join(pair), word)#将word中匹配到的子字符串替换为pair中的两
个元素合并后的字符串
        result[merged_word] = data[word]
    return result
```

后面传入pair为最高频子词

2.6 构建词表

```
def build_vocab(train_data, num_merges,size_bpe):

    # 初始化词表
    subwords = get_subwords(train_data)
    bpe_vocab = set(subwords.keys())
    # print(bpe_vocab, len(bpe_vocab))
    i = 1
    # 逐步生成词表
    for _ in tqdm.tqdm(range(num_merges),mininterval=10): #每十个轮次进行一次可视化
        # 根据语料统计相邻子词对的词频
        pairs = get_pair_with_frequency(train_data)
        # 取频率最大的子词对，如果pairs 为空或子词对的最大频次为1，则停止
        if not pairs: #如果pairs为空
            break
        best_pair = max(pairs, key=pairs.get)
        if pairs[best_pair] == 1:
            break
        # 合并语料
        train_data = merge_data_with_pair(best_pair, train_data)
        # 将子词加入词表中
        merged_word = "".join(best_pair)
        bpe_vocab.add(merged_word)
        # 删除子词
        subwords = get_subwords(train_data)
        if best_pair[0] not in subwords:
            bpe_vocab.remove(best_pair[0])
        if best_pair[1] not in subwords:#若两词相同，第一次删除后在subwords中则不存在对应的
子词
            if best_pair[1] in bpe_vocab: #防止例如(瑟、瑟)的pair
                bpe_vocab.remove(best_pair[1])
        i += 1
        if i%10==0:
            tqdm.tqdm.write(f"第{i}个iter：当前词表词数:{len(bpe_vocab)}")
            tqdm.tqdm.write(f"第{i}个iter：最高频子词对{best_pair}")
            logging.info(f"第{i}个iter：当前词表词数:{len(bpe_vocab)}")#在日志中进行记录
            logging.info(f"第{i}个iter：最高频子词对{best_pair}")
        if(len(bpe_vocab)==size_bpe): #设置最终停止频次，词典大小为10000时停止
            print("finish!")
            break
    return bpe_vocab
```

该代码为训练的核心代码，循环的进行best_pair的寻找，并针对子词表进行更新，从而不断将新词添加至词表之中。

2.7 后处理

```
logging.shutdown()
bpe_vocab = build_vocab(train_data, num_merges,size_bpe=10000)
temp=bpe_vocab#记录出真正属于中文词汇的bpe表格，而不是将标点等特殊符号算入
bpe_vocab.add(',')
bpe_vocab.add('。')
bpe_vocab.add('、')
bpe_vocab.add('《')
bpe_vocab.add('》')
bpe_vocab.add('：')
bpe_vocab.add('“')
bpe_vocab.add('”')
bpe_vocab.add('(')
bpe_vocab.add(')')
bpe_vocab.add('[')
bpe_vocab.add(']')
bpe_vocab.add('【')
bpe_vocab.add('】')
bpe_vocab.add('\n')
# print("词表: ", bpe_vocab)
```

首先对log进行缓存释放。之后，将构建好的子词表bpe_vocab存入temp文件中，便于后续写入txt文件进行词典保存。

而后，我们针对bpe_vocab文件进行中文中存在的特殊标点进行添加。由于后续的操作环节均在验证集当中进行，我们此时添加标点的目的是为了能让子词表在测试语料中匹配到单独的标点符号。

同时，这样的额外添加是基本合理的。首先，针对中文分词，我们更应关注的是汉字间关系，汉字与标点符号之间并不存在明显的子词绑定关系'其次，针对标点符号，很少有两个标点符号连续出现的情况，因此我们可以将所有标点符号作为单独的子词进行考虑，从而避免训练过程中出现**文字+标点**子词组合的错误模式。

2.8 子词匹配算法

```
def tokenize_word(word, sorted_vocab, unknown_token='<unk>'):
    """
    输入:
        - word: 待编码的单词
        - sorted_vocab: 排序后的子词词典
        - unknown_token: 不能被切分的子词替代符
    """
    # 如果传入的词为空
    if word == "":
        return []
    # 如果词表为空，则将输入的词替换为<UNK>
    if sorted_vocab == []:
        return [unknown_token]

    word_tokens = []
    # 遍历词表拆分单词
    for i in range(len(sorted_vocab)):
```

```
        token = sorted_vocab[i] #从词表中选取第i个词
        # 基于该token定义正则，同时将token里面包含句号的变成[.]
        token_reg = re.escape(token.replace('.', '[.]'))
        # 在当前word中进行遍历，找到匹配的token的起始和结束位置
        matched_positions = [(m.start(0), m.end(0)) for m in re.finditer(token_reg,
word)]
        # 如果当前token没有匹配到相应串，则跳过
        if  len(matched_positions) == 0:
            continue

        # 获取匹配到的子串的起始位置
        end_positions = [matched_position[0] for matched_position in
matched_positions]
        start_position = 0

        for end_position in end_positions:
            subword = word[start_position: end_position]
            word_tokens += tokenize_word(subword, sorted_vocab[i+1:], unknown_token)
            word_tokens += [token]
            start_position = end_position + len(token)
        # 匹配剩余的子串
        word_tokens += tokenize_word(word[start_position:], sorted_vocab[i+1:],
unknown_token)
        break
    else:
        # 如果word没有被匹配，则映射为<unk>
        word_tokens = [unknown_token] * len(word)

    return word_tokens
```

利用贪心算法，对最长的子词串从左至右进行最长匹配。若在子词表中找不到对应匹配，则映射为。

2.9 对子词表排序并利用2.8中贪心算法

```
def tokenize(text, bpe_vocab):

    sorted_vocab = sorted(bpe_vocab, key=lambda subword: len(subword), reverse=True)
#根据子词长度进行sort
    # print("待编码语句: ", text)
    tokens = []
    for word in text:
        # word=word.replace(' ','')
        # word = word + "</w>"
        word_tokens = tokenize_word(word, sorted_vocab, unknown_token='<unk>')
        tokens.extend(word_tokens)

    return tokens
```

该方法的主要功能为对子词进行排序，并利用tokenize_word方法进行贪心匹配。从而在验证集中匹配到最长子词。

2.10 验证

```
test_data=get_testdata(testpath)#验证集初始化
tokens = tokenize(test_data, bpe_vocab) #验证集推理
```

2.11 后处理，文件保存

```
with open('test_result.txt', 'w') as file:
    pass#清空文件
for i in tokens:
    with open('test_result.txt','a+')as f:
        if(i=='\n'):
            f.write('\n')
        else:
            f.write(i)
            f.write(' ')
with open('bpe_vocab.txt','w',encoding='utf-8')as f:
    for i in temp:
        f.write(i)
        f.write('\n')
```

由于tokens中存入了验证集推理后的聚类结果，我们只需要将tokens表中的内容进行concat，将验证结果写入test_result.txt文件中，同时以\n作为写入时换行的标志。

同时将temp文件保存至bpe_vocab.txt构建字词表

三、构建思路及分析

由于demo中给出的样例为英文的分割结果，作为中文而言，与英文有如下几点不同：

1、中文无需这样的token表示一个单词的两个子词（例如lear和ning），针对中文而言，中文的子词即一个或几个独立的中文字组合而成的词，单独的一个中文字也具有含义。

2、针对中文之中的标点符号，中文的标点符号比英文样式更丰富，同时为了突出中文词表的特性，在词表中应避免出现单独的标点符号或者符号与中文相连的情况。因此，我们在构建训练数据集时**首先将所有标点符号通过正则表达式进行删除，在构建完字词表过后，再将标点符号回填**，以便于和验证集进行匹配。

3、英文中少有出现例如e和ee两个子词同时出现的情况，但在中文可能包含，例如“天”和“天天”，因此我们在子词表删除时需要注意在set中已经将单字删除的情况，需要进行额外判断。