怎么让英文大语言模型支持中文？（一）—— 构建中文tokenization

来自： AiGC面试宝典

宁静致远　　2023年09月29日 12:24

扫码
查看更

一、为什么需要 构建中文tokenization?

大语言模型 百家争鸣趋势，但是 从 根本来看，主要 以 基于llama家族的模型 和 基于glm家族的模型为主。不过目前大多数 LLMs 还是以 基于llama家族的模型 为主，然而 原始基于llama家族的模型 主要训练语料为 英文语料，中文语料占比较少，直接导致，基于llama家族的模型 对于 中文的支持不太友好。

那么有什么办法 能够 解决 基于llama家族的模型 对于 中文的支持不太友好 问题呢?

本文利用 《斗破苍穹》作为语料，介绍 从 扩充vocab里面的词以对中文进行token化。

二、如何对 原始数据预处理?

《斗破苍穹》 原始数据

```
    《斗破苍穹》来自:



===上架感言===

又一次上架了，这次比上次还激动，甚至激动到了上传了章节却不知道发出来的地步。

    尴尬，关于新书，上架前成绩好得有些出乎土豆的意料，对于这份厚硕的成绩，土豆心里还真有几分惶恐与忐忑，虽说曾经有人说土豆是刷出来的数据，对于这些留言，我也并未太过在意，别的我不知道，我唯一能知道的，就是人在做，天在看！

    究竟刷没刷，自己心中有杆秤就能衡量，问心无愧，何惧留言？

    呵呵，在这里很感谢赐予土豆这种厚硕成绩的诸位书友，真的，很感谢你们。
    ...
```

上文摘取 部分 《斗破苍穹》 原始数据，从 数据中可以看出，数据中包含 大量换行 和 无效内容，所以需要对 《斗破苍穹》 原始数据 进行预处理，将每一行转化为一句或多句话，同时过滤掉 换行 和 无效内容。

代码讲解

```
# step 1: 《斗破苍穹》 原始数据 加载
with open("data/《斗破苍穹》.txt", "r", encoding="utf-8") as fp:
    data = fp.read().strip().split("\n")

# step 2: 将每一行转化为一句或多句话，同时过滤掉 换行 和 无效内容
sentences = []
for d in data:
    d = d.strip()
    if "===" in d or len(d) == 0 or d == "《斗破苍穹》来自:":
        continue
    sentences.append(d)
```

```
# step 3: 数据写入
with open("data/corpus.txt", "w", encoding="utf-8") as fp:
    fp.write("\n".join(sentences))
```

预处理之后的 corpus.txt

```
    又一次上架了，这次比上次还激动，甚至激动到了上传了章节却不知道发出来的地步。
    尴尬，关于新书，上架前成绩好得有些出乎土豆的意料，对于这份厚硕的成绩，土豆心里
还真有几分惶恐与忐忑，虽说曾经有人说土豆是刷出来的数据，对于这些留言，我也并未太过
在意，别的我不知道，我唯一能知道的，就是人在做，天在看！
    ...
```

三、如何构建中文的词库?

得到语料库 corpus.txt 之后，我们需要利用该 corpus.txt 构建 中文的词库。这里采用 sentencepiece 训练中文词库。

1. sentencepiece 安装

```
    $ pip install sentencepiece
```

2. 训练中文词库

```
    import sentencepiece as spm
    spm.SentencePieceTrainer.train(
        input='data/corpus.txt',
        model_prefix='tokenizer',
        vocab_size=50000,
        user_defined_symbols=['foo', 'bar'],
        character_coverage=1.0,
        model_type="bpe",
    )
```

- 参数介绍:
    - input：指定输入文本文件的路径或者是一个目录，可以指定多个输入文件或目录。其中每一行可以是一句话或者多句话；
    - tokenizer：保存的模型的名称前缀；
    - vocab_size：设置的词表大小；
    - user_defined_symbols：用于指定用户自定义的符号。这些符号将会被视为单独的 Token，不会被拆分成子词。这个参数的作用是将一些用户定义的特殊符号作为一个整体加入到生成的词表中，以便于后续的模型使用。这里我们简单进行了测试；
    - model_type: 指定模型的类型，有三种可选参数：unigram, bpe, char. word；
    - character_coverage：指定覆盖字符的数量，可以理解为限制字符集的大小。默认值为 1.0，即覆盖全部字符；
    - unk_id: 指定未登录词的 ID 号，即在词表中为未登录词分配一个整数 ID。默认值为 0；
    - bos_id: 指定句子开头符号的 ID 号，即在词表中为句子开头符号分配一个整数 ID。默认值为 1；
    - eos_id: 指定句子结束符号的 ID 号，即在词表中为句子结束符号分配一个整数 ID。默认值为 2；
    - pad_id: 指定填充符号的 ID 号，即在词表中为填充符号分配一个整数 ID。默认值为 -1，即不使用填充符号；

运行 上述代码之后会得到 tokenizer.model和tokenizer.vocab两个文件

tokenizer.vocab 词表: 除了一些特殊符号外，还有我们自定义的foo和bar，其余的一些词是BPE训练得到

```
<unk>	0
<s>	0
```

```
</s>	0
foo 0
bar 0
萧炎	-0
..	-1
▁“	-2
也是	-3
便是	-4
了一	-5
...
```

四、如何使用transformers库加载sentencepiece模型?

chinese_bpe.py

```python
import os

os.environ["PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION"] = "python"
from transformers import LlamaTokenizer
from sentencepiece import sentencepiece_model_pb2 as sp_pb2_model
import sentencepiece as spm
from tokenization import ChineseTokenizer

chinese_sp_model_file = "sentencepisece_tokenizer/tokenizer.model"

# load
chinese_sp_model = spm.SentencePieceProcessor()
chinese_sp_model.Load(chinese_sp_model_file)

chinese_spm = sp_pb2_model.ModelProto()
chinese_spm.ParseFromString(chinese_sp_model.serialized_model_proto())

## Save
output_dir = './transformers_tokenizer/chinese/'
os.makedirs(output_dir, exist_ok=True)
with open(output_dir + 'chinese.model', 'wb') as f:
    f.write(chinese_spm.SerializeToString())
tokenizer = ChineseTokenizer(vocab_file=output_dir + 'chinese.model')

tokenizer.save_pretrained(output_dir)
print(f"Chinese tokenizer has been saved to {output_dir}")

# Test
chinese_tokenizer = ChineseTokenizer.from_pretrained(output_dir)
print(tokenizer.all_special_tokens)
print(tokenizer.all_special_ids)
print(tokenizer.special_tokens_map)
text = '''白日依山尽，黄河入海流。欲穷千里目，更上一层楼。
The primary use of LLaMA is research on large language models, including'''
```

```
print("Test text:\n", text)
print(f"Tokenized by Chinese tokenizer:{chinese_tokenizer.tokenize(text)}")
```

运行结果

```
Chinese tokenizer has been saved to ./transformers_tokenizer/chinese/
['<s>', '</s>', '<unk>']
[1, 2, 0]
{'bos_token': '<s>', 'eos_token': '</s>', 'unk_token': '<unk>'}
Test text:
 白日依山尽，黄河入海流。欲穷千里目，更上一层楼。
The primary use of LLaMA is research on large language models, including
Tokenized by Chinese-LLaMA tokenizer:['▁', '白日', '依', '山', '尽', '，', '黄',
'河', '入', '海', '流', '。', '欲', '穷', '千里', '目', '，', '更', '上一层', '楼',
'。', '▁', 'T', 'h', 'e', '▁', 'p', 'r', 'i', 'm', 'a', 'r', 'y', '▁', 'u', 's',
'e', '▁', 'o', 'f', '▁', 'LL', 'a', 'MA', '▁i', 's', '▁', 'r', 'e', 's', 'e',
'a', 'r', 'ch', '▁', 'o', 'n', '▁', 'l', 'a', 'r', 'g', 'e', '▁', 'l', 'an',
'g', 'u', 'a', 'g', 'e', '▁', 'm', 'o', 'd', 'e', 'l', 's', ',', '▁i', 'n', 'c',
'lu', 'd', 'i', 'ng']
```

注：其中ChineseTokenizer这里参考了llama模型里面使用的方法，并稍微做些修改：

```
# coding=utf-8
# Copyright 2022 EleutherAI and the HuggingFace Inc. team. All rights reserved.
#
# This code is based on EleutherAI's GPT-NeoX library and the GPT-NeoX
# and OPT implementations in this library. It has been modified from its
# original forms to accommodate minor architectural differences compared
# to GPT-NeoX and OPT used by the Meta AI team that trained the model.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Tokenization classes for LLaMA."""
import os
from shutil import copyfile
from typing import Any, Dict, List, Optional, Tuple

import sentencepiece as spm
```

```
from transformers.tokenization_utils import AddedToken, PreTrainedTokenizer
from transformers.utils import logging


logger = logging.get_logger(__name__)

VOCAB_FILES_NAMES = {"vocab_file": "tokenizer.model"}

# PRETRAINED_VOCAB_FILES_MAP = {
#     "vocab_file": {
#         "hf-internal-testing/llama-tokenizer": "https://huggingface.co/hf-
internal-testing/llama-tokenizer/resolve/main/tokenizer.model",
#     },
#     "tokenizer_file": {
#         "hf-internal-testing/llama-tokenizer": "https://huggingface.co/hf-
internal-testing/llama-tokenizer/resolve/main/tokenizer_config.json",
#     },
# }
# PRETRAINED_POSITIONAL_EMBEDDINGS_SIZES = {
#     "hf-internal-testing/llama-tokenizer": 2048,
# }


class ChineseTokenizer(PreTrainedTokenizer):
    """
    Construct a Llama tokenizer. Based on byte-level Byte-Pair-Encoding.

    Args:
        vocab_file (`str`):
            Path to the vocabulary file.
    """

    vocab_files_names = VOCAB_FILES_NAMES
    # pretrained_vocab_files_map = PRETRAINED_VOCAB_FILES_MAP
    # max_model_input_sizes = PRETRAINED_POSITIONAL_EMBEDDINGS_SIZES
    model_input_names = ["input_ids", "attention_mask"]

    def __init__(
        self,
        vocab_file,
        unk_token="<unk>",
        bos_token="<s>",
        eos_token="</s>",
        pad_token=None,
        sp_model_kwargs: Optional[Dict[str, Any]] = None,
        add_bos_token=True,
        add_eos_token=False,
        clean_up_tokenization_spaces=False,
        **kwargs,
    ):
        self.sp_model_kwargs = {} if sp_model_kwargs is None else sp_model_kwargs
        bos_token = AddedToken(bos_token, lstrip=False, rstrip=False) if
isinstance(bos_token, str) else bos_token
        eos_token = AddedToken(eos_token, lstrip=False, rstrip=False) if
isinstance(eos_token, str) else eos_token
        unk_token = AddedToken(unk_token, lstrip=False, rstrip=False) if
isinstance(unk_token, str) else unk_token
        pad_token = AddedToken(pad_token, lstrip=False, rstrip=False) if
isinstance(pad_token, str) else pad_token
        super().__init__(
            bos_token=bos_token,
            eos_token=eos_token,
            unk_token=unk_token,
            pad_token=pad_token,
            add_bos_token=add_bos_token,
            add_eos_token=add_eos_token,
            sp_model_kwargs=self.sp_model_kwargs,
            clean_up_tokenization_spaces=clean_up_tokenization_spaces,
            **kwargs,
        )
        self.vocab_file = vocab_file
        self.add_bos_token = add_bos_token
        self.add_eos_token = add_eos_token
        self.sp_model = spm.SentencePieceProcessor(**self.sp_model_kwargs)
        self.sp_model.Load(vocab_file)

    def __getstate__(self):
        state = self.__dict__.copy()
        state["sp_model"] = None
        return state

    def __setstate__(self, d):
        self.__dict__ = d
        self.sp_model = spm.SentencePieceProcessor(**self.sp_model_kwargs)
        self.sp_model.Load(self.vocab_file)

    @property
    def vocab_size(self):
        """Returns vocab size"""
        return self.sp_model.get_piece_size()

    def get_vocab(self):
        """Returns vocab as a dict"""
        vocab = {self.convert_ids_to_tokens(i): i for i in range(self.vocab_size)}
        vocab.update(self.added_tokens_encoder)
        return vocab

    def _tokenize(self, text):
        """Returns a tokenized string."""
        return self.sp_model.encode(text, out_type=str)

    def _convert_token_to_id(self, token):
        """Converts a token (str) in an id using the vocab."""
        return self.sp_model.piece_to_id(token)

    def _convert_id_to_token(self, index):
        """Converts an index (integer) in a token (str) using the vocab."""
        token = self.sp_model.IdToPiece(index)
        return token

    def convert_tokens_to_string(self, tokens):
        """Converts a sequence of tokens (string) in a single string."""
        current_sub_tokens = []
        out_string = ""
        prev_is_special = False
        for i, token in enumerate(tokens):
            # make sure that special tokens are not decoded using sentencepiece
model
            if token in self.all_special_tokens:
                if not prev_is_special and i != 0:
                    out_string += " "
                out_string += self.sp_model.decode(current_sub_tokens) + token
                prev_is_special = True
                current_sub_tokens = []
            else:
                current_sub_tokens.append(token)
                prev_is_special = False
        out_string += self.sp_model.decode(current_sub_tokens)
        return out_string

    def save_vocabulary(self, save_directory, filename_prefix: Optional[str] =
None) -> Tuple[str]:
        """
        Save the vocabulary and special tokens file to a directory.

        Args:
            save_directory (`str`):
                The directory in which to save the vocabulary.

        Returns:
            `Tuple(str)`: Paths to the files saved.
        """
        if not os.path.isdir(save_directory):
            logger.error(f"Vocabulary path ({save_directory}) should be a
directory")
            return
        out_vocab_file = os.path.join(
            save_directory, (filename_prefix + "-" if filename_prefix else "") +
VOCAB_FILES_NAMES["vocab_file"]
        )

        if os.path.abspath(self.vocab_file) != os.path.abspath(out_vocab_file) and
os.path.isfile(self.vocab_file):
            copyfile(self.vocab_file, out_vocab_file)
        elif not os.path.isfile(self.vocab_file):
            with open(out_vocab_file, "wb") as fi:
                content_spiece_model = self.sp_model.serialized_model_proto()
                fi.write(content_spiece_model)

        return (out_vocab_file,)

    def build_inputs_with_special_tokens(self, token_ids_0, token_ids_1=None):
        bos_token_id = [self.bos_token_id] if self.add_bos_token else []
        eos_token_id = [self.eos_token_id] if self.add_eos_token else []

        output = bos_token_id + token_ids_0 + eos_token_id

        if token_ids_1 is not None:
            output = output + bos_token_id + token_ids_1 + eos_token_id

        return output

    def get_special_tokens_mask(
        self, token_ids_0: List[int], token_ids_1: Optional[List[int]] = None,
already_has_special_tokens: bool = False
    ) -> List[int]:
        """
        Retrieve sequence ids from a token list that has no special tokens added.
This method is called when adding
        special tokens using the tokenizer `prepare_for_model` method.

        Args:
            token_ids_0 (`List[int]`):
                List of IDs.
            token_ids_1 (`List[int]`, *optional*):
                Optional second list of IDs for sequence pairs.
            already_has_special_tokens (`bool`, *optional*, defaults to `False`):
                Whether or not the token list is already formatted with special
tokens for the model.

        Returns:
            `List[int]`: A list of integers in the range [0, 1]: 1 for a special
token, 0 for a sequence token.
        """
        if already_has_special_tokens:
            return super().get_special_tokens_mask(
                token_ids_0=token_ids_0, token_ids_1=token_ids_1,
already_has_special_tokens=True
            )

        bos_token_id = [1] if self.add_bos_token else []
        eos_token_id = [1] if self.add_eos_token else []

        if token_ids_1 is None:
            return bos_token_id + ([0] * len(token_ids_0)) + eos_token_id
        return (
            bos_token_id
            + ([0] * len(token_ids_0))
            + eos_token_id
            + bos_token_id
            + ([0] * len(token_ids_1))
            + eos_token_id
        )

    def create_token_type_ids_from_sequences(
        self, token_ids_0: List[int], token_ids_1: Optional[List[int]] = None
    ) -> List[int]:
        """
        Creates a mask from the two sequences passed to be used in a sequence-pair
classification task. An ALBERT
        sequence pair mask has the following format:

        ```
        0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
        | first sequence    | second sequence |
        ```

        if token_ids_1 is None, only returns the first portion of the mask (0s).

        Args:
            token_ids_0 (`List[int]`):
                List of ids.
            token_ids_1 (`List[int]`, *optional*):
                Optional second list of IDs for sequence pairs.
```

```
        Returns:
            `List[int]`: List of [token type IDs](../glossary#token-type-ids)
according to the given sequence(s).
        """
        bos_token_id = [self.bos_token_id] if self.add_bos_token else []
        eos_token_id = [self.eos_token_id] if self.add_eos_token else []

        output = [0] * len(bos_token_id + token_ids_0 + eos_token_id)

        if token_ids_1 is not None:
            output += [1] * len(bos_token_id + token_ids_1 + eos_token_id)

        return output
```

不难发现其实里面使用了一些sentencepiece里面的函数。

五、如何合并英文词表和中文词表?

chinese_llama_bpe.py

```
import os

os.environ["PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION"] = "python"
from transformers import LlamaTokenizer
from sentencepiece import sentencepiece_model_pb2 as sp_pb2_model
import sentencepiece as spm

llama_tokenizer_dir = "transformers_tokenizer/llama/tokenizer.model"
chinese_sp_model_file = "sentencepisece_tokenizer/tokenizer.model"

# load
llama_tokenizer = LlamaTokenizer.from_pretrained(llama_tokenizer_dir)
llama_spm = sp_pb2_model.ModelProto()
llama_spm.ParseFromString(llama_tokenizer.sp_model.serialized_model_proto())
chinese_spm = sp_pb2_model.ModelProto()
chinese_sp_model = spm.SentencePieceProcessor()
chinese_sp_model.Load(chinese_sp_model_file)
chinese_spm.ParseFromString(chinese_sp_model.serialized_model_proto())

# print number of tokens
print(len(llama_tokenizer), len(chinese_sp_model))
print(llama_tokenizer.all_special_tokens)
print(llama_tokenizer.all_special_ids)
print(llama_tokenizer.special_tokens_map)

## Add Chinese tokens to LLaMA tokenizer
llama_spm_tokens_set = set(p.piece for p in llama_spm.pieces)
print(len(llama_spm_tokens_set))
print(f"Before:{len(llama_spm_tokens_set)}")
for p in chinese_spm.pieces:
    piece = p.piece
    if piece not in llama_spm_tokens_set:
        new_p = sp_pb2_model.ModelProto().SentencePiece()
        new_p.piece = piece
        new_p.score = 0
        llama_spm.pieces.append(new_p)
print(f"New model pieces: {len(llama_spm.pieces)}")

## Save
output_sp_dir = 'transformers_tokenizer/llama_chinese'
output_hf_dir = 'transformers_tokenizer/llama_chinese'  # the path to save Chinese-
LLaMA tokenizer
os.makedirs(output_sp_dir, exist_ok=True)
with open(output_sp_dir + '/chinese_llama.model', 'wb') as f:
    f.write(llama_spm.SerializeToString())
tokenizer = LlamaTokenizer(vocab_file=output_sp_dir + '/chinese_llama.model')

tokenizer.save_pretrained(output_hf_dir)
print(f"Chinese-LLaMA tokenizer has been saved to {output_hf_dir}")

# Test
llama_tokenizer = LlamaTokenizer.from_pretrained(llama_tokenizer_dir)
chinese_llama_tokenizer = LlamaTokenizer.from_pretrained(output_hf_dir)
print(tokenizer.all_special_tokens)
print(tokenizer.all_special_ids)
print(tokenizer.special_tokens_map)
text = '''白日依山尽，黄河入海流。欲穷千里目，更上一层楼。
The primary use of LLaMA is research on large language models, including'''
print("Test text:\n", text)
print(f"Tokenized by LLaMA tokenizer:{llama_tokenizer.tokenize(text)}")
print(f"Tokenized by Chinese-LLaMA tokenizer:
{chinese_llama_tokenizer.tokenize(text)}")
```

代码中，最核心的是以下这一块，该代码的作用是 将 原始词表 中没有的 新词 加入 词表中

```
for p in chinese_spm.pieces:
    piece = p.piece
    if piece not in llama_spm_tokens_set:
        new_p = sp_pb2_model.ModelProto().SentencePiece()
        new_p.piece = piece
        new_p.score = 0
        llama_spm.pieces.append(new_p)
```

运行结果

```
32000 50000
['<s>', '</s>', '<unk>']
```

```
[1, 2, 0]
{'bos_token': '<s>', 'eos_token': '</s>', 'unk_token': '<unk>'}
32000
Before:32000
New model pieces: 81163
Chinese-LLaMA tokenizer has been saved to transformers_tokenizer/llama_chinese
['<s>', '</s>', '<unk>']
[1, 2, 0]
{'bos_token': '<s>', 'eos_token': '</s>', 'unk_token': '<unk>'}
Test text:
 白日依山尽，黄河入海流。欲穷千里目，更上一层楼。
The primary use of LLaMA is research on large language models, including
Tokenized by LLaMA tokenizer:['▁', '白', '日', '<0xE4>', '<0xBE>', '<0x9D>', '山',
'<0xE5>', '<0xB0>', '<0xBD>', '，', '黄', '河', '入', '海', '流', '。', '<0xE6>',
'<0xAC>', '<0xB2>', '<0xE7>', '<0xA9>', '<0xB7>', '千', '里', '目', '，', '更',
'上', '一', '<0xE5>', '<0xB1>', '<0x82>', '<0xE6>', '<0xA5>', '<0xBC>', '。',
'<0x0A>', 'The', '▁primary', '▁use', '▁of', '▁L', 'La', 'MA', '▁is',
'▁research', '▁on', '▁large', '▁language', '▁models', ',', '▁including']
Tokenized by Chinese-LLaMA tokenizer:['▁白', '日', '依', '山', '尽', '，', '黄',
'河', '入', '海', '流', '。', '欲', '穷', '千里', '目', '，', '更', '上一层', '楼',
'。', '<0x0A>', 'The', '▁primary', '▁use', '▁of', '▁L', 'La', 'MA', '▁is',
'▁research', '▁on', '▁large', '▁language', '▁models', ',', '▁including']
```

> 注：会发现再加入了我们定义的词表后确实能够对中文进行分词了。

六、怎么使用修改后的词表?

如果我们重新从头开始训练，那么其实使用起来很简单：

```
config = AutoConfig.from_pretrained(...)
tokenizer = LlamaTokenizer.from_pretrained(...)
model = LlamaForCausalLM.from_pretrained(..., config=config)
model_vocab_size = model.get_output_embeddings().weight.size(0)
model.resize_token_embeddings(len(tokenizer))
```

但是如果我们想要保留原始模型embedding的参数，那么我们可以这么做：

1. 找到新词表和旧词表id之间的映射关系。
2. 将模型里面新词表里面包含的旧词表用原始模型的embedding替换。
3. 如果新词在旧词表里面没有出现就进行相应的初始化再进行赋值。比如transformers库中的llama是这么进行初始化的：

```
 def _init_weights(self, module):
        std = self.config.initializer_range
        if isinstance(module, nn.Linear):
            module.weight.data.normal_(mean=0.0, std=std)
            if module.bias is not None:
                module.bias.data.zero_()
        elif isinstance(module, nn.Embedding):
            module.weight.data.normal_(mean=0.0, std=std)
            if module.padding_idx is not None:
                    module.weight.data[module.padding_idx].zero_()
```

> 注：具体怎么做可以参考一下这个：https://github.com/yangjianxin1/LLMPruner

总结一下 构建中文tokenization?

到这里为止，我们已经学会了：

1. 使用sentencepiece训练一个中文的词表。
2. 使用transformers加载sentencepiece模型。
3. 怎么合并中英文的词表，并使用transformers使用合并后的词表。
4. 在模型中怎么使用新词表。