Huggingface

主题：

文本生成任务：

1. **使用预训练的语言模型生成文本**：
    - 介绍预训练语言模型的概念和用途。
    - 使用Hugging Face Transformers库加载预训练模型。
    - 使用模型生成文本序列。
2. **调整生成文本的风格和内容**：
    - 引入条件生成的概念，控制生成文本的方向。
    - 修改输入文本以引导生成结果的风格和主题。
3. **使用不同的生成策略**：
    - 介绍不同的生成策略，如贪婪、随机采样、束搜索等。
    - 在Hugging Face Transformers库中应用不同的策略。

文本分类任务：

1. **使用预训练模型进行文本分类**：
    - 介绍文本分类任务及其应用。
    - 使用预训练模型加载分类器。
    - 输入文本并获得分类结果。
2. **准备数据集并进行微调**：
    - 解释微调的概念和步骤。
    - 准备自己的数据集，包括文本和标签。
    - 使用微调技术将预训练模型应用于特定任务。
3. **进行模型评估和预测**：
    - 划分数据集为训练集和测试集。
    - 训练微调后的模型，并评估其在测试集上的性能。
    - 使用模型进行新文本的分类预测。

（以下主题类似地进行介绍）

命名实体识别：

1. 使用预训练模型识别文本中的命名实体。
2. 解释命名实体识别任务和NER标签体系。
3. 准备带有NER标注的数据集。
4. 使用预训练模型进行NER。

情感分析：

1. 使用预训练模型分析文本情感。
2. 了解情感分析任务及其应用领域。
3. 准备情感分类数据集。
4. 微调模型进行情感分析。

问答系统：

1. 构建基于预训练模型的问答系统。
2. 理解问答系统的输入输出。
3. 准备问题和文本数据集。
4. 使用微调技术构建问答模型。

多模态任务：

1. 结合文本和图像信息进行多模态任务。
2. 介绍多模态任务的应用领域。
3. 准备多模态数据集。
4. 使用预训练的多模态模型处理数据。

模型微调和自定义：

1. 了解如何微调预训练模型以适应特定任务。
2. 自定义模型架构和训练过程。
3. 使用适当的优化器和损失函数。
4. 保存和加载微调后的模型。

模型共享和部署：

1. 将训练好的模型保存并共享给他人。
2. 使用Transformers库轻松部署模型。
3. 选择本地或云上部署NLP模型的方法。

实际项目应用：

1. 通过一个完整的NLP项目案例，将前述知识融合在一起。
2. 从数据收集、预处理到模型选择、微调和部署。

`transformers`

是由Hugging Face开发的一个用于自然语言处理（NLP）任务的Python库，它为训练和使用预训练的语言模型提供了简单且高效的接口。该库主要关注文本生成、文本分类、命名实体识别、机器翻译等各种NLP任务。`transformers`库基于PyTorch和TensorFlow等深度学习框架，并支持大量的预训练模型，如BERT、GPT-2、T5等。以下是`transformers`库中一些常用的功能和模块：

1. **加载预训练模型：**
   - `transformers.AutoModel`：自动加载适合任务的预训练模型。
2. **模型配置：**
   - `transformers.AutoConfig`：用于配置模型参数。
3. **分词和处理输入：**
   - `transformers.AutoTokenizer`：用于分词和编码文本。
   - `transformers.PreTrainedTokenizerFast`：高效的分词器。
4. **模型训练和评估：**
   - `transformers.Trainer`：用于模型训练。
   - `transformers.pipeline`：快速运行各种NLP任务。
5. **文本生成：**
   - `transformers.TextGenerationPipeline`：生成文本。
6. **文本分类：**
   - `transformers.TextClassificationPipeline`：进行文本分类。
7. **命名实体识别：**

- `transformers.TokenClassificationPipeline`：进行命名实体识别。

8. **翻译**：
   - `transformers.TranslationPipeline`：进行文本翻译。

`transformers`库使得使用预训练模型进行各种NLP任务变得非常便捷，可以大大缩短开发周期。您可以使用以下命令安装`transformers`库：

```
pip install transformers
```

然后，通过引入库并使用相应的模块和类，您可以开始使用`transformers`进行文本处理和NLP任务。如果您想深入了解`transformers`的使用和功能，可以查阅Hugging Face的官方文档以及相关教程。

1.install

```
#环境
#python=3.6

import sys

sys.version
```

```
'3.6.13 |Anaconda, Inc.| (default, Jun  4 2021, 14:25:59) \n[GCC 7.5.0]'
```

```
#安装torch,简单起见,避免环境问题,并且计算量不大,使用cpu运行本套代码.
#后面章节会介绍使用cuda计算的方法.

#pip install torch==1.10.1+cpu torchvision==0.11.2+cpu torchaudio==0.10.1 -f
https://download.pytorch.org/whl/cpu/torch_stable.html

import torch
import torchvision
import torchaudio

torch.__version__, torchvision.__version__, torchaudio.__version__
```

```
('1.10.1+cpu', '0.11.2+cpu', '0.10.1+cpu')
```

```
#安装transformers
#pip安装
#pip install transformers==4.18.0

#conda安装(不推荐)
#conda install -c huggingface transformers

import transformers

transformers.__version__
```

```
'4.18.0'
```

```
#安装datasets
#pip安装
#pip install datasets==2.4.0

#conda安装(不推荐)
#conda install -c huggingface -c conda-forge datasets

import datasets

datasets.__version__
```

```
'2.4.0'
```

```
#安装torchtext
#pip install torchtext==0.11.2

import torchtext

torchtext.__version__
```

```
'0.11.2'
```

2.tokenizer

```
from transformers import BertTokenizer
```

```
#加载预训练字典和分词方法
tokenizer = BertTokenizer.from_pretrained(
    pretrained_model_name_or_path='bert-base-chinese',
    cache_dir=None,
    force_download=False,
)

sents = [
    '选择珠江花园的原因就是方便。',
    '笔记本的键盘确实爽。',
    '房间太小。其他的都一般。',
    '今天才知道这书还有第6卷,真有点郁闷.',
    '机器背面似乎被撕了张什么标签，残胶还在。',
]

tokenizer, sents
```

```
Downloading:   0%|          | 0.00/107k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/29.0 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/624 [00:00<?, ?B/s]
```

```
(PreTrainedTokenizer(name_or_path='bert-base-chinese', vocab_size=21128,
model_max_len=512, is_fast=False, padding_side='right', truncation_side='right',
special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]',
'cls_token': '[CLS]', 'mask_token': '[MASK]'}),
 ['选择珠江花园的原因就是方便。',
  '笔记本的键盘确实爽。',
  '房间太小。其他的都一般。',
  '今天才知道这书还有第6卷,真有点郁闷.',
  '机器背面似乎被撕了张什么标签，残胶还在。'])
```

```
#编码两个句子
out = tokenizer.encode(
    text=sents[0],
    text_pair=sents[1],

    #当句子长度大于max_length时,截断
    truncation=True,

    #一律补pad到max_length长度
    padding='max_length',
```

```
    add_special_tokens=True,
    max_length=30,
    return_tensors=None,
)

print(out)

tokenizer.decode(out)
```

```
[101, 6848, 2885, 4403, 3736, 5709, 1736, 4638, 1333, 1728, 2218, 3221, 3175,
912, 511, 102, 5011, 6381, 3315, 4638, 7241, 4669, 4802, 2141, 4272, 511, 102, 0,
0, 0]
```

```
'[CLS] 选 择 珠 江 花 园 的 原 因 就 是 方 便 。 [SEP] 笔 记 本 的 键 盘 确 实 爽 。
[SEP] [PAD] [PAD] [PAD]'
```

```
#增强的编码函数
out = tokenizer.encode_plus(
    text=sents[0],
    text_pair=sents[1],

    #当句子长度大于max_length时,截断
    truncation=True,

    #一律补零到max_length长度
    padding='max_length',
    max_length=30,
    add_special_tokens=True,

    #可取值tf,pt,np,默认为返回list
    return_tensors=None,

    #返回token_type_ids
    return_token_type_ids=True,

    #返回attention_mask
    return_attention_mask=True,

    #返回special_tokens_mask 特殊符号标识
    return_special_tokens_mask=True,

    #返回offset_mapping 标识每个词的起止位置,这个参数只能BertTokenizerFast使用
    #return_offsets_mapping=True,

    #返回length 标识长度
    return_length=True,
)
```

```
#input_ids 就是编码后的词
#token_type_ids 第一个句子和特殊符号的位置是0,第二个句子的位置是1
#special_tokens_mask 特殊符号的位置是1,其他位置是0
#attention_mask pad的位置是0,其他位置是1
#length 返回句子长度
for k, v in out.items():
    print(k, ':', v)

tokenizer.decode(out['input_ids'])
```

```
input_ids : [101, 6848, 2885, 4403, 3736, 5709, 1736, 4638, 1333, 1728, 2218,
3221, 3175, 912, 511, 102, 5011, 6381, 3315, 4638, 7241, 4669, 4802, 2141, 4272,
511, 102, 0, 0, 0]
token_type_ids : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0]
special_tokens_mask : [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
attention_mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0]
length : 30
```

```
'[CLS] 选 择 珠 江 花 园 的 原 因 就 是 方 便 。 [SEP] 笔 记 本 的 键 盘 确 实 爽 。
[SEP] [PAD] [PAD] [PAD]'
```

```
#批量编码句子
out = tokenizer.batch_encode_plus(
    batch_text_or_text_pairs=[sents[0], sents[1]],
    add_special_tokens=True,

    #当句子长度大于max_length时,截断
    truncation=True,

    #一律补零到max_length长度
    padding='max_length',
    max_length=15,

    #可取值tf,pt,np,默认为返回list
    return_tensors=None,

    #返回token_type_ids
    return_token_type_ids=True,

    #返回attention_mask
    return_attention_mask=True,

    #返回special_tokens_mask 特殊符号标识
    return_special_tokens_mask=True,
```

```
    #返回offset_mapping 标识每个词的起止位置,这个参数只能BertTokenizerFast使用
    #return_offsets_mapping=True,

    #返回length 标识长度
    return_length=True,
)

#input_ids 就是编码后的词
#token_type_ids 第一个句子和特殊符号的位置是0,第二个句子的位置是1
#special_tokens_mask 特殊符号的位置是1,其他位置是0
#attention_mask pad的位置是0,其他位置是1
#length 返回句子长度
for k, v in out.items():
    print(k, ':', v)

tokenizer.decode(out['input_ids'][0]), tokenizer.decode(out['input_ids'][1])
```

```
input_ids : [[101, 6848, 2885, 4403, 3736, 5709, 1736, 4638, 1333, 1728, 2218,
3221, 3175, 912, 102], [101, 5011, 6381, 3315, 4638, 7241, 4669, 4802, 2141,
4272, 511, 102, 0, 0, 0]]
token_type_ids : [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
special_tokens_mask : [[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]]
length : [15, 12]
attention_mask : [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 0, 0]]
```

```
('[CLS] 选 择 珠 江 花 园 的 原 因 就 是 方 便 [SEP]',
 '[CLS] 笔 记 本 的 键 盘 确 实 爽 。 [SEP] [PAD] [PAD] [PAD]')
```

```
#批量编码成对的句子
out = tokenizer.batch_encode_plus(
    batch_text_or_text_pairs=[(sents[0], sents[1]), (sents[2], sents[3])],
    add_special_tokens=True,

    #当句子长度大于max_length时,截断
    truncation=True,

    #一律补零到max_length长度
    padding='max_length',
    max_length=30,

    #可取值tf,pt,np,默认为返回list
    return_tensors=None,

    #返回token_type_ids
    return_token_type_ids=True,
```

```
    #返回attention_mask
    return_attention_mask=True,

    #返回special_tokens_mask 特殊符号标识
    return_special_tokens_mask=True,

    #返回offset_mapping 标识每个词的起止位置,这个参数只能BertTokenizerFast使用
    #return_offsets_mapping=True,

    #返回length 标识长度
    return_length=True,
)

#input_ids 就是编码后的词
#token_type_ids 第一个句子和特殊符号的位置是0,第二个句子的位置是1
#special_tokens_mask 特殊符号的位置是1,其他位置是0
#attention_mask pad的位置是0,其他位置是1
#length 返回句子长度
for k, v in out.items():
    print(k, ':', v)

tokenizer.decode(out['input_ids'][0])
```

```
Be aware, overflowing tokens are not returned for the setting you have chosen,
i.e. sequence pairs with the 'longest_first' truncation strategy. So the returned
list will always be empty even if some tokens have been removed.
```

```
input_ids : [[101, 6848, 2885, 4403, 3736, 5709, 1736, 4638, 1333, 1728, 2218,
3221, 3175, 912, 511, 102, 5011, 6381, 3315, 4638, 7241, 4669, 4802, 2141, 4272,
511, 102, 0, 0, 0], [101, 2791, 7313, 1922, 2207, 511, 1071, 800, 4638, 6963,
671, 5663, 511, 102, 791, 1921, 2798, 4761, 6887, 6821, 741, 6820, 3300, 5018,
127, 1318, 117, 4696, 3300, 102]]
token_type_ids : [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
special_tokens_mask : [[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
length : [27, 30]
attention_mask : [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
```

```
'[CLS] 选 择 珠 江 花 园 的 原 因 就 是 方 便 。 [SEP] 笔 记 本 的 键 盘 确 实 爽 。
[SEP] [PAD] [PAD] [PAD]'
```

```
#获取字典
zidian = tokenizer.get_vocab()

type(zidian), len(zidian), '月光' in zidian,
```

```
(dict, 21128, False)
```

```
#添加新词
tokenizer.add_tokens(new_tokens=['月光', '希望'])

#添加新符号
tokenizer.add_special_tokens({'eos_token': '[EOS]'})

zidian = tokenizer.get_vocab()

type(zidian), len(zidian), zidian['月光'], zidian['[EOS]']
```

```
(dict, 21131, 21128, 21130)
```

```
#编码新添加的词
out = tokenizer.encode(
    text='月光的新希望[EOS]',
    text_pair=None,

    #当句子长度大于max_length时,截断
    truncation=True,

    #一律补pad到max_length长度
    padding='max_length',
    add_special_tokens=True,
    max_length=8,
    return_tensors=None,
)

print(out)

tokenizer.decode(out)
```

```
[101, 21128, 4638, 3173, 21129, 21130, 102, 0]
```

```
'[CLS] 月光 的 新 希望 [EOS] [SEP] [PAD]'
```

3.datasets

```
from datasets import load_dataset

#加载数据
#注意：如果你的网络不允许你执行这段的代码，则直接运行【从磁盘加载数据】即可，我已经给你准备了本地化的数据文件
#转载自seamew/ChnSentiCorp
dataset = load_dataset(path='lansinuote/ChnSentiCorp')

dataset
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet (/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d6f236ec)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 9600
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 1200
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 1200
    })
})
```

```
#保存数据集到磁盘
#注意：运行这段代码要确保【加载数据】运行是正常的，否则直接运行【从磁盘加载数据】即可
dataset.save_to_disk(dataset_dict_path='./data/ChnSentiCorp')
```

```
#从磁盘加载数据
from datasets import load_from_disk

dataset = load_from_disk('./data/ChnSentiCorp')

dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 9600
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 1200
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 1200
    })
})
```

```
#取出训练集
dataset = dataset['train']

dataset
```

```
Dataset({
    features: ['text', 'label'],
    num_rows: 9600
})
```

```
#查看一个数据
dataset[0]
```

```
{'text': '选择珠江花园的原因就是方便，有电动扶梯直接到达海边，周围餐馆、食廊、商场、超市、摊位一应俱全。酒店装修一般，但还算整洁。 泳池在大堂的屋顶，因此很小，不过女儿倒是喜欢。 包的早餐是西式的，还算丰富。 服务吗，一般',
 'label': 1}
```

```
#sort

#未排序的label是乱序的
print(dataset['label'][:10])

#排序之后label有序了
sorted_dataset = dataset.sort('label')
print(sorted_dataset['label'][:10])
print(sorted_dataset['label'][-10:])
```

```
Loading cached sorted indices for dataset at data/ChnSentiCorp/train/cache-c15802113fb68285.arrow
```

```
[1, 1, 0, 0, 1, 0, 0, 0, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
#shuffle

#打乱顺序
shuffled_dataset = sorted_dataset.shuffle(seed=42)

shuffled_dataset['label'][:10]
```

```
Loading cached shuffled indices for dataset at data/ChnSentiCorp/train/cache-c0614c09b7c97d80.arrow
```

```
[0, 1, 0, 0, 1, 0, 1, 0, 1, 0]
```

```
#select
dataset.select([0, 10, 20, 30, 40, 50])
```

```
Dataset({
    features: ['text', 'label'],
    num_rows: 6
})
```

```
#filter
def f(data):
    return data['text'].startswith('选择')


start_with_ar = dataset.filter(f)

len(start_with_ar), start_with_ar['text']
```

```
Loading cached processed dataset at data/ChnSentiCorp/train/cache-
0857ea564c515b0e.arrow
```

```
(2,
 ['选择珠江花园的原因就是方便，有电动扶梯直接到达海边，周围餐馆、食廊、商场、超市、摊位一应俱全。酒店装修一般，但还算整洁。 泳池在大堂的屋顶，因此很小，不过女儿倒是喜欢。 包的早餐是西式的，还算丰富。 服务吗，一般',
  '选择的事例太离奇了，夸大了心理咨询的现实意义，让人失去了信任感！如果说这样写的效果能在一开始抓住读者的眼球，但是看到案例主人公心理问题的原因解释时就逐渐失去了兴趣，反正有点拣了芝麻丢了西瓜的感觉。'])
```

```
#train_test_split, 切分训练集和测试集
dataset.train_test_split(test_size=0.1)
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 8640
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 960
    })
})
```

```
#shard
#把数据切分到4个桶中,均匀分配
dataset.shard(num_shards=4, index=0)
```

```
Dataset({
    features: ['text', 'label'],
    num_rows: 2400
})
```

```
#rename_column
dataset.rename_column('text', 'textA')
```

```
Dataset({
    features: ['textA', 'label'],
    num_rows: 9600
})
```

```
#remove_columns
dataset.remove_columns(['text'])
```

```
Dataset({
    features: ['label'],
    num_rows: 9600
})
```

```
#map
def f(data):
    data['text'] = 'My sentence: ' + data['text']
    return data


datatset_map = dataset.map(f)

datatset_map['text'][:5]
```

```
Loading cached processed dataset at data/ChnSentiCorp/train/cache-
f6a0b1347879d84e.arrow
```

```
['My sentence: 选择珠江花园的原因就是方便，有电动扶梯直接到达海边，周围餐馆、食廊、商场、超市、摊位一应俱全。酒店装修一般，但还算整洁。 泳池在大堂的屋顶，因此很小，不过女儿倒是喜欢。 包的早餐是西式的，还算丰富。 服务吗，一般',
 'My sentence: 15.4寸笔记本的键盘确实爽，基本跟台式机差不多了，蛮喜欢数字小键盘，输数字特方便，样子也很美观，做工也相当不错',
 'My sentence: 房间太小。其他的都一般。。。。。。。。。',
 'My sentence: 1.接电源没有几分钟,电源适配器热的不行. 2.摄像头用不起来. 3.机盖的钢琴漆，手不能摸，一摸一个印. 4.硬盘分区不好办.',
 'My sentence: 今天才知道这书还有第6卷,真有点郁闷:为什么同一套书有两种版本呢?当当网是不是该跟出版社商量商量,单独出个第6卷,让我们的孩子不会有所遗憾。']
```

```
#set_format
dataset.set_format(type='torch', columns=['label'])

dataset[0]
```

```
{'label': tensor(1)}
```

```
#第三章/导出为csv格式
dataset = load_dataset(path='lansinuote/ChnSentiCorp', split='train')
dataset.to_csv(path_or_buf='./data/ChnSentiCorp.csv')

#加载csv格式数据
csv_dataset = load_dataset(path='csv',
                           data_files='./data/ChnSentiCorp.csv',
                           split='train')
csv_dataset[20]
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
Creating CSV from Arrow format:   0%|          | 0/1 [00:00<?, ?ba/s]
```

```
Using custom data configuration default-b26d23b0ffd2d191
```

```
Downloading and preparing dataset csv/default to
/root/.cache/huggingface/datasets/csv/default-
b26d23b0ffd2d191/0.0.0/652c3096f041ee27b04d2232d41f10547a8fecda3e284a79a0ec4053c
916ef7a...
```

```
Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Extracting data files:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
0 tables [00:00, ? tables/s]
```

```
Dataset csv downloaded and prepared to
/root/.cache/huggingface/datasets/csv/default-
b26d23b0ffd2d191/0.0.0/652c3096f041ee27b04d2232d41f10547a8fecda3e284a79a0ec4053c
916ef7a. Subsequent calls will reuse this data.
```

```
{'Unnamed: 0': 20, 'text': '非常不错，服务很好，位于市中心区，交通方便，不过价格也高！',
'label': 1}
```

```
#第三章/导出为json格式
dataset = load_dataset(path='lansinuote/ChnSentiCorp', split='train')
dataset.to_json(path_or_buf='./data/ChnSentiCorp.json')

#加载json格式数据
json_dataset = load_dataset(path='json',
                            data_files='./data/ChnSentiCorp.json',
                            split='train')
json_dataset[20]
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
Creating json from Arrow format:   0%|          | 0/1 [00:00<?, ?ba/s]
```

```
Using custom data configuration default-e31f8430caec396d
```

```
Downloading and preparing dataset json/default to
/root/.cache/huggingface/datasets/json/default-
e31f8430caec396d/0.0.0/a3e658c4731e59120d44081ac10bf85dc7e1388126b92338344ce9661
907f253...
```

```
Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
Extracting data files:   0%|          | 0/1 [00:00<?, ?it/s]
```

```
0 tables [00:00, ? tables/s]
```

```
Dataset json downloaded and prepared to
/root/.cache/huggingface/datasets/json/default-
e31f8430caec396d/0.0.0/a3e658c4731e59120d44081ac10bf85dc7e1388126b92338344ce9661
907f253. Subsequent calls will reuse this data.
```

```
{'text': '非常不错，服务很好，位于市中心区，交通方便，不过价格也高！', 'label': 1}
```

4.metrics

```
from datasets import list_metrics

#列出评价指标
metrics_list = list_metrics()

len(metrics_list), metrics_list
```

```
(121,
 ['accuracy',
  'bertscore',
  'bleu',
  'bleurt',
  'brier_score',
  'cer',
  'character',
  'charcut_mt',
  'chrf',
```

```
  'code_eval',
  'comet',
  'competition_math',
  'coval',
  'cuad',
  'exact_match',
  'f1',
  'frugalscore',
  'glue',
  'google_bleu',
  'indic_glue',
  'mae',
  'mahalanobis',
  'mape',
  'mase',
  'matthews_correlation',
  'mauve',
  'mean_iou',
  'meteor',
  'mse',
  'nist_mt',
  'pearsonr',
  'perplexity',
  'poseval',
  'precision',
  'recall',
  'rl_reliability',
  'roc_auc',
  'rouge',
  'sacrebleu',
  'sari',
  'seqeval',
  'smape',
  'spearmanr',
  'squad',
  'squad_v2',
  'super_glue',
  'ter',
  'trec_eval',
  'wer',
  'wiki_split',
  'xnli',
  'xtreme_s',
  'BucketHeadP65/confusion_matrix',
  'BucketHeadP65/roc_curve',
  'Drunper/metrica_tesi',
  'Felipehonorato/my_metric',
  'GMFTBY/dailydialog_evaluate',
  'GMFTBY/dailydialogevaluate',
  'JP-SystemsX/nDCG',
  'Josh98/nl2bash_m',
  'KevinSpaghetti/accuracyk',
  'NCSOFT/harim_plus',
  'NikitaMartynov/spell-check-metric',
  'NimaBoscarino/weat',
```

```
'Ochiroo/rouge_mn',
'Vertaix/vendiscore',
'Viona/infolm',
'Vlasta/pr_auc',
'abdusah/aradiawer',
'abidlabs/mean_iou',
'abidlabs/mean_iou2',
'angelina-wang/directional_bias_amplification',
'anz2/iliauniiccocrevaluation',
'bstrai/classification_report',
'cakiki/ndcg',
'codeparrot/apps_metric',
'cpllab/syntaxgym',
'daiyizheng/valid',
'dvitel/codebleu',
'ecody726/bertscore',
'erntkn/dice_coefficient',
'giulio98/code_eval_outputs',
'giulio98/codebleu',
'gnail/cosine_similarity',
'gorkaartola/metric_for_tp_fp_samples',
'hack/test_metric',
'harshhpareek/bertscore',
'hpi-dhc/FairEval',
'idsedykh/codebleu',
'idsedykh/codebleu2',
'idsedykh/megaglue',
'idsedykh/metric',
'jordyvl/ece',
'jpxkqx/peak_signal_to_noise_ratio',
'jpxkqx/signal_to_reconstrution_error',
'jzm-mailchimp/joshs_second_test_metric',
'kaggle/ai4code',
'kaggle/amex',
'kashif/mape',
'kasmith/woodscore',
'kyokote/my_metric2',
'leslyarun/fbeta_score',
'loubnabnl/apps_metric2',
'lvwerra/accuracy_score',
'lvwerra/bary_score',
'lvwerra/test',
'mfumanelli/geometric_mean',
'mgfrantz/roc_auc_macro',
'ola13/precision_at_k',
'omidf/squad_precision_recall',
'posicube/mean_reciprocal_rank',
'ronaldahmed/nwentfaithfulness',
'sportlosos/sescore',
'transZ/sbert_cosine',
'transZ/test_parascore',
'xu1998hz/sescore',
'ybelkada/cocoevaluate',
'yonting/average_precision_score',
'yulong-me/yl_metric',
```

```
  'yzha/ctc_eval',
  'zbeloki/m2'])
```

```
from datasets import load_metric

#加载一个评价指标
metric = load_metric('glue', 'mrpc')

print(metric.inputs_description)
```

```
Compute GLUE evaluation metric associated to each GLUE dataset.
Args:
    predictions: list of predictions to score.
        Each translation should be tokenized into a list of tokens.
    references: list of lists of references for each translation.
        Each reference should be tokenized into a list of tokens.
Returns: depending on the GLUE subset, one or several of:
    "accuracy": Accuracy
    "f1": F1 score
    "pearson": Pearson Correlation
    "spearmanr": Spearman Correlation
    "matthews_correlation": Matthew Correlation
Examples:

    >>> glue_metric = datasets.load_metric('glue', 'sst2')  # 'sst2' or any of
["mnli", "mnli_mismatched", "mnli_matched", "qnli", "rte", "wnli", "hans"]
    >>> references = [0, 1]
    >>> predictions = [0, 1]
    >>> results = glue_metric.compute(predictions=predictions,
references=references)
    >>> print(results)
    {'accuracy': 1.0}

    >>> glue_metric = datasets.load_metric('glue', 'mrpc')  # 'mrpc' or 'qqp'
    >>> references = [0, 1]
    >>> predictions = [0, 1]
    >>> results = glue_metric.compute(predictions=predictions,
references=references)
    >>> print(results)
    {'accuracy': 1.0, 'f1': 1.0}

    >>> glue_metric = datasets.load_metric('glue', 'stsb')
    >>> references = [0., 1., 2., 3., 4., 5.]
    >>> predictions = [0., 1., 2., 3., 4., 5.]
    >>> results = glue_metric.compute(predictions=predictions,
references=references)
    >>> print({"pearson": round(results["pearson"], 2), "spearmanr":
round(results["spearmanr"], 2)})
    {'pearson': 1.0, 'spearmanr': 1.0}

    >>> glue_metric = datasets.load_metric('glue', 'cola')
    >>> references = [0, 1]
```

```
    >>> predictions = [0, 1]
    >>> results = glue_metric.compute(predictions=predictions,
references=references)
    >>> print(results)
    {'matthews_correlation': 1.0}
```

```
#计算一个评价指标
predictions = [0, 1, 0]
references = [0, 1, 1]

final_score = metric.compute(predictions=predictions, references=references)

final_score
```

```
{'accuracy': 0.6666666666666666, 'f1': 0.6666666666666666}
```

5.pipeline

```
from transformers import pipeline

#文本分类
classifier = pipeline("sentiment-analysis")

result = classifier("I hate you")[0]
print(result)

result = classifier("I love you")[0]
print(result)
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-
english (https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english)
```

```
Downloading:   0%|          | 0.00/629 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/255M [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/48.0 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/226k [00:00<?, ?B/s]
```

```
{'label': 'NEGATIVE', 'score': 0.9991129040718079}
{'label': 'POSITIVE', 'score': 0.9998656511306763}
```

```
from transformers import pipeline

#阅读理解
question_answerer = pipeline("question-answering")

context = r"""
Extractive Question Answering is the task of extracting an answer from a text
given a question. An example of a
question answering dataset is the SQuAD dataset, which is entirely based on that
task. If you would like to fine-tune
a model on a SQuAD task, you may leverage the examples/pytorch/question-
answering/run_squad.py script.
"""

result = question_answerer(question="What is extractive question answering?",
                           context=context)
print(result)

result = question_answerer(
    question="What is a good example of a question answering dataset?",
    context=context)

print(result)
```

```
No model was supplied, defaulted to distilbert-base-cased-distilled-squad
(https://huggingface.co/distilbert-base-cased-distilled-squad)
```

```
Downloading:   0%|          | 0.00/473 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/249M [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/29.0 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/208k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/426k [00:00<?, ?B/s]
```

```
{'score': 0.6177273988723755, 'start': 34, 'end': 95, 'answer': 'the task of
extracting an answer from a text given a question'}
{'score': 0.5152313113212585, 'start': 148, 'end': 161, 'answer': 'SQuAD
dataset'}
```

```
from transformers import pipeline

#完形填空
unmasker = pipeline("fill-mask")

from pprint import pprint

sentence = 'HuggingFace is creating a <mask> that the community uses to solve
NLP tasks.'

unmasker(sentence)
```

```
No model was supplied, defaulted to distilroberta-base
(https://huggingface.co/distilroberta-base)
```

```
Downloading:   0%|          | 0.00/480 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/316M [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/878k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/446k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/1.29M [00:00<?, ?B/s]
```

```
[{'score': 0.17927563190460205,
  'token': 3944,
```

```
  'token_str': ' tool',
  'sequence': 'HuggingFace is creating a tool that the community uses to solve
NLP tasks.'},
 {'score': 0.11349444836378098,
  'token': 7208,
  'token_str': ' framework',
  'sequence': 'HuggingFace is creating a framework that the community uses to
solve NLP tasks.'},
 {'score': 0.052435252815485,
  'token': 5560,
  'token_str': ' library',
  'sequence': 'HuggingFace is creating a library that the community uses to solve
NLP tasks.'},
 {'score': 0.034935373812913895,
  'token': 8503,
  'token_str': ' database',
  'sequence': 'HuggingFace is creating a database that the community uses to
solve NLP tasks.'},
 {'score': 0.02860225923359394,
  'token': 17715,
  'token_str': ' prototype',
  'sequence': 'HuggingFace is creating a prototype that the community uses to
solve NLP tasks.'}]
```

```
from transformers import pipeline

#文本生成
text_generator = pipeline("text-generation")

text_generator("As far as I am concerned, I will",
               max_length=50,
               do_sample=False)
```

```
No model was supplied, defaulted to gpt2 (https://huggingface.co/gpt2)
```

```
Downloading:   0%|          | 0.00/665 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/523M [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/0.99M [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/446k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/1.29M [00:00<?, ?B/s]
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
[{'generated_text': 'As far as I am concerned, I will be the first to admit that I am not a fan of the idea of a "free market." I think that the idea of a free market is a bit of a stretch. I think that the idea'}]
```

```
from transformers import pipeline

#命名实体识别
ner_pipe = pipeline("ner")

sequence = """Hugging Face Inc. is a company based in New York City. Its headquarters are in DUMBO,
therefore very close to the Manhattan Bridge which is visible from the window."""

for entity in ner_pipe(sequence):
    print(entity)
```

```
No model was supplied, defaulted to dbmdz/bert-large-cased-finetuned-conll03-english (https://huggingface.co/dbmdz/bert-large-cased-finetuned-conll03-english)
```

```
Downloading:   0%|          | 0.00/998 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/1.24G [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/60.0 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/208k [00:00<?, ?B/s]
```

```
{'entity': 'I-ORG', 'score': 0.99957865, 'index': 1, 'word': 'Hu', 'start': 0,
'end': 2}
{'entity': 'I-ORG', 'score': 0.9909764, 'index': 2, 'word': '##gging', 'start':
2, 'end': 7}
{'entity': 'I-ORG', 'score': 0.9982224, 'index': 3, 'word': 'Face', 'start': 8,
'end': 12}
{'entity': 'I-ORG', 'score': 0.9994879, 'index': 4, 'word': 'Inc', 'start': 13,
'end': 16}
{'entity': 'I-LOC', 'score': 0.9994344, 'index': 11, 'word': 'New', 'start': 40,
'end': 43}
{'entity': 'I-LOC', 'score': 0.99931955, 'index': 12, 'word': 'York', 'start':
44, 'end': 48}
{'entity': 'I-LOC', 'score': 0.9993794, 'index': 13, 'word': 'City', 'start': 49,
'end': 53}
{'entity': 'I-LOC', 'score': 0.98625827, 'index': 19, 'word': 'D', 'start': 79,
'end': 80}
{'entity': 'I-LOC', 'score': 0.95142686, 'index': 20, 'word': '##UM', 'start':
80, 'end': 82}
{'entity': 'I-LOC', 'score': 0.933659, 'index': 21, 'word': '##BO', 'start': 82,
'end': 84}
{'entity': 'I-LOC', 'score': 0.9761654, 'index': 28, 'word': 'Manhattan',
'start': 114, 'end': 123}
{'entity': 'I-LOC', 'score': 0.9914629, 'index': 29, 'word': 'Bridge', 'start':
124, 'end': 130}
```

```python
from transformers import pipeline

#文本总结
summarizer = pipeline("summarization")

ARTICLE = """ New York (CNN)When Liana Barrientos was 23 years old, she got
married in Westchester County, New York.
A year later, she got married again in Westchester County, but to a different man
and without divorcing her first husband.
Only 18 days after that marriage, she got hitched yet again. Then, Barrientos
declared "I do" five more times, sometimes only within two weeks of each other.
In 2010, she married once more, this time in the Bronx. In an application for a
marriage license, she stated it was her "first and only" marriage.
Barrientos, now 39, is facing two criminal counts of "offering a false instrument
for filing in the first degree," referring to her false statements on the
2010 marriage license application, according to court documents.
Prosecutors said the marriages were part of an immigration scam.
On Friday, she pleaded not guilty at State Supreme Court in the Bronx, according
to her attorney, Christopher Wright, who declined to comment further.
After leaving court, Barrientos was arrested and charged with theft of service
and criminal trespass for allegedly sneaking into the New York subway through an
emergency exit, said Detective
Annette Markowski, a police spokeswoman. In total, Barrientos has been married 10
times, with nine of her marriages occurring between 1999 and 2002.
All occurred either in Westchester County, Long Island, New Jersey or the Bronx.
She is believed to still be married to four men, and at one time, she was married
to eight men at once, prosecutors say.
```

```
Prosecutors said the immigration scam involved some of her husbands, who filed
for permanent residence status shortly after the marriages.
Any divorces happened only after such filings were approved. It was unclear
whether any of the men will be prosecuted.
The case was referred to the Bronx District Attorney\'s Office by Immigration and
Customs Enforcement and the Department of Homeland Security\'s
Investigation Division. Seven of the men are from so-called "red-flagged"
countries, including Egypt, Turkey, Georgia, Pakistan and Mali.
Her eighth husband, Rashid Rajput, was deported in 2006 to his native Pakistan
after an investigation by the Joint Terrorism Task Force.
If convicted, Barrientos faces up to four years in prison.  Her next court
appearance is scheduled for May 18.
"""

summarizer(ARTICLE, max_length=130, min_length=30, do_sample=False)
```

```
No model was supplied, defaulted to sshleifer/distilbart-cnn-12-6
(https://huggingface.co/sshleifer/distilbart-cnn-12-6)
```

```
Downloading:   0%|          | 0.00/1.76k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/1.14G [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/26.0 [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/878k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/446k [00:00<?, ?B/s]
```

```
[{'summary_text': ' Liana Barrientos, 39, is charged with two counts of "offering
a false instrument for filing in the first degree" In total, she has been married
10 times, with nine of her marriages occurring between 1999 and 2002 . At one
time, she was married to eight men at once, prosecutors say .'}]
```

```
from transformers import pipeline

#翻译
translator = pipeline("translation_en_to_de")

sentence = "Hugging Face is a technology company based in New York and Paris"

translator(sentence, max_length=40)
```

```
No model was supplied, defaulted to t5-base (https://huggingface.co/t5-base)
```

```
Downloading:   0%|          | 0.00/1.18k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/850M [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/773k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/1.32M [00:00<?, ?B/s]
```

```
[{'translation_text': 'Hugging Face ist ein Technologieunternehmen mit Sitz in
New York und Paris.'}]
```

6.命名实体识别_中文

```
from transformers import AutoTokenizer

#加载分词器
tokenizer = AutoTokenizer.from_pretrained('hfl/rbt6')

print(tokenizer)

#分词测试
tokenizer.batch_encode_plus(
    [[
        '海', '钓', '比', '赛', '地', '点', '在', '厦', '门', '与', '金', '门', '之',
'间',
        '的', '海', '域', '。'
```

```
    ],
     [
         '这', '座', '依', '山', '傍', '水', '的', '博', '物', '馆', '由', '国',
'内', '一',
         '流', '的', '设', '计', '师', '主', '持', '设', '计', '，', '整', '个',
'建', '筑',
         '群', '精', '美', '而', '恢', '宏', '。'
     ]],
    truncation=True,
    padding=True,
    return_tensors='pt',
    is_split_into_words=True)
```

```
Asking to truncate to max_length but no maximum length is provided and the model
has no predefined maximum length. Default to no truncation.
```

```
PreTrainedTokenizerFast(name_or_path='hfl/rbt6', vocab_size=21128,
model_max_len=1000000000000000019884624838656, is_fast=True,
padding_side='right', truncation_side='right', special_tokens={'unk_token':
'[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]',
'mask_token': '[MASK]'})
```

```
{'input_ids': tensor([[ 101, 3862, 7157, 3683, 6612, 1765, 4157, 1762, 1336,
7305,  680, 7032,
         7305,  722, 7313, 4638, 3862, 1818,  511,  102,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0],
        [ 101, 6821, 2429,  898, 2255,  988, 3717, 4638, 1300, 4289, 7667, 4507,
         1744, 1079,  671, 3837, 4638, 6392, 6369, 2360,  712, 2898, 6392, 6369,
         8024, 3146,  702, 2456, 5029, 5408, 5125, 5401, 5445, 2612, 2131,  511,
          102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]), 'attention_mask': tensor([[1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

```
import torch
from datasets import load_dataset, load_from_disk


class Dataset(torch.utils.data.Dataset):
    def __init__(self, split):
        #names=['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC']
```

```
        #在线加载数据集
        #dataset = load_dataset(path='peoples_daily_ner', split=split)

        #离线加载数据集
        dataset = load_from_disk(dataset_path='./data')[split]

        #过滤掉太长的句子
        def f(data):
            return len(data['tokens']) <= 512 - 2

        dataset = dataset.filter(f)

        self.dataset = dataset

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, i):
        tokens = self.dataset[i]['tokens']
        labels = self.dataset[i]['ner_tags']

        return tokens, labels


dataset = Dataset('train')

tokens, labels = dataset[0]

len(dataset), tokens, labels
```

```
Loading cached processed dataset at data/train/cache-534d84a68ea10ebc.arrow
```

```
(20852,
 ['海',
  '钓',
  '比',
  '赛',
  '地',
  '点',
  '在',
  '厦',
  '门',
  '与',
  '金',
  '门',
  '之',
  '间',
  '的',
  '海',
  '域',
```

```
  '。'],
 [0, 0, 0, 0, 0, 0, 0, 5, 6, 0, 5, 6, 0, 0, 0, 0, 0, 0])
```

```
#数据整理函数
def collate_fn(data):
    tokens = [i[0] for i in data]
    labels = [i[1] for i in data]

    inputs = tokenizer.batch_encode_plus(tokens,
                                         truncation=True,
                                         padding=True,
                                         return_tensors='pt',
                                         is_split_into_words=True)

    lens = inputs['input_ids'].shape[1]

    for i in range(len(labels)):
        labels[i] = [7] + labels[i]
        labels[i] += [7] * lens
        labels[i] = labels[i][:lens]

    return inputs, torch.LongTensor(labels)


#数据加载器
loader = torch.utils.data.DataLoader(dataset=dataset,
                                     batch_size=16,
                                     collate_fn=collate_fn,
                                     shuffle=True,
                                     drop_last=True)

#查看数据样例
for i, (inputs, labels) in enumerate(loader):
    break

print(len(loader))
print(tokenizer.decode(inputs['input_ids'][0]))
print(labels[0])

for k, v in inputs.items():
    print(k, v.shape)
```

```
1303
[CLS] 马 来 西 亚 一 些 大 公 司 从 2 月 开 始 ， 展 开 了 一 场 为 期 半 年 的 [UNK] 攻 关 战 [UNK] ， 计 划 花 费 3 0 0 万 至 5 0 0 万 美 元 ， 在 全 球 范 围 做 广 告 。
[SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD]
tensor([7, 5, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
        7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])
input_ids torch.Size([16, 88])
token_type_ids torch.Size([16, 88])
attention_mask torch.Size([16, 88])
```

```python
from transformers import AutoModel

#加载预训练模型
pretrained = AutoModel.from_pretrained('hfl/rbt6')

#统计参数量
print(sum(i.numel() for i in pretrained.parameters()) / 10000)

#模型试算
#[b, lens] -> [b, lens, 768]
pretrained(**inputs).last_hidden_state.shape
```

```
Some weights of the model checkpoint at hfl/rbt6 were not used when initializing
BertModel: ['cls.seq_relationship.weight',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.decoder.weight',
'cls.predictions.bias', 'cls.predictions.decoder.bias',
'cls.seq_relationship.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

```
5974.0416
```

```
torch.Size([16, 88, 768])
```

```
#定义下游模型
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.tuneing = False
        self.pretrained = None

        self.rnn = torch.nn.GRU(768, 768,batch_first=True)
        self.fc = torch.nn.Linear(768, 8)

    def forward(self, inputs):
        if self.tuneing:
            out = self.pretrained(**inputs).last_hidden_state
        else:
            with torch.no_grad():
                out = pretrained(**inputs).last_hidden_state

        out, _ = self.rnn(out)

        out = self.fc(out).softmax(dim=2)

        return out

    def fine_tuneing(self, tuneing):
        self.tuneing = tuneing
        if tuneing:
            for i in pretrained.parameters():
                i.requires_grad = True

            pretrained.train()
            self.pretrained = pretrained
        else:
            for i in pretrained.parameters():
                i.requires_grad_(False)

            pretrained.eval()
            self.pretrained = None


model = Model()

model(inputs).shape
```

```
torch.Size([16, 88, 8])
```

```
#对计算结果和label变形,并且移除pad
def reshape_and_remove_pad(outs, labels, attention_mask):
    #变形,便于计算loss
    #[b, lens, 8] -> [b*lens, 8]
    outs = outs.reshape(-1, 8)
```

```
    #[b, lens] -> [b*lens]
    labels = labels.reshape(-1)

    #忽略对pad的计算结果
    #[b, lens] -> [b*lens - pad]
    select = attention_mask.reshape(-1) == 1
    outs = outs[select]
    labels = labels[select]

    return outs, labels


reshape_and_remove_pad(torch.randn(2, 3, 8), torch.ones(2, 3),
                       torch.ones(2, 3))
```

```
(tensor([[ 0.1653,  0.9074,  1.1993, -0.0515, -0.4700,  0.3700,  0.0175,
-1.0165],
         [-0.5450,  1.7236, -0.1515, -1.9181,  0.5940, -0.5028,  1.6496,
1.7369],
         [-0.1785, -0.5002, -0.9051,  0.2528, -0.9384, -0.4375, -1.0452,
0.6255],
         [ 0.2369, -0.8779,  0.3852,  2.3229,  0.9584, -0.9273,  1.4566,
-0.0438],
         [ 0.0610,  0.2239,  0.1392,  0.3481,  2.3022, -0.6476, -1.1643,
0.4135],
         [ 0.7769, -0.5040,  0.0106, -0.3306, -0.6428, -1.5164,  0.9515,
0.7806]]),
 tensor([1., 1., 1., 1., 1., 1.]))
```

```
#获取正确数量和总数
def get_correct_and_total_count(labels, outs):
    #[b*lens, 8] -> [b*lens]
    outs = outs.argmax(dim=1)
    correct = (outs == labels).sum().item()
    total = len(labels)

    #计算除了0以外元素的正确率,因为0太多了,包括的话,正确率很容易虚高
    select = labels != 0
    outs = outs[select]
    labels = labels[select]
    correct_content = (outs == labels).sum().item()
    total_content = len(labels)

    return correct, total, correct_content, total_content


get_correct_and_total_count(torch.ones(16), torch.randn(16, 8))
```

```
(2, 16, 2, 16)
```

```
from transformers import AdamW


#训练
def train(epochs):
    lr = 2e-5 if model.tuneing else 5e-4

    #训练
    optimizer = AdamW(model.parameters(), lr=lr)
    criterion = torch.nn.CrossEntropyLoss()

    model.train()
    for epoch in range(epochs):
        for step, (inputs, labels) in enumerate(loader):
            #模型计算
            #[b, lens] -> [b, lens, 8]
            outs = model(inputs)

            #对outs和label变形,并且移除pad
            #outs -> [b, lens, 8] -> [c, 8]
            #labels -> [b, lens] -> [c]
            outs, labels = reshape_and_remove_pad(outs, labels,
                                                  inputs['attention_mask'])

            #梯度下降
            loss = criterion(outs, labels)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

            if step % 50 == 0:
                counts = get_correct_and_total_count(labels, outs)

                accuracy = counts[0] / counts[1]
                accuracy_content = counts[2] / counts[3]

                print(epoch, step, loss.item(), accuracy, accuracy_content)

        torch.save(model, 'model/命名实体识别_中文.model')


model.fine_tuneing(False)
print(sum(p.numel() for p in model.parameters()) / 10000)
#train(1)
```

```
354.9704
```

```
model.fine_tuneing(True)
print(sum(p.numel() for p in model.parameters()) / 10000)
#train(2)
```

```
6329.012
```

```
#测试
def test():
    model_load = torch.load('model/命名实体识别_中文.model')
    model_load.eval()

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('validation'),
                                              batch_size=128,
                                              collate_fn=collate_fn,
                                              shuffle=True,
                                              drop_last=True)

    correct = 0
    total = 0

    correct_content = 0
    total_content = 0

    for step, (inputs, labels) in enumerate(loader_test):
        if step == 5:
            break
        print(step)

        with torch.no_grad():
            #[b, lens] -> [b, lens, 8] -> [b, lens]
            outs = model_load(inputs)

        #对outs和label变形,并且移除pad
        #outs -> [b, lens, 8] -> [c, 8]
        #labels -> [b, lens] -> [c]
        outs, labels = reshape_and_remove_pad(outs, labels,
                                              inputs['attention_mask'])

        counts = get_correct_and_total_count(labels, outs)
        correct += counts[0]
        total += counts[1]
        correct_content += counts[2]
        total_content += counts[3]

    print(correct / total, correct_content / total_content)


test()
```

```
Loading cached processed dataset at data/validation/cache-80ee7b679fd38e82.arrow
```

```
0
1
2
3
4
0.9907604360542409 0.9553249097472925
```

```python
#测试
def predict():
    model_load = torch.load('model/命名实体识别_中文.model')
    model_load.eval()

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('validation'),
                                              batch_size=32,
                                              collate_fn=collate_fn,
                                              shuffle=True,
                                              drop_last=True)

    for i, (inputs, labels) in enumerate(loader_test):
        break

    with torch.no_grad():
        #[b, lens] -> [b, lens, 8] -> [b, lens]
        outs = model_load(inputs).argmax(dim=2)

    for i in range(32):
        #移除pad
        select = inputs['attention_mask'][i] == 1
        input_id = inputs['input_ids'][i, select]
        out = outs[i, select]
        label = labels[i, select]

        #输出原句子
        print(tokenizer.decode(input_id).replace(' ', ''))

        #输出tag
        for tag in [label, out]:
            s = ''
            for j in range(len(tag)):
                if tag[j] == 0:
                    s += '.'
                    continue
                s += tokenizer.decode(input_id[j])
                s += str(tag[j].item())

            print(s)
        print('============================')


predict()
```

```
Loading cached processed dataset at data/validation/cache-80ee7b679fd38e82.arrow
```

```
[CLS]胡老介绍说：菊花的品种不一样，叶子也不相同，有三岐两缺的、五岐四缺的，多到七岐五缺的，其中以五岐四缺最为常见；根据叶子的形状可以区别花朵的类型，叶裂缺刻圆钝的多为宽瓣花类，叶裂缺刻尖锐的多为细瓣花类，而叶背中肋有深色纹的以紫色花为多。[SEP]
[CLS]7胡
1·····[SEP]7
[CLS]7胡
1·····[SEP]7
==========
[CLS]他自幼学习书法，八九岁时即闻名乡里，被誉为[UNK]神童[UNK]，少年时代被称为[UNK]东乡才子[UNK]。[SEP]
[CLS]7·····东5乡6····[SEP]7
[CLS]7·····东5乡6····[SEP]7
==========
[CLS]周涛是以诗人的气质写散文，以『游牧』的眼光审视历史和文化，极富艺术个性。[SEP]
[CLS]7周1涛2·····[SEP]7
[CLS]7周1涛2·····[SEP]7
==========
[CLS]皇天不负苦心人，不久，孩子放学回家老远就喊着冲进门来了：[UNK]爸[UNK][UNK][UNK]爸！[UNK][SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]拉特纳亚克议长还向江泽民主席介绍了斯里兰卡议会和国内的情况，并转达了库马拉通加总统对他的亲切问候。[SEP]
[CLS]7拉1特2纳2亚2克2····江1泽2民2·····斯3里4兰4卡4议4会4·····库1马2拉2通2加2·····[SEP]7
[CLS]7拉1特2纳2亚2克2····江1泽2民2·····斯3里4兰4卡4议4会4·····库1马2拉2通2加2·····[SEP]7
==========
[CLS]他说，中国共产党领导的多党合作和政治协商制度，是中国的基本政治制度，中国人民政治协商会议则是实现这个制度的组织形式。[SEP]
[CLS]7···中3国4共4产4党4·····中5国6·····中3国4人4民4政4治4协4商4会4议4·····[SEP]7
[CLS]7···中3国4共4产4党4·····中5国6·····中3国4人4民4·治4协4·····[SEP]7
==========
[CLS]只有当时当势最大之事，只有万千人利益共存共在之事，众目所注，万念归一，其事成而社会民族喜，其事败而社会民族悲。[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]她的步法与腕上功夫的完美结合更是独特，看她的比赛往往使人恍如欣赏芭蕾舞表演，而忘却这是竞争激烈的赛场[UNK][UNK]本届汤尤杯赛揭幕战，她首战英国名将曼尔，仅用15分钟便以11：0、11：1轻松利落地为印尼队赢得一个开门红。[SEP]
[CLS]7·····汤1尤1·····英5国6··曼1尔2·····印3尼4队4·····[SEP]7
[CLS]7·····汤1尤2·····英5国6··曼1尔2·····印3尼4队4·····[SEP]7
==========
```

[CLS]加快政府信息资源的数字化、网络化进程，建设高性能政府信息网络，提高政府工作效率和决策质量，适应快速变化的外部世界，提高政府透明度，为反腐败和廉政建设创造物质条件。[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]●英国副首相和墨西哥外长将访华新华社北京6月25日电外交部发言人唐国强今天在记者招待会上宣布：应国务院副总理吴邦国的邀请，大不列颠及北爱尔兰联合王国副首相约翰·普雷斯科特将于7月1日至7日对中国进行正式访问。[SEP]
[CLS]7·英5国6····墨5西6哥6····华5新3华4社4北5京6·····外3交4部4···唐1国2强2·····国3务4院4···吴1邦2国2····大5不6列6颠6及6北6爱6尔6兰6联6合6王6国6···约1翰2·2普2雷2斯2科2特2·····中5国6·····[SEP]7
[CLS]7·英5国6····墨5西6哥6····华5新3华4社4北5京6·····外3交4部4···唐1国2强2·····国3务4院4···吴1邦2国2····大5不6列6颠6·北5爱6尔6兰6联4合6王6国6···约1翰2·2普2雷2斯2科2特2·····中5国6·····[SEP]7
==========
[CLS]本来会议并未邀请南平镇参加，但袁正军获悉后，火速派人赶往长沙联系。[SEP]
[CLS]7·····南5平6镇6····袁1正2军2·····长5沙6···[SEP]7
[CLS]7·····南5平6镇6····袁1正2军2·····长5沙6···[SEP]7
==========
[CLS]针对平安承保的沿江、沿湖、低洼及历史受淹标的情况，办事处抽调专人对这些标的进行清点和检查，并提出整改意见及汛期预防和标的转移的具体方案，先后对8家重点单位进行了检查，以此引起各单位领导的重视和支持。[SEP]
[CLS]7··平3安4·····办3事4处
4·····[SEP]7
[CLS]7·····处
4·····[SEP]7
==========
[CLS]我国大城市普遍采用复式计次制，一些边远地区采用包月制。[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]据有关专家分析，印度的股市和汇市纷纷下跌的主要原因是，除了美国和日本等国家因印度进行核试验而对其采取经济制裁措施外，世界银行和亚洲开发银行等决定停止向印度提供贷款。[SEP]
[CLS]7·····印5度6·····美5国6·日5本6····印5度6·····世3界4银4行4·亚3洲4开4发4银4行4·····印5度6·····[SEP]7
[CLS]7·····印5度6·····美5国6·日5本6····印5度6·····世3界4银4行4·亚3洲4开4发4银4行4·····印5度6·····[SEP]7
==========
[CLS]1996年初，石渠县发生特大雪灾，全县牲畜死亡率超过37%，而搞了[UNK]人草畜[UNK]配套建设的牧户牲畜死亡率平均不到10%。[SEP]
[CLS]7·····石5渠6县6·····[SEP]7
[CLS]7·····石5渠6县6·····[SEP]7
==========
[CLS]1984年他刚到海尔的前身[UNK][UNK][UNK]青岛电冰箱总厂时，面对的是一个发不出工资、濒临倒闭的烂摊子。[SEP]
[CLS]7·····海3尔4·····青3岛4电4冰4箱4总4厂4·····[SEP]7
[CLS]7·····海3尔4·····青3岛4电4冰4箱4总4厂4·····[SEP]7
==========
[CLS]本报北京6月17日讯新华社记者王黎、本报记者董洪亮报道：记者日前从教育部有关部门了解到，天津市在教育经费的投入上，多年来一直保持高于财政经常性收入增长的比例，有不少区县教育预算占地方财政一半以上；同时按政策积极组织社会力量筹措资金，在一定程度上弥补了教育经费的不足。[SEP]

[CLS]7··北5京6·····新3华4社4··王1黎2·····董1洪2亮2·····教3育4部4·····天5津6市6·····[SEP]7
[CLS]7··北5京6·····新3华4社4··王1黎2·····董1洪2亮2·····教3育4部4·····天5津6市6·····[SEP]7
==========
[CLS]吴健雄教授是当代第一流的实验原子核物理学家。[SEP]
[CLS]7吴1健2雄2·····[SEP]7
[CLS]7吴1健2雄2·····[SEP]7
==========
[CLS]1996年，为了改变海城的城市形象，市委、市政府决定对老城区进行重新规划和改造，为了解决改造过程中的一系列难题，我们一方面邀请著名的专家学者论证，一方面通过新闻媒体在全市开展了[UNK]市民建言[UNK]活动，结果收到市民的合理化建议60多条，为城市改造工程的顺利实施创造了先决条件。[SEP]
[CLS]7·····海5城6·····市3委4·····[SEP]7
[CLS]7·····海5城6·····市3委4·····[SEP]7
==========
[CLS]然而，前方吃紧，后方紧吃，吃在成都，乐在官场。[SEP]
[CLS]7·····成5都6·····[SEP]7
[CLS]7·····成5都6·····[SEP]7
==========
[CLS]1、每次飞行前，空中乘务员都要提前着装整齐在签到室里接受监督检查。[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]成立大会上，经济日报社社长徐心华、光明日报总编辑王晨都表示，要以组建报业集团为契机，大胆探索，勇于创新，加快改革开放步伐，为推动有中国特色的社会主义现代化报业集团的健康发展作出贡献。[SEP]
[CLS]7·····经3济4日4报4社4··徐1心2华2·光3明4日4报4···王1晨2·····中5国6·····[SEP]7
[CLS]7·····经3济4日4报4社4··徐1心2华2·光3明4日4报4···王1晨2·····中5国6·····[SEP]7
==========
[CLS]我们沿着崎岖的山路来到王永祥跟前时，他正伫立在丛林中间，凝视着充满生机的一片绿色。[SEP]
[CLS]7·····王1永2祥2·····[SEP]7
[CLS]7·····王1永2祥2·····[SEP]7
==========
[CLS]全国人大常委会法工委研究室吴高盛认为：目前我国已有了一套包括《反不正当竞争法》、《产品质量法》、《消费者权益保护法》等等在内的比较完备的市场竞争规则，但是还需要完善，还存在有法不依、执法不严的现象。[SEP]
[CLS]7全3国4人4大4常4委4会4法4工4委4研4究4室4吴1高2盛2·····[SEP]7
[CLS]7全3国4人4大4常4委4会4法4工4委4研4究4室4吴1高2盛2·····[SEP]7
==========
[CLS]但是，对于中国汽车产业来说，丰田是强有力的竞争对手之一。[SEP]
[CLS]7·····中5国6·····丰3田4·····[SEP]7
[CLS]7·····中5国6·····丰3田4·····[SEP]7
==========

```
[CLS]为了开辟再就业的途径，不讲清[UNK]转变观念天地宽[UNK]的道理怎么行呢？[SEP]
[CLS]7·····························[SEP]7
[CLS]7·····························[SEP]7
==========
[CLS]为消除西方一些人在西藏问题上的偏见，卓科达先生在长期研究的基础上，撰写了有关中国宗教问题的专著。[SEP]
[CLS]7·····西5藏6·····卓1科2达2·····中5国6·····[SEP]7
[CLS]7·····西5藏6·····卓1科2达2·····中5国6·····[SEP]7
==========
[CLS]经过六届人大常委会第四次、第五次会议审议，决定提请第六届全国人大第二次会议审议，会上我作了关于《中华人民共和国民族区域自治法草案》的说明。[SEP]
[CLS]7··六3届4人4大4常4委4会4·····第3六4届4全4国4人4大4·····中5华6人6民6共6和6国6·····[SEP]7
[CLS]7··六3届4人4大4常4委4会4·····第3六4届4全4国4人4大4·····中5华6人6民6共6和6国6·····[SEP]7
==========
[CLS]不过那种乐于把诗歌的公共性乃至战斗性视为其美学要素的声音也没有停息。[SEP]
[CLS]7·····························[SEP]7
[CLS]7·····························[SEP]7
==========
[CLS]不久前的一天上午，刚毕业于曼谷万颂绿叻差博学院企业管理系的少女玛裕勒和诗丽婉到吞武里一带求职未果。[SEP]
[CLS]7·····曼3谷4万4颂4绿4叻4差4博4学4院4企4业4管4理4系4···玛1裕2勒2·诗1丽2婉2·吞5武6里6·····[SEP]7
[CLS]7·····曼5谷6万4颂4绿4叻4差4博4学4院4企4业4管4理4系4···玛1裕2勒2·诗1丽2婉2·吞5武6里6·····[SEP]7
==========
[CLS]上海工部局一两试铸银币是存世极为稀少的钱币品种。[SEP]
[CLS]7上3海4工4部4局4·····[SEP]7
[CLS]7上3海4工4部4局4·····[SEP]7
==========
[CLS]5月23日是墨西哥全国防疫接种卫生周第一天。[SEP]
[CLS]7·····墨5西6哥6·····[SEP]7
[CLS]7·····墨5西6哥6·····[SEP]7
==========
```

```python
from transformers import AutoTokenizer

#加载分词器
tokenizer = AutoTokenizer.from_pretrained('hfl/rbt6')

print(tokenizer)

#分词测试
tokenizer.batch_encode_plus(
    [[
        '海', '钓', '比', '赛', '地', '点', '在', '厦', '门', '与', '金', '门', '之',
'间',
        '的', '海', '域', '。'
    ],
     [
         '这', '座', '依', '山', '傍', '水', '的', '博', '物', '馆', '由', '国',
'内', '一',
```

```
        '流', '的', '设', '计', '师', '主', '持', '设', '计', ', ', '整', '个',
'建', '筑',
        '群', '精', '美', '而', '恢', '宏', '。'
     ]],
    truncation=True,
    padding=True,
    return_tensors='pt',
    is_split_into_words=True)
```

```
Asking to truncate to max_length but no maximum length is provided and the model
has no predefined maximum length. Default to no truncation.
```

```
PreTrainedTokenizerFast(name_or_path='hfl/rbt6', vocab_size=21128,
model_max_len=1000000000000000019884624838656, is_fast=True,
padding_side='right', truncation_side='right', special_tokens={'unk_token':
'[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]',
'mask_token': '[MASK]'})
```

```
{'input_ids': tensor([[ 101, 3862, 7157, 3683, 6612, 1765, 4157, 1762, 1336,
7305,  680, 7032,
         7305,  722, 7313, 4638, 3862, 1818,  511,  102,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0],
        [ 101, 6821, 2429,  898, 2255,  988, 3717, 4638, 1300, 4289, 7667, 4507,
         1744, 1079,  671, 3837, 4638, 6392, 6369, 2360,  712, 2898, 6392, 6369,
         8024, 3146,  702, 2456, 5029, 5408, 5125, 5401, 5445, 2612, 2131,  511,
          102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]), 'attention_mask': tensor([[1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

```
import torch
from datasets import load_dataset, load_from_disk


class Dataset(torch.utils.data.Dataset):
    def __init__(self, split):
        #names=['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC']

        #在线加载数据集
        #dataset = load_dataset(path='peoples_daily_ner', split=split)
```

```
        #离线加载数据集
        dataset = load_from_disk(dataset_path='./data')[split]

        #过滤掉太长的句子
        def f(data):
            return len(data['tokens']) <= 512 - 2

        dataset = dataset.filter(f)

        self.dataset = dataset

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, i):
        tokens = self.dataset[i]['tokens']
        labels = self.dataset[i]['ner_tags']

        return tokens, labels


dataset = Dataset('train')

tokens, labels = dataset[0]

len(dataset), tokens, labels
```

```
Loading cached processed dataset at data/train/cache-534d84a68ea10ebc.arrow
```

```
(20852,
 ['海',
  '钓',
  '比',
  '赛',
  '地',
  '点',
  '在',
  '厦',
  '门',
  '与',
  '金',
  '门',
  '之',
  '间',
  '的',
  '海',
  '域',
  '。'],
 [0, 0, 0, 0, 0, 0, 0, 5, 6, 0, 5, 6, 0, 0, 0, 0, 0, 0])
```

```python
#数据整理函数
def collate_fn(data):
    tokens = [i[0] for i in data]
    labels = [i[1] for i in data]

    inputs = tokenizer.batch_encode_plus(tokens,
                                         truncation=True,
                                         padding=True,
                                         return_tensors='pt',
                                         is_split_into_words=True)

    lens = inputs['input_ids'].shape[1]

    for i in range(len(labels)):
        labels[i] = [7] + labels[i]
        labels[i] += [7] * lens
        labels[i] = labels[i][:lens]

    return inputs, torch.LongTensor(labels)


#数据加载器
loader = torch.utils.data.DataLoader(dataset=dataset,
                                     batch_size=16,
                                     collate_fn=collate_fn,
                                     shuffle=True,
                                     drop_last=True)

#查看数据样例
for i, (inputs, labels) in enumerate(loader):
    break

print(len(loader))
print(tokenizer.decode(inputs['input_ids'][0]))
print(labels[0])

for k, v in inputs.items():
    print(k, v.shape)
```

```
1303
[CLS] 马 来 西 亚 一 些 大 公 司 从 2 月 开 始 ， 展 开 了 一 场 为 期 半 年 的 [UNK] 攻
关 战 [UNK] ， 计 划 花 费 3 0 0 万 至 5 0 0 万 美 元 ， 在 全 球 范 围 做 广 告 。
[SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD]
tensor([7, 5, 6, 6, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
        7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])
input_ids torch.Size([16, 88])
token_type_ids torch.Size([16, 88])
attention_mask torch.Size([16, 88])
```

```
from transformers import AutoModel

#加载预训练模型
pretrained = AutoModel.from_pretrained('hfl/rbt6')

#统计参数量
print(sum(i.numel() for i in pretrained.parameters()) / 10000)

#模型试算
#[b, lens] -> [b, lens, 768]
pretrained(**inputs).last_hidden_state.shape
```

```
Some weights of the model checkpoint at hfl/rbt6 were not used when initializing
BertModel: ['cls.seq_relationship.weight',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.decoder.weight',
'cls.predictions.bias', 'cls.predictions.decoder.bias',
'cls.seq_relationship.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

```
5974.0416
```

```
torch.Size([16, 88, 768])
```

```
#定义下游模型
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.tuneing = False
        self.pretrained = None

        self.rnn = torch.nn.GRU(768, 768,batch_first=True)
        self.fc = torch.nn.Linear(768, 8)

    def forward(self, inputs):
        if self.tuneing:
            out = self.pretrained(**inputs).last_hidden_state
        else:
```

```
            with torch.no_grad():
                out = pretrained(**inputs).last_hidden_state

        out, _ = self.rnn(out)

        out = self.fc(out).softmax(dim=2)

        return out

    def fine_tuneing(self, tuneing):
        self.tuneing = tuneing
        if tuneing:
            for i in pretrained.parameters():
                i.requires_grad = True

            pretrained.train()
            self.pretrained = pretrained
        else:
            for i in pretrained.parameters():
                i.requires_grad_(False)

            pretrained.eval()
            self.pretrained = None


model = Model()

model(inputs).shape
```

```
torch.Size([16, 88, 8])
```

```
#对计算结果和label变形,并且移除pad
def reshape_and_remove_pad(outs, labels, attention_mask):
    #变形,便于计算loss
    #[b, lens, 8] -> [b*lens, 8]
    outs = outs.reshape(-1, 8)
    #[b, lens] -> [b*lens]
    labels = labels.reshape(-1)

    #忽略对pad的计算结果
    #[b, lens] -> [b*lens - pad]
    select = attention_mask.reshape(-1) == 1
    outs = outs[select]
    labels = labels[select]

    return outs, labels


reshape_and_remove_pad(torch.randn(2, 3, 8), torch.ones(2, 3),
                       torch.ones(2, 3))
```

```
(tensor([[ 0.1653,  0.9074,  1.1993, -0.0515, -0.4700,  0.3700,  0.0175,
-1.0165],
         [-0.5450,  1.7236, -0.1515, -1.9181,  0.5940, -0.5028,  1.6496,
1.7369],
         [-0.1785, -0.5002, -0.9051,  0.2528, -0.9384, -0.4375, -1.0452,
0.6255],
         [ 0.2369, -0.8779,  0.3852,  2.3229,  0.9584, -0.9273,  1.4566,
-0.0438],
         [ 0.0610,  0.2239,  0.1392,  0.3481,  2.3022, -0.6476, -1.1643,
0.4135],
         [ 0.7769, -0.5040,  0.0106, -0.3306, -0.6428, -1.5164,  0.9515,
0.7806]]),
 tensor([1., 1., 1., 1., 1., 1.]))
```

```python
#获取正确数量和总数
def get_correct_and_total_count(labels, outs):
    #[b*lens, 8] -> [b*lens]
    outs = outs.argmax(dim=1)
    correct = (outs == labels).sum().item()
    total = len(labels)

    #计算除了0以外元素的正确率,因为0太多了,包括的话,正确率很容易虚高
    select = labels != 0
    outs = outs[select]
    labels = labels[select]
    correct_content = (outs == labels).sum().item()
    total_content = len(labels)

    return correct, total, correct_content, total_content


get_correct_and_total_count(torch.ones(16), torch.randn(16, 8))
```

```
(2, 16, 2, 16)
```

```python
from transformers import AdamW


#训练
def train(epochs):
    lr = 2e-5 if model.tuneing else 5e-4

    #训练
    optimizer = AdamW(model.parameters(), lr=lr)
    criterion = torch.nn.CrossEntropyLoss()
```

```
    model.train()
    for epoch in range(epochs):
        for step, (inputs, labels) in enumerate(loader):
            #模型计算
            #[b, lens] -> [b, lens, 8]
            outs = model(inputs)

            #对outs和label变形,并且移除pad
            #outs -> [b, lens, 8] -> [c, 8]
            #labels -> [b, lens] -> [c]
            outs, labels = reshape_and_remove_pad(outs, labels,
                                                  inputs['attention_mask'])

            #梯度下降
            loss = criterion(outs, labels)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

            if step % 50 == 0:
                counts = get_correct_and_total_count(labels, outs)

                accuracy = counts[0] / counts[1]
                accuracy_content = counts[2] / counts[3]

                print(epoch, step, loss.item(), accuracy, accuracy_content)

        torch.save(model, 'model/命名实体识别_中文.model')


model.fine_tuneing(False)
print(sum(p.numel() for p in model.parameters()) / 10000)
#train(1)
```

```
354.9704
```

```
model.fine_tuneing(True)
print(sum(p.numel() for p in model.parameters()) / 10000)
#train(2)
```

```
6329.012
```

```
#测试
def test():
    model_load = torch.load('model/命名实体识别_中文.model')
    model_load.eval()

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('validation'),
```

```
                                           batch_size=128,
                                           collate_fn=collate_fn,
                                           shuffle=True,
                                           drop_last=True)

    correct = 0
    total = 0

    correct_content = 0
    total_content = 0

    for step, (inputs, labels) in enumerate(loader_test):
        if step == 5:
            break
        print(step)

        with torch.no_grad():
            #[b, lens] -> [b, lens, 8] -> [b, lens]
            outs = model_load(inputs)

        #对outs和label变形,并且移除pad
        #outs -> [b, lens, 8] -> [c, 8]
        #labels -> [b, lens] -> [c]
        outs, labels = reshape_and_remove_pad(outs, labels,
                                              inputs['attention_mask'])

        counts = get_correct_and_total_count(labels, outs)
        correct += counts[0]
        total += counts[1]
        correct_content += counts[2]
        total_content += counts[3]

    print(correct / total, correct_content / total_content)


test()
```

```
Loading cached processed dataset at data/validation/cache-80ee7b679fd38e82.arrow
```

```
0
1
2
3
4
0.9907604360542409 0.9553249097472925
```

```
#测试
def predict():
    model_load = torch.load('model/命名实体识别_中文.model')
    model_load.eval()

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('validation'),
```

```
                                           batch_size=32,
                                           collate_fn=collate_fn,
                                           shuffle=True,
                                           drop_last=True)

    for i, (inputs, labels) in enumerate(loader_test):
        break

    with torch.no_grad():
        #[b, lens] -> [b, lens, 8] -> [b, lens]
        outs = model_load(inputs).argmax(dim=2)

    for i in range(32):
        #移除pad
        select = inputs['attention_mask'][i] == 1
        input_id = inputs['input_ids'][i, select]
        out = outs[i, select]
        label = labels[i, select]

        #输出原句子
        print(tokenizer.decode(input_id).replace(' ', ''))

        #输出tag
        for tag in [label, out]:
            s = ''
            for j in range(len(tag)):
                if tag[j] == 0:
                    s += '.'
                    continue
                s += tokenizer.decode(input_id[j])
                s += str(tag[j].item())

            print(s)
        print('============================')


predict()
```

```
Loading cached processed dataset at data/validation/cache-80ee7b679fd38e82.arrow
```

```
[CLS]胡老介绍说：菊花的品种不一样，叶子也不相同，有三岐两缺的、五岐四缺的，多到七岐五缺的，其中以五岐四缺最为常见；根据叶子的形状可以区别花朵的类型，叶裂缺刻圆钝的多为宽瓣花类，叶裂缺刻尖锐的多为细瓣花类，而叶背中肋有深色纹的以紫色花为多。[SEP]
[CLS]7胡
1.....[SEP]7
[CLS]7胡
1.....[SEP]7
============================
[CLS]他自幼学习书法，八九岁时即闻名乡里，被誉为[UNK]神童[UNK]，少年时代被称为[UNK]东乡才子[UNK]。[SEP]
[CLS]7.....东5乡6....[SEP]7
[CLS]7.....东5乡6....[SEP]7
```

==========
[CLS]周涛是以诗人的气质写散文，以『游牧』的眼光审视历史和文化，极富艺术个性。[SEP]
[CLS]7周1涛2·····[SEP]7
[CLS]7周1涛2·····[SEP]7
==========
[CLS]皇天不负苦心人，不久，孩子放学回家老远就喊着冲进门来了：[UNK]爸[UNK][UNK][UNK]爸！[UNK][SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]拉特纳亚克议长还向江泽民主席介绍了斯里兰卡议会和国内的情况，并转达了库马拉通加总统对他的亲切问候。[SEP]
[CLS]7拉1特2纳2亚2克2····江1泽2民2·····斯3里4兰4卡4议4会4·····库1马2拉2通2加2·····[SEP]7
[CLS]7拉1特2纳2亚2克2····江1泽2民2·····斯3里4兰4卡4议4会4·····库1马2拉2通2加2·····[SEP]7
==========
[CLS]他说，中国共产党领导的多党合作和政治协商制度，是中国的基本政治制度，中国人民政治协商会议则是实现这个制度的组织形式。[SEP]
[CLS]7···中3国4共4产4党4·····中5国6·····中3国4人4民4政4治4协4商4会4议4·····[SEP]7
[CLS]7···中3国4共4产4党4·····中5国6·····中3国4人4民4·治4协4·····[SEP]7
==========
[CLS]只有当时当势最大之事，只有万千人利益共存共在之事，众目所注，万念归一，其事成而社会民族喜，其事败而社会民族悲。[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]她的步法与腕上功夫的完美结合更是独特，看她的比赛往往使人恍如欣赏芭蕾舞表演，而忘却这是竞争激烈的赛场[UNK][UNK]本届汤尤杯赛揭幕战，她首战英国名将曼尔，仅用15分钟便以11：0、11：1轻松利落地为印尼队赢得一个开门红。[SEP]
[CLS]7·····汤1尤1·····英5国6··曼1尔2·····印3尼4队4·····[SEP]7
[CLS]7·····汤1尤2·····英5国6··曼1尔2·····印3尼4队4·····[SEP]7
==========
[CLS]加快政府信息资源的数字化、网络化进程，建设高性能政府信息网络，提高政府工作效率和决策质量，适应快速变化的外部世界，提高政府透明度，为反腐败和廉政建设创造物质条件。[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]●英国副首相和墨西哥外长将访华新华社北京6月25日电外交部发言人唐国强今天在记者招待会上宣布：应国务院副总理吴邦国的邀请，大不列颠及北爱尔兰联合王国副首相约翰·普雷斯科特将于7月1日至7日对中国进行正式访问。[SEP]
[CLS]7·英5国6····墨5西6哥6····华5新3华4社4北5京6·····外3交4部4···唐1国2强2·····国3务4院4···吴1邦2国2····大5不6列6颠6及6北6爱6尔6兰6联6合6王6国6···约1翰2·2普2雷2斯2科2特2·····中5国6·····[SEP]7
[CLS]7·英5国6····墨5西6哥6····华5新3华4社4北5京6·····外3交4部4···唐1国2强2·····国3务4院4···吴1邦2国2····大5不6列6颠6·北5爱6尔6兰6联4合6王6国6···约1翰2·2普2雷2斯2科2特2·····中5国6·····[SEP]7
==========
[CLS]本来会议并未邀请南平镇参加，但袁正军获悉后，火速派人赶往长沙联系。[SEP]

[CLS]7·····南5平6镇6····袁1正2军2·····长5沙6···[SEP]7
[CLS]7·····南5平6镇6····袁1正2军2·····长5沙6···[SEP]7
==========
[CLS]针对平安承保的沿江、沿湖、低洼及历史受淹标的情况，办事处抽调专人对这些标的进行清点和检查，并提出整改意见及汛期预防和标的转移的具体方案，先后对8家重点单位进行了检查，以此引起各单位领导的重视和支持。[SEP]
[CLS]7··平3安4·····办3事4处
4·····[SEP]7
[CLS]7·····处
4·····[SEP]7
==========
[CLS]我国大城市普遍采用复式计次制，一些边远地区采用包月制。[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]据有关专家分析，印度的股市和汇市纷纷下跌的主要原因是，除了美国和日本等国家因印度进行核试验而对其采取经济制裁措施外，世界银行和亚洲开发银行等决定停止向印度提供贷款。[SEP]
[CLS]7·····印5度6·····美5国6·日5本6····印5度6·····世3界4银4行4·亚3洲4开4发4银4行4·····印5度6·····[SEP]7
[CLS]7·····印5度6·····美5国6·日5本6····印5度6·····世3界4银4行4·亚3洲4开4发4银4行4·····印5度6·····[SEP]7
==========
[CLS]1996年初，石渠县发生特大雪灾，全县牲畜死亡率超过37%，而搞了[UNK]人草畜[UNK]配套建设的牧户牲畜死亡率平均不到10%。[SEP]
[CLS]7·····石5渠6县6·····[SEP]7
[CLS]7·····石5渠6县6·····[SEP]7
==========
[CLS]1984年他刚到海尔的前身[UNK][UNK][UNK]青岛电冰箱总厂时，面对的是一个发不出工资、濒临倒闭的烂摊子。[SEP]
[CLS]7·····海3尔4·····青3岛4电4冰4箱4总4厂4·····[SEP]7
[CLS]7·····海3尔4·····青3岛4电4冰4箱4总4厂4·····[SEP]7
==========
[CLS]本报北京6月17日讯新华社记者王黎、本报记者董洪亮报道：记者日前从教育部有关部门了解到，天津市在教育经费的投入上，多年来一直保持高于财政经常性收入增长的比例，有不少区县教育预算占地方财政一半以上；同时按政策积极组织社会力量筹措资金，在一定程度上弥补了教育经费的不足。[SEP]
[CLS]7··北5京6·····新3华4社4··王1黎2·····董1洪2亮2·····教3育4部4·····天5津6市6·····[SEP]7
[CLS]7··北5京6·····新3华4社4··王1黎2·····董1洪2亮2·····教3育4部4·····天5津6市6·····[SEP]7
==========
[CLS]吴健雄教授是当代第一流的实验原子核物理学家。[SEP]
[CLS]7吴1健2雄2·····[SEP]7
[CLS]7吴1健2雄2·····[SEP]7
==========
[CLS]1996年，为了改变海城的城市形象，市委、市政府决定对老城区进行重新规划和改造，为了解决改造过程中的一系列难题，我们一方面邀请著名的专家学者论证，一方面通过新闻媒体在全市开展了[UNK]市民建言[UNK]活动，结果收到市民的合理化建议60多条，为城市改造工程的顺利实施创造了先决条件。[SEP]
[CLS]7·····海5城6·····市3委
4·····[SEP]7
[CLS]7·····海5城6·····市3委
4·····[SEP]7

==========
[CLS]然而，前方吃紧，后方紧吃，吃在成都，乐在官场。[SEP]
[CLS]7··············成5都6·····[SEP]7
[CLS]7··············成5都6·····[SEP]7
==========
[CLS]1、每次飞行前，空中乘务员都要提前着装整齐在签到室里接受监督检查。[SEP]
[CLS]7·····························[SEP]7
[CLS]7·····························[SEP]7
==========
[CLS]成立大会上，经济日报社社长徐心华、光明日报总编辑王晨都表示，要以组建报业集团为契机，大胆探索，勇于创新，加快改革开放步伐，为推动有中国特色的社会主义现代化报业集团的健康发展作出贡献。
[SEP]
[CLS]7·····经3济4日4报4社4··徐1心2华2·光3明4日4报4···王1晨
2·····································中5国6·····[SEP]7
[CLS]7·····经3济4日4报4社4··徐1心2华2·光3明4日4报4···王1晨
2·····································中5国6·····[SEP]7
==========
[CLS]我们沿着崎岖的山路来到王永祥跟前时，他正伫立在丛林中间，凝视着充满生机的一片绿色。[SEP]
[CLS]7··········王1永2祥2·····[SEP]7
[CLS]7··········王1永2祥2·····[SEP]7
==========
[CLS]全国人大常委会法工委研究室吴高盛认为：目前我国已有了一套包括《反不正当竞争法》、《产品质量法》、《消费者权益保护法》等等在内的比较完备的市场竞争规则，但是还需要完善，还存在有法不依、执法不严的现象。[SEP]
[CLS]7全3国4人4大4常4委4会4法4工4委4研4究4室4吴1高2盛
2·····[SEP]7
[CLS]7全3国4人4大4常4委4会4法4工4委4研4究4室4吴1高2盛
2·····[SEP]7
==========
[CLS]但是，对于中国汽车产业来说，丰田是强有力的竞争对手之一。[SEP]
[CLS]7·····中5国6·····丰3田4·····[SEP]7
[CLS]7·····中5国6·····丰3田4·····[SEP]7
==========
[CLS]为了开辟再就业的途径，不讲清[UNK]转变观念天地宽[UNK]的道理怎么行呢？[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========
[CLS]为消除西方一些人在西藏问题上的偏见，卓科达先生在长期研究的基础上，撰写了有关中国宗教问题的专著。[SEP]
[CLS]7·····西5藏6·····卓1科2达2·····中5国6·····[SEP]7
[CLS]7·····西5藏6·····卓1科2达2·····中5国6·····[SEP]7
==========
[CLS]经过六届人大常委会第四次、第五次会议审议，决定提请第六届全国人大第二次会议审议，会上我作了关于《中华人民共和国民族区域自治法草案》的说明。[SEP]
[CLS]7··六3届4人4大4常4委4会4·····第3六4届4全4国4人4大4·····中5华6人6民6共6和6国6·····[SEP]7
[CLS]7··六3届4人4大4常4委4会4·····第3六4届4全4国4人4大4·····中5华6人6民6共6和6国6·····[SEP]7
==========
[CLS]不过那种乐于把诗歌的公共性乃至战斗性视为其美学要素的声音也没有停息。[SEP]
[CLS]7·····[SEP]7
[CLS]7·····[SEP]7
==========

```
[CLS]不久前的一天上午，刚毕业于曼谷万颂绿叻差博学院企业管理系的少女玛裕勒和诗丽婉到吞武里一带求职未果。[SEP]
[CLS]7·····曼3谷4万4颂4绿4叻4差4博4学4院4企4业4管4理4系4···玛1裕2勒2·诗1丽2婉2·吞5武6里6·····[SEP]7
[CLS]7·····曼5谷6万4颂4绿4叻4差4博4学4院4企4业4管4理4系4···玛1裕2勒2·诗1丽2婉2·吞5武6里6·····[SEP]7
============================
[CLS]上海工部局一两试铸银币是存世极为稀少的钱币品种。[SEP]
[CLS]7上3海4工4部4局4·····[SEP]7
[CLS]7上3海4工4部4局4·····[SEP]7
============================
[CLS]5月23日是墨西哥全国防疫接种卫生周第一天。[SEP]
[CLS]7·····墨5西6哥6·····[SEP]7
[CLS]7·····墨5西6哥6·····[SEP]7
============================
```

7.中文分类

```python
import torch
from datasets import load_dataset


#定义数据集
class Dataset(torch.utils.data.Dataset):
    def __init__(self, split):
        self.dataset = load_dataset(path='lansinuote/ChnSentiCorp', split=split)

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, i):
        text = self.dataset[i]['text']
        label = self.dataset[i]['label']

        return text, label


dataset = Dataset('train')

len(dataset), dataset[0]
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet (/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d6f236ec)
```

```
(9600,
 ('选择珠江花园的原因就是方便，有电动扶梯直接到达海边，周围餐馆、食廊、商场、超市、摊位一应俱全。酒店装修一般，但还算整洁。 泳池在大堂的屋顶，因此很小，不过女儿倒是喜欢。 包的早餐是西式的，还算丰富。 服务吗，一般',
  1))
```

```
from transformers import BertTokenizer

#加载字典和分词工具
token = BertTokenizer.from_pretrained('bert-base-chinese')

token
```

```
PreTrainedTokenizer(name_or_path='bert-base-chinese', vocab_size=21128,
model_max_len=512, is_fast=False, padding_side='right', truncation_side='right',
special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]',
'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```

```
def collate_fn(data):
    sents = [i[0] for i in data]
    labels = [i[1] for i in data]

    #编码
    data = token.batch_encode_plus(batch_text_or_text_pairs=sents,
                                   truncation=True,
                                   padding='max_length',
                                   max_length=500,
                                   return_tensors='pt',
                                   return_length=True)

    #input_ids:编码之后的数字
    #attention_mask:是补零的位置是0,其他位置是1
    input_ids = data['input_ids']
    attention_mask = data['attention_mask']
    token_type_ids = data['token_type_ids']
    labels = torch.LongTensor(labels)

    #print(data['length'], data['length'].max())

    return input_ids, attention_mask, token_type_ids, labels


#数据加载器
loader = torch.utils.data.DataLoader(dataset=dataset,
                                     batch_size=16,
                                     collate_fn=collate_fn,
                                     shuffle=True,
```

```
                                    drop_last=True)

for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    break

print(len(loader))
input_ids.shape, attention_mask.shape, token_type_ids.shape, labels
```

```
600
```

```
(torch.Size([16, 500]),
 torch.Size([16, 500]),
 torch.Size([16, 500]),
 tensor([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1]))
```

```
from transformers import BertModel

#加载预训练模型
pretrained = BertModel.from_pretrained('bert-base-chinese')

#不训练,不需要计算梯度
for param in pretrained.parameters():
    param.requires_grad_(False)

#模型试算
out = pretrained(input_ids=input_ids,
           attention_mask=attention_mask,
           token_type_ids=token_type_ids)

out.last_hidden_state.shape
```

```
Downloading:   0%|          | 0.00/393M [00:00<?, ?B/s]
```

```
Some weights of the model checkpoint at bert-base-chinese were not used when
initializing BertModel: ['cls.seq_relationship.bias',
'cls.predictions.decoder.weight', 'cls.predictions.bias',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.weight',
'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

```
torch.Size([16, 500, 768])
```

```
#定义下游任务模型
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = torch.nn.Linear(768, 2)

    def forward(self, input_ids, attention_mask, token_type_ids):
        with torch.no_grad():
            out = pretrained(input_ids=input_ids,
                             attention_mask=attention_mask,
                             token_type_ids=token_type_ids)

        out = self.fc(out.last_hidden_state[:, 0])

        out = out.softmax(dim=1)

        return out


model = Model()

model(input_ids=input_ids,
      attention_mask=attention_mask,
      token_type_ids=token_type_ids).shape
```

```
torch.Size([16, 2])
```

```
from transformers import AdamW

#训练
optimizer = AdamW(model.parameters(), lr=5e-4)
criterion = torch.nn.CrossEntropyLoss()

model.train()
for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    out = model(input_ids=input_ids,
                attention_mask=attention_mask,
                token_type_ids=token_type_ids)

    loss = criterion(out, labels)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    if i % 5 == 0:
        out = out.argmax(dim=1)
        accuracy = (out == labels).sum().item() / len(labels)

        print(i, loss.item(), accuracy)

    if i == 300:
        break
```

```
/root/anaconda3/envs/pt36/lib/python3.6/site-
packages/transformers/optimization.py:309: FutureWarning: This implementation of
AdamW is deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to
disable this warning
  FutureWarning,
```

```
0 0.7399377822875977 0.375
5 0.6872416138648987 0.5
10 0.6332055330276489 0.75
15 0.6033581495285034 0.875
20 0.5943211317062378 0.75
25 0.6094914674758911 0.6875
30 0.6384122371673584 0.6875
35 0.5677111744880676 0.8125
40 0.5644276142120361 0.875
45 0.5818642377853394 0.8125
50 0.5436866879463196 0.875
55 0.5654083490371704 0.75
60 0.6283971071243286 0.6875
65 0.4929084777832031 0.875
70 0.5428925156593323 0.8125
75 0.5615941882133484 0.75
80 0.49903589487075806 0.8125
85 0.5711023211479187 0.6875
90 0.5560241341590881 0.8125
95 0.5039068460464478 0.75
```

```
100 0.45021048188209534 0.9375
105 0.5826895833015442 0.6875
110 0.6238031983375549 0.625
115 0.4193401634693146 1.0
120 0.5034649968147278 0.875
125 0.4235936105251312 0.9375
130 0.42960819602012634 1.0
135 0.37958282232284546 1.0
140 0.5208486318588257 0.8125
145 0.5100721716880798 0.8125
150 0.43825772404670715 0.875
155 0.4317510724067688 0.9375
160 0.4287576675415039 0.9375
165 0.4266912639141083 0.9375
170 0.4405173361301422 0.875
175 0.41791820526123047 0.875
180 0.4508487582206726 0.8125
185 0.4445679485797882 0.8125
190 0.6184951066970825 0.625
195 0.5331169962882996 0.75
200 0.5569592714309692 0.75
205 0.3625647723674774 1.0
210 0.4005920886993408 0.9375
215 0.4386642575263977 0.9375
220 0.5098661184310913 0.8125
225 0.447701096534729 0.9375
230 0.3920193314552307 0.9375
235 0.4688347280025482 0.8125
240 0.46488767862319946 0.9375
245 0.4752849340438843 0.875
250 0.48376595973968506 0.8125
255 0.4025830626487732 0.9375
260 0.40194043517112273 0.9375
265 0.5092278122901917 0.8125
270 0.6099189519882202 0.625
275 0.42547380924224854 0.875
280 0.5527713894844055 0.75
285 0.5336724519729614 0.75
290 0.4221529960632324 0.8125
295 0.38887107372283936 1.0
300 0.4890342056751251 0.8125
```

```python
#测试
def test():
    model.eval()
    correct = 0
    total = 0

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('validation'),
                                              batch_size=32,
                                              collate_fn=collate_fn,
                                              shuffle=True,
                                              drop_last=True)
```

```
    for i, (input_ids, attention_mask, token_type_ids,
            labels) in enumerate(loader_test):

        if i == 5:
            break

        print(i)

        with torch.no_grad():
            out = model(input_ids=input_ids,
                        attention_mask=attention_mask,
                        token_type_ids=token_type_ids)

        out = out.argmax(dim=1)
        correct += (out == labels).sum().item()
        total += len(labels)

    print(correct / total)


test()
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
0
1
2
3
4
0.85
```

```
import torch
from datasets import load_dataset


#定义数据集
class Dataset(torch.utils.data.Dataset):
    def __init__(self, split):
        self.dataset = load_dataset(path='lansinuote/ChnSentiCorp', split=split)

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, i):
        text = self.dataset[i]['text']
        label = self.dataset[i]['label']
```

```
        return text, label


dataset = Dataset('train')

len(dataset), dataset[0]
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
(9600,
 ('选择珠江花园的原因就是方便，有电动扶梯直接到达海边，周围餐馆、食廊、商场、超市、摊位一应俱
全。酒店装修一般，但还算整洁。 泳池在大堂的屋顶，因此很小，不过女儿倒是喜欢。 包的早餐是西式的，
还算丰富。 服务吗，一般',
  1))
```

```
from transformers import BertTokenizer

#加载字典和分词工具
token = BertTokenizer.from_pretrained('bert-base-chinese')

token
```

```
PreTrainedTokenizer(name_or_path='bert-base-chinese', vocab_size=21128,
model_max_len=512, is_fast=False, padding_side='right', truncation_side='right',
special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]',
'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```

```
def collate_fn(data):
    sents = [i[0] for i in data]
    labels = [i[1] for i in data]

    #编码
    data = token.batch_encode_plus(batch_text_or_text_pairs=sents,
                                   truncation=True,
                                   padding='max_length',
                                   max_length=500,
                                   return_tensors='pt',
                                   return_length=True)
```

```
    #input_ids:编码之后的数字
    #attention_mask:是补零的位置是0,其他位置是1
    input_ids = data['input_ids']
    attention_mask = data['attention_mask']
    token_type_ids = data['token_type_ids']
    labels = torch.LongTensor(labels)

    #print(data['length'], data['length'].max())

    return input_ids, attention_mask, token_type_ids, labels


#数据加载器
loader = torch.utils.data.DataLoader(dataset=dataset,
                                     batch_size=16,
                                     collate_fn=collate_fn,
                                     shuffle=True,
                                     drop_last=True)

for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    break

print(len(loader))
input_ids.shape, attention_mask.shape, token_type_ids.shape, labels
```

```
600
```

```
(torch.Size([16, 500]),
 torch.Size([16, 500]),
 torch.Size([16, 500]),
 tensor([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1]))
```

```
from transformers import BertModel

#加载预训练模型
pretrained = BertModel.from_pretrained('bert-base-chinese')

#不训练,不需要计算梯度
for param in pretrained.parameters():
    param.requires_grad_(False)

#模型试算
out = pretrained(input_ids=input_ids,
           attention_mask=attention_mask,
           token_type_ids=token_type_ids)

out.last_hidden_state.shape
```

```
Downloading:   0%|          | 0.00/393M [00:00<?, ?B/s]
```

```
Some weights of the model checkpoint at bert-base-chinese were not used when
initializing BertModel: ['cls.seq_relationship.bias',
'cls.predictions.decoder.weight', 'cls.predictions.bias',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.weight',
'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

```
torch.Size([16, 500, 768])
```

```
#定义下游任务模型
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = torch.nn.Linear(768, 2)

    def forward(self, input_ids, attention_mask, token_type_ids):
        with torch.no_grad():
            out = pretrained(input_ids=input_ids,
                             attention_mask=attention_mask,
                             token_type_ids=token_type_ids)

        out = self.fc(out.last_hidden_state[:, 0])

        out = out.softmax(dim=1)

        return out


model = Model()

model(input_ids=input_ids,
      attention_mask=attention_mask,
      token_type_ids=token_type_ids).shape
```

```
torch.Size([16, 2])
```

```
from transformers import AdamW

#训练
optimizer = AdamW(model.parameters(), lr=5e-4)
criterion = torch.nn.CrossEntropyLoss()

model.train()
for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    out = model(input_ids=input_ids,
                attention_mask=attention_mask,
                token_type_ids=token_type_ids)

    loss = criterion(out, labels)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    if i % 5 == 0:
        out = out.argmax(dim=1)
        accuracy = (out == labels).sum().item() / len(labels)

        print(i, loss.item(), accuracy)

    if i == 300:
        break
```

```
/root/anaconda3/envs/pt36/lib/python3.6/site-
packages/transformers/optimization.py:309: FutureWarning: This implementation of
AdamW is deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to
disable this warning
  FutureWarning,
```

```
0 0.7399377822875977 0.375
5 0.6872416138648987 0.5
10 0.6332055330276489 0.75
15 0.6033581495285034 0.875
20 0.5943211317062378 0.75
25 0.6094914674758911 0.6875
30 0.6384122371673584 0.6875
35 0.5677111744880676 0.8125
40 0.5644276142120361 0.875
45 0.5818642377853394 0.8125
50 0.5436866879463196 0.875
55 0.5654083490371704 0.75
60 0.6283971071243286 0.6875
65 0.4929084777832031 0.875
70 0.5428925156593323 0.8125
75 0.5615941882133484 0.75
80 0.49903589487075806 0.8125
85 0.5711023211479187 0.6875
```

```
90 0.5560241341590881 0.8125
95 0.5039068460464478 0.75
100 0.45021048188209534 0.9375
105 0.5826895833015442 0.6875
110 0.6238031983375549 0.625
115 0.4193401634693146 1.0
120 0.5034649968147278 0.875
125 0.4235936105251312 0.9375
130 0.42960819602012634 1.0
135 0.37958282232284546 1.0
140 0.5208486318588257 0.8125
145 0.5100721716880798 0.8125
150 0.43825772404670715 0.875
155 0.4317510724067688 0.9375
160 0.4287576675415039 0.9375
165 0.4266912639141083 0.9375
170 0.4405173361301422 0.875
175 0.41791820526123047 0.875
180 0.4508487582206726 0.8125
185 0.4445679485797882 0.8125
190 0.6184951066970825 0.625
195 0.5331169962882996 0.75
200 0.5569592714309692 0.75
205 0.3625647723674774 1.0
210 0.4005920886993408 0.9375
215 0.4386642575263977 0.9375
220 0.5098661184310913 0.8125
225 0.447701096534729 0.9375
230 0.3920193314552307 0.9375
235 0.4688347280025482 0.8125
240 0.46488767862319946 0.9375
245 0.4752849340438843 0.875
250 0.48376595973968506 0.8125
255 0.4025830626487732 0.9375
260 0.4019404351711273 0.9375
265 0.5092278122901917 0.8125
270 0.6099189519882202 0.625
275 0.42547380924224854 0.875
280 0.5527713894844055 0.75
285 0.5336724519729614 0.75
290 0.4221529960632324 0.8125
295 0.38887107372283936 1.0
300 0.4890342056751251 0.8125
```

```python
#测试
def test():
    model.eval()
    correct = 0
    total = 0

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('validation'),
                                              batch_size=32,
                                              collate_fn=collate_fn,
```

```
                                           shuffle=True,
                                           drop_last=True)

    for i, (input_ids, attention_mask, token_type_ids,
            labels) in enumerate(loader_test):

        if i == 5:
            break

        print(i)

        with torch.no_grad():
            out = model(input_ids=input_ids,
                        attention_mask=attention_mask,
                        token_type_ids=token_type_ids)

        out = out.argmax(dim=1)
        correct += (out == labels).sum().item()
        total += len(labels)

    print(correct / total)


test()
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
0
1
2
3
4
0.85
```

8.中文填空

```
import torch
from datasets import load_dataset


#定义数据集
class Dataset(torch.utils.data.Dataset):
    def __init__(self, split):
        dataset = load_dataset(path='lansinuote/ChnSentiCorp', split=split)

        def f(data):
            return len(data['text']) > 30
```

```
        self.dataset = dataset.filter(f)

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, i):
        text = self.dataset[i]['text']

        return text


dataset = Dataset('train')

len(dataset), dataset[0]
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
  0%|          | 0/10 [00:00<?, ?ba/s]
```

```
(9192,
 '选择珠江花园的原因就是方便，有电动扶梯直接到达海边，周围餐馆、食廊、商场、超市、摊位一应俱全。 酒店装修一般，但还算整洁。 泳池在大堂的屋顶，因此很小，不过女儿倒是喜欢。 包的早餐是西式的，还算丰富。 服务吗，一般')
```

```
from transformers import BertTokenizer

#加载字典和分词工具
token = BertTokenizer.from_pretrained('bert-base-chinese')

token
```

```
PreTrainedTokenizer(name_or_path='bert-base-chinese', vocab_size=21128,
model_max_len=512, is_fast=False, padding_side='right', truncation_side='right',
special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]',
'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```

```
def collate_fn(data):
    #编码
    data = token.batch_encode_plus(batch_text_or_text_pairs=data,
                                   truncation=True,
                                   padding='max_length',
                                   max_length=30,
                                   return_tensors='pt',
                                   return_length=True)

    #input_ids:编码之后的数字
    #attention_mask:是补零的位置是0,其他位置是1
    input_ids = data['input_ids']
    attention_mask = data['attention_mask']
    token_type_ids = data['token_type_ids']

    #把第15个词固定替换为mask
    labels = input_ids[:, 15].reshape(-1).clone()
    input_ids[:, 15] = token.get_vocab()[token.mask_token]

    #print(data['length'], data['length'].max())

    return input_ids, attention_mask, token_type_ids, labels


#数据加载器
loader = torch.utils.data.DataLoader(dataset=dataset,
                                     batch_size=16,
                                     collate_fn=collate_fn,
                                     shuffle=True,
                                     drop_last=True)

for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    break

print(len(loader))
print(token.decode(input_ids[0]))
print(token.decode(labels[0]))
input_ids.shape, attention_mask.shape, token_type_ids.shape, labels.shape
```

```
574
[CLS] 屏 幕 的 长 宽 比 例 太 奇 怪 ， 看 了 很 [MASK] 爽 ！ 镜 面 感 太 强 ， 还 是 喜 欢 亚 [SEP]
不
```

```
(torch.Size([16, 30]),
 torch.Size([16, 30]),
 torch.Size([16, 30]),
 torch.Size([16]))
```

```
from transformers import BertModel

#加载预训练模型
pretrained = BertModel.from_pretrained('bert-base-chinese')

#不训练,不需要计算梯度
for param in pretrained.parameters():
    param.requires_grad_(False)

#模型试算
out = pretrained(input_ids=input_ids,
           attention_mask=attention_mask,
           token_type_ids=token_type_ids)

out.last_hidden_state.shape
```

```
Some weights of the model checkpoint at bert-base-chinese were not used when
initializing BertModel: ['cls.seq_relationship.weight',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.decoder.weight',
'cls.predictions.transform.dense.weight', 'cls.seq_relationship.bias',
'cls.predictions.bias', 'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

```
torch.Size([16, 30, 768])
```

```
#定义下游任务模型
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.decoder = torch.nn.Linear(768, token.vocab_size, bias=False)
        self.bias = torch.nn.Parameter(torch.zeros(token.vocab_size))
        self.decoder.bias = self.bias

    def forward(self, input_ids, attention_mask, token_type_ids):
        with torch.no_grad():
            out = pretrained(input_ids=input_ids,
                             attention_mask=attention_mask,
                             token_type_ids=token_type_ids)

        out = self.decoder(out.last_hidden_state[:, 15])
```

```
        return out


model = Model()

model(input_ids=input_ids,
      attention_mask=attention_mask,
      token_type_ids=token_type_ids).shape
```

```
torch.Size([16, 21128])
```

```
from transformers import AdamW

#训练
optimizer = AdamW(model.parameters(), lr=5e-4)
criterion = torch.nn.CrossEntropyLoss()

model.train()
for epoch in range(5):
    for i, (input_ids, attention_mask, token_type_ids,
            labels) in enumerate(loader):
        out = model(input_ids=input_ids,
                    attention_mask=attention_mask,
                    token_type_ids=token_type_ids)

        loss = criterion(out, labels)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if i % 50 == 0:
            out = out.argmax(dim=1)
            accuracy = (out == labels).sum().item() / len(labels)

            print(epoch, i, loss.item(), accuracy)
```

```
/root/anaconda3/envs/pt36/lib/python3.6/site-
packages/transformers/optimization.py:309: FutureWarning: This implementation of
AdamW is deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to
disable this warning
  FutureWarning,
```

```
0 0 9.913239479064941 0.0
0 50 7.882239818572998 0.1875
0 100 6.802951812744141 0.125
0 150 4.709199905395508 0.25
0 200 5.053264141082764 0.375
```

```
0 250 5.4513750076293945 0.375
0 300 4.834373474121094 0.4375
0 350 2.740715503692627 0.6875
0 400 4.249405384063721 0.375
0 450 2.4256627559661865 0.6875
0 500 3.162682294845581 0.375
0 550 2.479234457015991 0.5625
1 0 3.3130388259887695 0.4375
1 50 2.428676128387451 0.625
1 100 1.8036706447601318 0.8125
1 150 2.2352967262268066 0.625
1 200 1.6442852020263672 0.8125
1 250 1.8351222276687622 0.75
1 300 1.6686547994613647 0.8125
1 350 2.3184170722961426 0.5625
1 400 2.627448797225952 0.5625
1 450 2.4967753887176514 0.5625
1 500 2.7677855491638184 0.625
1 550 2.1934893131256104 0.6875
2 0 1.5646746158599854 0.75
2 50 1.2516077756881714 0.75
2 100 0.6442171931266785 0.9375
2 150 0.8584479093551636 0.875
2 200 2.042940855026245 0.75
2 250 0.7246753573417664 0.8125
2 300 1.1209107637405396 0.75
2 350 1.4746692180633545 0.75
2 400 0.8257594108581543 0.875
2 450 1.384639859199524 0.6875
2 500 1.2145320177078247 0.8125
2 550 1.2005428075790405 0.75
3 0 0.8594522476196289 0.75
3 50 0.474687784910202 0.9375
3 100 0.5052637457847595 0.875
3 150 0.9241865277290344 0.75
3 200 0.6268807649612427 0.9375
3 250 0.5165101885795593 0.875
3 300 0.5541099905967712 0.9375
3 350 0.6904842853546143 0.9375
3 400 0.2985838055610657 0.9375
3 450 1.1752042770385742 0.75
3 500 0.7742744088172913 0.9375
3 550 0.7787161469459534 0.75
4 0 0.23652660846710205 1.0
4 50 0.28316396474838257 1.0
4 100 0.5179316997528076 0.875
4 150 0.3183712959289551 1.0
4 200 0.5086650252342224 0.875
4 250 0.28567108511924744 0.9375
4 300 0.4397639334201813 0.875
4 350 0.23462174832820892 0.9375
4 400 0.5601446628570557 0.875
4 450 0.1961936503648758 1.0
4 500 0.32445845007896423 1.0
4 550 0.2166948765516281 0.9375
```

```python
#测试
def test():
    model.eval()
    correct = 0
    total = 0

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('test'),
                                              batch_size=32,
                                              collate_fn=collate_fn,
                                              shuffle=True,
                                              drop_last=True)

    for i, (input_ids, attention_mask, token_type_ids,
            labels) in enumerate(loader_test):

        if i == 15:
            break

        print(i)

        with torch.no_grad():
            out = model(input_ids=input_ids,
                        attention_mask=attention_mask,
                        token_type_ids=token_type_ids)

        out = out.argmax(dim=1)
        correct += (out == labels).sum().item()
        total += len(labels)

        print(token.decode(input_ids[0]))
        print(token.decode(labels[0]), token.decode(labels[0]))

    print(correct / total)


test()
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
  0%|          | 0/2 [00:00<?, ?ba/s]
```

```
0
[CLS] 床 发 出 吱 嘎 吱 嘎 的 声 音 ， 房 间 隔 [MASK] 太 差 ， 赠 送 的 早 餐 非 常 好
吃 。 [SEP]
音 音
```

1
[CLS] 非 常 不 好 ， 我 们 渡 过 了 一 个 让 人 [MASK] 以 忍 受 的 纪 念 日. com / thread - 136 [SEP]
难 难
2
[CLS] 定 的 商 务 大 床 房 ， 房 间 偏 小 了 ， [MASK] 过 经 济 性 酒 店 也 就 这 样 ； 环 境 [SEP]
不 不
3
[CLS] 确 实 是 山 上 最 好 的 酒 店 ， 环 境 和 [MASK] 施 都 很 不 错 。 我 们 这 次 住 的 是 [SEP]
设 设
4
[CLS] 这 本 书 紧 接 《 春 秋 大 义 》 ， 作 者 [MASK] 以 贯 之 地 以 浅 显 的 语 言 ， 告 诉 [SEP]
一 一
5
[CLS] 非 常 不 满 这 酒 店 ， 配 不 上 5 星 。 [MASK] 一 , 客 房 服 务 员 没 有 水 平 , 房 [SEP]
第 第
6
[CLS] 合 庆 的 商 务 单 间 可 以 堪 称 豪 华 , [MASK] 施 特 别 先 进 ， 特 别 是 少 有 的 先 [SEP]
设 设
7
[CLS] 这 是 我 住 过 的 最 差 的 酒 店 ， 房 间 [MASK] 味 难 闻 ， 刚 打 了 灭 蚊 药 水 ， 换 [SEP]
气 气
8
[CLS] 总 体 很 满 意 ， 但 有 一 点 需 改 进 ， [MASK] 在 9 楼 入 住 ， 走 时 到 1 楼 前 台 [SEP]
我 我
9
[CLS] 这 本 书 有 别 于 以 往 看 过 的 早 教 书 [MASK] ， 结 合 了 说 明 文 的 写 实 ， 散 文 [SEP]
籍 籍
10
[CLS] [UNK] 用 起 来 不 习 惯 ， 速 度 慢 ， 分 区 [MASK] 烦 ， 带 了 很 多 垃 圾 软 件 ， 卸 载 [SEP]
麻 麻
11
[CLS] 这 一 套 书 我 基 本 买 齐 了 ， 也 看 了 [MASK] 多 本 了 。 是 利 用 闲 暇 时 间 巩 固 [SEP]
好 好
12
[CLS] 渡 假 村 周 围 景 色 不 错 , 但 较 落 乡 [MASK] 硬 件 太 差 , 服 务 水 准 有 待 提 高 [SEP]
. .
13
[CLS] 相 对 来 说 ， 比 起 东 莞 ， 深 圳 的 酒 [MASK] ， 该 酒 店 服 务 质 量 还 是 略 有 小 [SEP]
店 店
14
[CLS] 目 前 是 新 昌 最 好 的 酒 店 ， 环 境 和 [MASK] 务 质 量 都 非 常 好 ， 国 内 长 度 免 [SEP]

```
服 服
0.6875
```

9.中文句子关系推断

```
import torch
from datasets import load_dataset
import random


#定义数据集
class Dataset(torch.utils.data.Dataset):
    def __init__(self, split):
        dataset = load_dataset(path='lansinuote/ChnSentiCorp', split=split)

        def f(data):
            return len(data['text']) > 40

        self.dataset = dataset.filter(f)

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, i):
        text = self.dataset[i]['text']

        #切分一句话为前半句和后半句
        sentence1 = text[:20]
        sentence2 = text[20:40]
        label = 0

        #有一半的概率把后半句替换为一句无关的话
        if random.randint(0, 1) == 0:
            j = random.randint(0, len(self.dataset) - 1)
            sentence2 = self.dataset[j]['text'][20:40]
            label = 1

        return sentence1, sentence2, label


dataset = Dataset('train')

sentence1, sentence2, label = dataset[0]

len(dataset), sentence1, sentence2, label
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
  0%|          | 0/10 [00:00<?, ?ba/s]
```

```
(8001, '选择珠江花园的原因就是方便，有电动扶梯直', '这本书也就靠色戒作卖点,别的小说和文章都', 1)
```

```
from transformers import BertTokenizer

#加载字典和分词工具
token = BertTokenizer.from_pretrained('bert-base-chinese')

token
```

```
PreTrainedTokenizer(name_or_path='bert-base-chinese', vocab_size=21128, model_max_len=512, is_fast=False, padding_side='right', truncation_side='right', special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```

```
def collate_fn(data):
    sents = [i[:2] for i in data]
    labels = [i[2] for i in data]

    #编码
    data = token.batch_encode_plus(batch_text_or_text_pairs=sents,
                                   truncation=True,
                                   padding='max_length',
                                   max_length=45,
                                   return_tensors='pt',
                                   return_length=True,
                                   add_special_tokens=True)

    #input_ids:编码之后的数字
    #attention_mask:是补零的位置是0,其他位置是1
    #token_type_ids:第一个句子和特殊符号的位置是0,第二个句子的位置是1
    input_ids = data['input_ids']
    attention_mask = data['attention_mask']
    token_type_ids = data['token_type_ids']
    labels = torch.LongTensor(labels)

    #print(data['length'], data['length'].max())

    return input_ids, attention_mask, token_type_ids, labels
```

```
#数据加载器
loader = torch.utils.data.DataLoader(dataset=dataset,
                                     batch_size=8,
                                     collate_fn=collate_fn,
                                     shuffle=True,
                                     drop_last=True)

for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    break

print(len(loader))
print(token.decode(input_ids[0]))
input_ids.shape, attention_mask.shape, token_type_ids.shape, labels
```

```
1000
[CLS] 这 部 书 写 得 很 好 。 它 抛 开 了 那 些 众 所 周 知 的 东 [SEP] 错 ， 去 那 里 都 很 方 便 。 3 服 务 态 度 好 。 [SEP] [PAD] [PAD] [PAD] [PAD] [PAD]
```

```
(torch.Size([8, 45]),
 torch.Size([8, 45]),
 torch.Size([8, 45]),
 tensor([1, 0, 1, 1, 1, 1, 0, 1]))
```

```
from transformers import BertModel

#加载预训练模型
pretrained = BertModel.from_pretrained('bert-base-chinese')

#不训练,不需要计算梯度
for param in pretrained.parameters():
    param.requires_grad_(False)

#模型试算
out = pretrained(input_ids=input_ids,
           attention_mask=attention_mask,
           token_type_ids=token_type_ids)

out.last_hidden_state.shape
```

```
Some weights of the model checkpoint at bert-base-chinese were not used when initializing BertModel: ['cls.predictions.decoder.weight', 'cls.predictions.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.seq_relationship.weight', 'cls.predictions.transform.dense.bias', 'cls.seq_relationship.bias', 'cls.predictions.transform.LayerNorm.weight']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```

```
torch.Size([8, 45, 768])
```

```
#定义下游任务模型
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = torch.nn.Linear(768, 2)

    def forward(self, input_ids, attention_mask, token_type_ids):
        with torch.no_grad():
            out = pretrained(input_ids=input_ids,
                             attention_mask=attention_mask,
                             token_type_ids=token_type_ids)

        out = self.fc(out.last_hidden_state[:, 0])

        out = out.softmax(dim=1)

        return out


model = Model()

model(input_ids=input_ids,
      attention_mask=attention_mask,
      token_type_ids=token_type_ids).shape
```

```
torch.Size([8, 2])
```

```
from transformers import AdamW
```

```
#训练
optimizer = AdamW(model.parameters(), lr=5e-4)
criterion = torch.nn.CrossEntropyLoss()

model.train()
for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    out = model(input_ids=input_ids,
                attention_mask=attention_mask,
                token_type_ids=token_type_ids)

    loss = criterion(out, labels)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    if i % 5 == 0:
        out = out.argmax(dim=1)
        accuracy = (out == labels).sum().item() / len(labels)
        print(i, loss.item(), accuracy)

    if i == 300:
        break
```

```
/root/anaconda3/envs/pt36/lib/python3.6/site-
packages/transformers/optimization.py:309: FutureWarning: This implementation of
AdamW is deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to
disable this warning
  FutureWarning,
```

```
0 0.6634587645530701 0.5
5 0.6098350286483765 0.875
10 0.5052497386932373 0.875
15 0.5447399020195007 0.75
20 0.4267336130142212 0.875
25 0.5306932330131531 0.75
30 0.36168521642684937 1.0
35 0.3812262713909149 1.0
40 0.35402268171310425 1.0
45 0.37419548630714417 1.0
50 0.43632933497428894 0.875
55 0.5140734910964966 0.75
60 0.41520506143569946 0.875
65 0.6602439880371094 0.625
70 0.4336013197898865 1.0
75 0.3786592185497284 1.0
80 0.3748184144496918 1.0
85 0.3181508779525757 1.0
90 0.449066698551178 0.875
95 0.41723987460136414 0.875
100 0.3945035934448242 1.0
105 0.3723413944244385 1.0
```

```
110 0.6518347263336182 0.625
115 0.6475785970687866 0.625
120 0.3987446427345276 0.875
125 0.5805999040603638 0.625
130 0.41772183775901794 0.875
135 0.5152970552444458 0.875
140 0.4091838002204895 0.875
145 0.45193392038345337 0.875
150 0.45574262738227844 0.875
155 0.40596330165863037 0.875
160 0.4846689701080322 0.75
165 0.3870573937892914 1.0
170 0.6176437139511108 0.625
175 0.5534878373146057 0.75
180 0.5309603214263916 0.625
185 0.3175361454486847 1.0
190 0.46388697624206543 0.75
195 0.5528424978256226 0.75
200 0.46861058473587036 0.875
205 0.45440012216567993 0.875
210 0.3227441608905792 1.0
215 0.5495651960372925 0.75
220 0.4661363959312439 0.875
225 0.4828658699989319 0.875
230 0.5163189768791199 0.75
235 0.42287150025367737 0.875
240 0.33448562026023865 1.0
245 0.507199227809906 0.75
250 0.48013466596603394 0.875
255 0.47705042362213135 0.75
260 0.4274448752403259 0.875
265 0.39601731300354004 1.0
270 0.4313715696334839 0.875
275 0.41638004779815674 0.875
280 0.4004001319408417 0.875
285 0.493851900100708 0.75
290 0.43360376358032227 0.875
295 0.4535151720046997 0.875
300 0.6158324480056763 0.625
```

```python
#测试
def test():
    model.eval()
    correct = 0
    total = 0

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('test'),
                                              batch_size=32,
                                              collate_fn=collate_fn,
                                              shuffle=True,
                                              drop_last=True)

    for i, (input_ids, attention_mask, token_type_ids,
```

```
            labels) in enumerate(loader_test):

        if i == 5:
            break

        print(i)

        with torch.no_grad():
            out = model(input_ids=input_ids,
                        attention_mask=attention_mask,
                        token_type_ids=token_type_ids)

        pred = out.argmax(dim=1)

        correct += (pred == labels).sum().item()
        total += len(labels)

    print(correct / total)


test()
```

```
Using custom data configuration lansinuote--ChnSentiCorp-4d058ef86e3db8d5
Reusing dataset parquet
(/root/.cache/huggingface/datasets/lansinuote___parquet/lansinuote--ChnSentiCorp-
4d058ef86e3db8d5/0.0.0/2a3b91fbd88a2c90d1dbbb32b460cf621d31bd5b05b934492fdef7d8d
6f236ec)
```

```
  0%|          | 0/2 [00:00<?, ?ba/s]
```

```
0
1
2
3
4
0.8625
```

10.trainer

```
from transformers import AutoTokenizer

#加载分词工具
tokenizer = AutoTokenizer.from_pretrained('bert-base-cased')

tokenizer
```

```
PreTrainedTokenizerFast(name_or_path='bert-base-cased', vocab_size=28996,
model_max_len=512, is_fast=True, padding_side='right', truncation_side='right',
special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]',
'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```

```
from datasets import load_dataset
from datasets import load_from_disk

#加载数据集
#从网络加载
#datasets = load_dataset(path='glue', name='sst2')

#从本地磁盘加载数据
datasets = load_from_disk('./data/glue_sst2')


#分词
def f(data):
    return tokenizer(
        data['sentence'],
        padding='max_length',
        truncation=True,
        max_length=30,
    )


datasets = datasets.map(f, batched=True, batch_size=1000, num_proc=4)

#取数据子集，否则数据太多跑不动
dataset_train = datasets['train'].shuffle().select(range(1000))
dataset_test = datasets['validation'].shuffle().select(range(200))

del datasets

dataset_train
```

```
Loading cached processed dataset at data/glue_sst2/train/cache-
8cfcfd522c905812.arrow
```

```
Loading cached processed dataset at data/glue_sst2/train/cache-
b3e205139be224a8.arrow
```

```
Loading cached processed dataset at data/glue_sst2/train/cache-
890424ad8ed86a19.arrow
```

```
Loading cached processed dataset at data/glue_sst2/train/cache-
de855cf12202060d.arrow
```

```
Loading cached processed dataset at data/glue_sst2/validation/cache-
1de25e78c7da64ff.arrow
```

```
Loading cached processed dataset at data/glue_sst2/validation/cache-
620f9bf015596baf.arrow
```

```
Loading cached processed dataset at data/glue_sst2/validation/cache-
0a0b01f98a68ed20.arrow
```

```
Loading cached processed dataset at data/glue_sst2/validation/cache-
f7f25a94b8aafa52.arrow
```

```
Loading cached processed dataset at data/glue_sst2/test/cache-
524ecec34978f89b.arrow
```

```
Loading cached processed dataset at data/glue_sst2/test/cache-
86844412c2345cce.arrow
```

```
Loading cached processed dataset at data/glue_sst2/test/cache-
632228277b828dc1.arrow
```

```
Loading cached processed dataset at data/glue_sst2/test/cache-
fd22071d68ee727d.arrow
```

```
Dataset({
    features: ['sentence', 'label', 'idx', 'input_ids', 'token_type_ids',
'attention_mask'],
    num_rows: 1000
})
```

```
from transformers import AutoModelForSequenceClassification

#加载模型
model = AutoModelForSequenceClassification.from_pretrained('bert-base-cased',
                                                           num_labels=2)

print(sum([i.nelement() for i in model.parameters()]) / 10000)
```

```
Some weights of the model checkpoint at bert-base-cased were not used when
initializing BertForSequenceClassification:
['cls.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.bias',
'cls.predictions.transform.dense.weight', 'cls.seq_relationship.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertForSequenceClassification from the
checkpoint of a model trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForSequenceClassification from
the checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model
checkpoint at bert-base-cased and are newly initialized: ['classifier.bias',
'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
10831.181
```

```
import numpy as np
from datasets import load_metric
from transformers.trainer_utils import EvalPrediction

#加载评价函数
#有时会因为网络问题卡主,反复尝试会成功的
metric = load_metric('accuracy')


#定义评价函数
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    logits = logits.argmax(axis=1)
    return metric.compute(predictions=logits, references=labels)
```

```
#模拟测试输出
eval_pred = EvalPrediction(
    predictions=np.array([[0, 1], [2, 3], [4, 5], [6, 7]]),
    label_ids=np.array([1, 1, 1, 1]),
)

compute_metrics(eval_pred)
```

```
{'accuracy': 1.0}
```

```
from transformers import TrainingArguments, Trainer

#初始化训练参数
args = TrainingArguments(output_dir='./output_dir',
                         evaluation_strategy='epoch',
                         no_cuda=True)
args.num_train_epochs = 1
args.learning_rate = 1e-4
args.weight_decay = 1e-2
args.per_device_eval_batch_size = 32
args.per_device_train_batch_size = 16

#初始化训练器
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=dataset_train,
    eval_dataset=dataset_test,
    compute_metrics=compute_metrics,
)

#评价模型
trainer.evaluate()
```

```
The following columns in the evaluation set  don't have a corresponding argument
in `BertForSequenceClassification.forward` and have been ignored: sentence, idx.
If sentence, idx are not expected by `BertForSequenceClassification.forward`,
you can safely ignore this message.
***** Running Evaluation *****
  Num examples = 200
  Batch size = 32
```

```
<div>

  <progress value='14' max='7' style='width:300px; height:20px; vertical-align:
middle;'></progress>
  [7/7 01:35]
</div>
```

```
{'eval_loss': 0.7613461017608643,
 'eval_accuracy': 0.505,
 'eval_runtime': 4.5246,
 'eval_samples_per_second': 44.202,
 'eval_steps_per_second': 1.547}
```

```
#训练
trainer.train()
```

```
The following columns in the training set  don't have a corresponding argument in
`BertForSequenceClassification.forward` and have been ignored: sentence, idx. If
sentence, idx are not expected by `BertForSequenceClassification.forward`,  you
can safely ignore this message.
/root/anaconda3/envs/pt36/lib/python3.6/site-
packages/transformers/optimization.py:309: FutureWarning: This implementation of
AdamW is deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to
disable this warning
  FutureWarning,
***** Running training *****
  Num examples = 1000
  Num Epochs = 1
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 63
```

```
<div>

  <progress value='63' max='63' style='width:300px; height:20px; vertical-align:
middle;'></progress>
  [63/63 01:30, Epoch 1/1]
</div>
<table border="1" class="dataframe">
```

Epoch
Training Loss
Validation Loss
Accuracy

1
No log
0.462459
0.790000

```
The following columns in the evaluation set  don't have a corresponding argument
in `BertForSequenceClassification.forward` and have been ignored: sentence, idx.
If sentence, idx are not expected by `BertForSequenceClassification.forward`,
you can safely ignore this message.
***** Running Evaluation *****
  Num examples = 200
  Batch size = 32
```

```
Training completed. Do not forget to share your model on huggingface.co/models
=)
```

```
TrainOutput(global_step=63, training_loss=0.5899859837123326, metrics=
{'train_runtime': 92.1116, 'train_samples_per_second': 10.856,
'train_steps_per_second': 0.684, 'total_flos': 15416663400000.0, 'train_loss':
0.5899859837123326, 'epoch': 1.0})
```

```
#评价模型
trainer.evaluate()
```

```
The following columns in the evaluation set  don't have a corresponding argument in `BertForSequenceClassification.forward` and have been ignored: sentence, idx. If sentence, idx are not expected by `BertForSequenceClassification.forward`, you can safely ignore this message.
***** Running Evaluation *****
  Num examples = 200
  Batch size = 32
```

```
<div>

  <progress value='7' max='7' style='width:300px; height:20px; vertical-align: middle;'></progress>
  [7/7 00:03]
</div>
```

```
{'eval_loss': 0.46245864033699036,
 'eval_accuracy': 0.79,
 'eval_runtime': 4.2879,
 'eval_samples_per_second': 46.643,
 'eval_steps_per_second': 1.633,
 'epoch': 1.0}
```

```
#保存模型
trainer.save_model(output_dir='./output_dir')
```

```
Saving model checkpoint to ./output_dir
Configuration saved in ./output_dir/config.json
Model weights saved in ./output_dir/pytorch_model.bin
```

```
import torch


def collate_fn(data):
    label = [i['label'] for i in data]
    input_ids = [i['input_ids'] for i in data]
    token_type_ids = [i['token_type_ids'] for i in data]
    attention_mask = [i['attention_mask'] for i in data]

    label = torch.LongTensor(label)
    input_ids = torch.LongTensor(input_ids)
    token_type_ids = torch.LongTensor(token_type_ids)
    attention_mask = torch.LongTensor(attention_mask)
```

```
    return label, input_ids, token_type_ids, attention_mask


#数据加载器
loader_test = torch.utils.data.DataLoader(dataset=dataset_test,
                                          batch_size=4,
                                          collate_fn=collate_fn,
                                          shuffle=True,
                                          drop_last=True)

for i, (label, input_ids, token_type_ids,
        attention_mask) in enumerate(loader_test):
    break

label, input_ids, token_type_ids, attention_mask
```

```
(tensor([1, 0, 1, 1]),
 tensor([[  101,  1177,  1277,   175,  7409,  4759,  5531,   117,  1216, 10509,
           4133,   117,  1177,  1376,  2523,   119,   102,     0,     0,     0,
              0,     0,     0,     0,     0,     0,     0,     0,     0,     0],
         [  101,  1175,  1132,  4928,  7996,  1992,  1536,  1111,   188,  2522,
           1358,  1103,  7010, 17757,  1106, 11231,  1194,   119,   102,     0,
              0,     0,     0,     0,     0,     0,     0,     0,     0,     0],
         [  101,  1209,  1821,  5613,  1105,  5250, 14638, 16889, 25576,  6323,
           1107, 13858,  9165,   119,   102,     0,     0,     0,     0,     0,
              0,     0,     0,     0,     0,     0,     0,     0,     0,     0],
         [  101,  1141,  1104,  1103,  1167,  9998,  1482,   112,   188,  5558,
           1106,  1855, 13090,  1142,  1214,   119,   102,     0,     0,     0,
              0,     0,     0,     0,     0,     0,     0,     0,     0,
0]]),
 tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0]]),
 tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0],
         [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0],
         [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0],
```

```
         [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
0,
          0, 0, 0, 0, 0, 0]]))
```

```
import torch


#测试
def test():
    #加载参数
    model.load_state_dict(torch.load('./output_dir/pytorch_model.bin'))

    model.eval()

    #运算
    out = model(input_ids=input_ids,
                token_type_ids=token_type_ids,
                attention_mask=attention_mask)

    #[4, 2] -> [4]
    out = out['logits'].argmax(dim=1)

    correct = (out == label).sum().item()

    return correct / len(label)


test()
```

```
0.75
```

```
from transformers import AutoTokenizer

#加载分词工具
tokenizer = AutoTokenizer.from_pretrained('bert-base-cased')

tokenizer
```

```
PreTrainedTokenizerFast(name_or_path='bert-base-cased', vocab_size=28996,
model_max_len=512, is_fast=True, padding_side='right', truncation_side='right',
special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]',
'cls_token': '[CLS]', 'mask_token': '[MASK]'})
```

```
from datasets import load_dataset
from datasets import load_from_disk

#加载数据集
#从网络加载
#datasets = load_dataset(path='glue', name='sst2')

#从本地磁盘加载数据
datasets = load_from_disk('./data/glue_sst2')


#分词
def f(data):
    return tokenizer(
        data['sentence'],
        padding='max_length',
        truncation=True,
        max_length=30,
    )


datasets = datasets.map(f, batched=True, batch_size=1000, num_proc=4)

#取数据子集，否则数据太多跑不动
dataset_train = datasets['train'].shuffle().select(range(1000))
dataset_test = datasets['validation'].shuffle().select(range(200))

del datasets

dataset_train
```

```
Loading cached processed dataset at data/glue_sst2/train/cache-
8cfcfd522c905812.arrow
```

```
Loading cached processed dataset at data/glue_sst2/train/cache-
b3e205139be224a8.arrow
```

```
Loading cached processed dataset at data/glue_sst2/train/cache-
890424ad8ed86a19.arrow
```

```
Loading cached processed dataset at data/glue_sst2/train/cache-
de855cf12202060d.arrow
```

```
Loading cached processed dataset at data/glue_sst2/validation/cache-
1de25e78c7da64ff.arrow
```

```
Loading cached processed dataset at data/glue_sst2/validation/cache-
620f9bf015596baf.arrow
```

```
Loading cached processed dataset at data/glue_sst2/validation/cache-
0a0b01f98a68ed20.arrow
```

```
Loading cached processed dataset at data/glue_sst2/validation/cache-
f7f25a94b8aafa52.arrow
```

```
Loading cached processed dataset at data/glue_sst2/test/cache-
524ecec34978f89b.arrow
```

```
Loading cached processed dataset at data/glue_sst2/test/cache-
86844412c2345cce.arrow
```

```
Loading cached processed dataset at data/glue_sst2/test/cache-
632228277b828dc1.arrow
```

```
Loading cached processed dataset at data/glue_sst2/test/cache-
fd22071d68ee727d.arrow
```

```
Dataset({
    features: ['sentence', 'label', 'idx', 'input_ids', 'token_type_ids',
'attention_mask'],
    num_rows: 1000
})
```

```
from transformers import AutoModelForSequenceClassification

#加载模型
model = AutoModelForSequenceClassification.from_pretrained('bert-base-cased',
                                                           num_labels=2)

print(sum([i.nelement() for i in model.parameters()]) / 10000)
```

```
Some weights of the model checkpoint at bert-base-cased were not used when
initializing BertForSequenceClassification:
['cls.predictions.transform.LayerNorm.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.bias',
'cls.predictions.transform.dense.weight', 'cls.seq_relationship.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight']
- This IS expected if you are initializing BertForSequenceClassification from the
checkpoint of a model trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForSequenceClassification from
the checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model
checkpoint at bert-base-cased and are newly initialized: ['classifier.bias',
'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
10831.181
```

```
import numpy as np
from datasets import load_metric
from transformers.trainer_utils import EvalPrediction

#加载评价函数
#有时会因为网络问题卡主,反复尝试会成功的
metric = load_metric('accuracy')


#定义评价函数
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    logits = logits.argmax(axis=1)
    return metric.compute(predictions=logits, references=labels)


#模拟测试输出
eval_pred = EvalPrediction(
    predictions=np.array([[0, 1], [2, 3], [4, 5], [6, 7]]),
    label_ids=np.array([1, 1, 1, 1]),
)
```

```
compute_metrics(eval_pred)
```

```
{'accuracy': 1.0}
```

```
from transformers import TrainingArguments, Trainer

#初始化训练参数
args = TrainingArguments(output_dir='./output_dir',
                         evaluation_strategy='epoch',
                         no_cuda=True)
args.num_train_epochs = 1
args.learning_rate = 1e-4
args.weight_decay = 1e-2
args.per_device_eval_batch_size = 32
args.per_device_train_batch_size = 16

#初始化训练器
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=dataset_train,
    eval_dataset=dataset_test,
    compute_metrics=compute_metrics,
)

#评价模型
trainer.evaluate()
```

```
The following columns in the evaluation set  don't have a corresponding argument
in `BertForSequenceClassification.forward` and have been ignored: sentence, idx.
If sentence, idx are not expected by `BertForSequenceClassification.forward`,
you can safely ignore this message.
***** Running Evaluation *****
  Num examples = 200
  Batch size = 32
```

```
<div>

  <progress value='14' max='7' style='width:300px; height:20px; vertical-align:
middle;'></progress>
  [7/7 01:35]
</div>
```

```
{'eval_loss': 0.7613461017608643,
 'eval_accuracy': 0.505,
 'eval_runtime': 4.5246,
 'eval_samples_per_second': 44.202,
 'eval_steps_per_second': 1.547}
```

```
#训练
trainer.train()
```

```
The following columns in the training set  don't have a corresponding argument in `BertForSequenceClassification.forward` and have been ignored: sentence, idx. If sentence, idx are not expected by `BertForSequenceClassification.forward`,  you can safely ignore this message.
/root/anaconda3/envs/pt36/lib/python3.6/site-packages/transformers/optimization.py:309: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this warning
  FutureWarning,
***** Running training *****
  Num examples = 1000
  Num Epochs = 1
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 63
```

```
<div>

  <progress value='63' max='63' style='width:300px; height:20px; vertical-align: middle;'></progress>
  [63/63 01:30, Epoch 1/1]
</div>
<table border="1" class="dataframe">
```

Epoch
Training Loss
Validation Loss
Accuracy

1
No log
0.462459

0.790000

```
The following columns in the evaluation set  don't have a corresponding argument
in `BertForSequenceClassification.forward` and have been ignored: sentence, idx.
If sentence, idx are not expected by `BertForSequenceClassification.forward`,
you can safely ignore this message.
***** Running Evaluation *****
  Num examples = 200
  Batch size = 32
```

```
Training completed. Do not forget to share your model on huggingface.co/models
=)
```

```
TrainOutput(global_step=63, training_loss=0.5899859837123326, metrics=
{'train_runtime': 92.1116, 'train_samples_per_second': 10.856,
'train_steps_per_second': 0.684, 'total_flos': 15416663400000.0, 'train_loss':
0.5899859837123326, 'epoch': 1.0})
```

```
#评价模型
trainer.evaluate()
```

```
The following columns in the evaluation set  don't have a corresponding argument
in `BertForSequenceClassification.forward` and have been ignored: sentence, idx.
If sentence, idx are not expected by `BertForSequenceClassification.forward`,
you can safely ignore this message.
***** Running Evaluation *****
  Num examples = 200
  Batch size = 32
```

```
<div>

  <progress value='7' max='7' style='width:300px; height:20px; vertical-align:
middle;'></progress>
  [7/7 00:03]
</div>
```

```
{'eval_loss': 0.46245864033699036,
 'eval_accuracy': 0.79,
 'eval_runtime': 4.2879,
 'eval_samples_per_second': 46.643,
 'eval_steps_per_second': 1.633,
 'epoch': 1.0}
```

```
#保存模型
trainer.save_model(output_dir='./output_dir')
```

```
Saving model checkpoint to ./output_dir
Configuration saved in ./output_dir/config.json
Model weights saved in ./output_dir/pytorch_model.bin
```

```
import torch


def collate_fn(data):
    label = [i['label'] for i in data]
    input_ids = [i['input_ids'] for i in data]
    token_type_ids = [i['token_type_ids'] for i in data]
    attention_mask = [i['attention_mask'] for i in data]

    label = torch.LongTensor(label)
    input_ids = torch.LongTensor(input_ids)
    token_type_ids = torch.LongTensor(token_type_ids)
    attention_mask = torch.LongTensor(attention_mask)

    return label, input_ids, token_type_ids, attention_mask


#数据加载器
loader_test = torch.utils.data.DataLoader(dataset=dataset_test,
                                          batch_size=4,
                                          collate_fn=collate_fn,
                                          shuffle=True,
                                          drop_last=True)

for i, (label, input_ids, token_type_ids,
        attention_mask) in enumerate(loader_test):
    break

label, input_ids, token_type_ids, attention_mask
```

```
(tensor([1, 0, 1, 1]),
 tensor([[  101,  1177,  1277,   175,  7409,  4759,  5531,   117,  1216, 10509,
          4133,   117,  1177,  1376,  2523,   119,   102,     0,     0,     0,
```

```
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0],
         [  101,  1175,  1132,  4928,  7996,  1992,  1536,  1111,   188,  2522,
           1358,  1103,  7010, 17757,  1106, 11231,  1194,   119,   102,     0,
              0,     0,     0,     0,     0,     0,     0,     0,     0,     0],
         [  101,  1209,  1821,  5613,  1105,  5250, 14638, 16889, 25576,  6323,
           1107, 13858,  9165,   119,   102,     0,     0,     0,     0,     0,
              0,     0,     0,     0,     0,     0,     0,     0,     0,     0],
         [  101,  1141,  1104,  1103,  1167,  9998,  1482,   112,   188,  5558,
           1106,  1855, 13090,  1142,  1214,   119,   102,     0,     0,     0,
              0,     0,     0,     0,     0,     0,     0,     0,     0,
0]]),
 tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
         0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
         0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
         0, 0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
         0, 0, 0, 0, 0, 0]]),
 tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0,
         0, 0, 0, 0, 0, 0],
         [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0,
         0, 0, 0, 0, 0, 0],
         [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0,
         0, 0, 0, 0, 0, 0],
         [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0,
         0, 0, 0, 0, 0, 0]]))
```

```python
import torch


#测试
def test():
    #加载参数
    model.load_state_dict(torch.load('./output_dir/pytorch_model.bin'))

    model.eval()

    #运算
    out = model(input_ids=input_ids,
                token_type_ids=token_type_ids,
                attention_mask=attention_mask)

    #[4, 2] -> [4]
    out = out['logits'].argmax(dim=1)
```

```
    correct = (out == label).sum().item()

    return correct / len(label)


test()
```

```
0.75
```

11.中文分类_CUDA

```
#安装cuda版的torch,请参照https://pytorch.org/get-started/previous-versions
#pip install torch==1.10.1+cu113 -f
https://download.pytorch.org/whl/cu113/torch_stable.html

import torch

torch.__version__
```

```
'1.10.1+cu113'
```

```
#快速演示
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print('device=', device)

from transformers import BertModel

#加载预训练模型
pretrained = BertModel.from_pretrained('bert-base-chinese')
#需要移动到cuda上
pretrained.to(device)

#不训练,不需要计算梯度
for param in pretrained.parameters():
    param.requires_grad_(False)


#定义下游任务模型
class Model(torch.nn.Module):

    def __init__(self):
        super().__init__()
        self.fc = torch.nn.Linear(768, 2)
```

```
    def forward(self, input_ids, attention_mask, token_type_ids):
        with torch.no_grad():
            out = pretrained(input_ids=input_ids,
                             attention_mask=attention_mask,
                             token_type_ids=token_type_ids)

        out = self.fc(out.last_hidden_state[:, 0])
        out = out.softmax(dim=1)
        return out


model = Model()
#同样要移动到cuda
model.to(device)

#虚拟一批数据,需要把所有的数据都移动到cuda上
input_ids = torch.ones(16, 100).long().to(device)
attention_mask = torch.ones(16, 100).long().to(device)
token_type_ids = torch.ones(16, 100).long().to(device)
labels = torch.ones(16).long().to(device)

#试算
model(input_ids=input_ids,
      attention_mask=attention_mask,
      token_type_ids=token_type_ids).shape

#后面的计算和中文分类完全一样，只是放在了cuda上计算
```

```
device= cuda
```

```
Some weights of the model checkpoint at bert-base-chinese were not used when
initializing BertModel: ['cls.predictions.decoder.weight',
'cls.seq_relationship.bias', 'cls.predictions.transform.dense.weight',
'cls.seq_relationship.weight', 'cls.predictions.bias',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.LayerNorm.weight',
'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
```

```
torch.Size([16, 2])
```

```
from datasets import load_dataset, load_from_disk
```

```
#定义数据集
class Dataset(torch.utils.data.Dataset):

    def __init__(self, split):
        #self.dataset = load_dataset(path='seamew/ChnSentiCorp', split=split)
        self.dataset = load_from_disk('./data/ChnSentiCorp')[split]

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, i):
        text = self.dataset[i]['text']
        label = self.dataset[i]['label']

        return text, label


dataset = Dataset('train')

len(dataset), dataset[0]
```

```
(9600,
 ('选择珠江花园的原因就是方便，有电动扶梯直接到达海边，周围餐馆、食廊、商场、超市、摊位一应俱全。酒店装修一般，但还算整洁。 泳池在大堂的屋顶，因此很小，不过女儿倒是喜欢。 包的早餐是西式的，还算丰富。 服务吗，一般',
  1))
```

```
from transformers import BertTokenizer

#加载字典和分词工具
token = BertTokenizer.from_pretrained('bert-base-chinese')


def collate_fn(data):
    sents = [i[0] for i in data]
    labels = [i[1] for i in data]

    #编码
    data = token.batch_encode_plus(batch_text_or_text_pairs=sents,
                                   truncation=True,
                                   padding='max_length',
                                   max_length=500,
                                   return_tensors='pt',
                                   return_length=True)

    #input_ids:编码之后的数字
    #attention_mask:是补零的位置是0,其他位置是1
    input_ids = data['input_ids'].to(device)
```

```
    attention_mask = data['attention_mask'].to(device)
    token_type_ids = data['token_type_ids'].to(device)
    labels = torch.LongTensor(labels).to(device)

    #print(data['length'], data['length'].max())

    return input_ids, attention_mask, token_type_ids, labels


#数据加载器
loader = torch.utils.data.DataLoader(dataset=dataset,
                                     batch_size=16,
                                     collate_fn=collate_fn,
                                     shuffle=True,
                                     drop_last=True)

for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    break

print(len(loader))
input_ids.shape, attention_mask.shape, token_type_ids.shape, labels
```

```
600
```

```
(torch.Size([16, 500]),
 torch.Size([16, 500]),
 torch.Size([16, 500]),
 tensor([1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1], device='cuda:0'))
```

```
from transformers import AdamW

#训练
optimizer = AdamW(model.parameters(), lr=5e-4)
criterion = torch.nn.CrossEntropyLoss()

model.train()
for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    out = model(input_ids=input_ids,
                attention_mask=attention_mask,
                token_type_ids=token_type_ids)

    loss = criterion(out, labels)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
```

```
    if i % 5 == 0:
        out = out.argmax(dim=1)
        accuracy = (out == labels).sum().item() / len(labels)

        print(i, loss.item(), accuracy)

    if i == 100:
        break
```

```
0 0.7051423192024231 0.5625
```

```
/root/anaconda3/envs/pt36/lib/python3.6/site-
packages/transformers/optimization.py:309: FutureWarning: This implementation of
AdamW is deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to
disable this warning
  FutureWarning,
```

```
5 0.6951683759689331 0.375
10 0.6328134536743164 0.75
15 0.6661140322685242 0.5625
20 0.624814510345459 0.75
25 0.6466094851493835 0.5625
30 0.6576337218284607 0.5
35 0.5227590799331665 0.9375
40 0.5514883399009705 0.9375
45 0.5737294554710388 0.9375
50 0.6367558240890503 0.6875
55 0.5158730149269104 0.875
60 0.48895588517189026 0.9375
65 0.5335288047790527 0.875
70 0.5290642380714417 0.75
75 0.48458394408226013 0.9375
80 0.5176761150360107 0.875
85 0.5343820452690125 0.8125
90 0.4468529224395752 0.875
95 0.5393693447113037 0.8125
100 0.6096935272216797 0.75
```

```
#测试
def test():
    model.eval()
    correct = 0
    total = 0

    loader_test = torch.utils.data.DataLoader(dataset=Dataset('validation'),
                                              batch_size=32,
                                              collate_fn=collate_fn,
                                              shuffle=True,
                                              drop_last=True)
```

```
    for i, (input_ids, attention_mask, token_type_ids,
            labels) in enumerate(loader_test):

        if i == 5:
            break

        print(i)

        with torch.no_grad():
            out = model(input_ids=input_ids,
                        attention_mask=attention_mask,
                        token_type_ids=token_type_ids)

        out = out.argmax(dim=1)
        correct += (out == labels).sum().item()
        total += len(labels)

    print(correct / total)


test()
```

```
0
1
2
3
4
0.88125
```

扩展

"Transformers" 是一种强大的深度学习模型架构，用于处理序列数据，如文本和语言。它在自然语言处理（NLP）任务中取得了许多显著的突破，包括机器翻译、文本生成、情感分析等。下面是关于 Transformers 常用方法的详细介绍：

1. **自注意力机制（Self-Attention）**： 自注意力机制是 Transformers 架构的核心。在处理序列数据时，模型能够在不同位置上分配不同的注意力权重。自注意力允许模型考虑输入序列中的所有位置，并根据它们之间的关系进行加权。
2. **多头注意力（Multi-Head Attention）**： 多头注意力是一种扩展的自注意力机制，它将注意力机制应用到多个子空间中，然后将它们的结果连接起来。这样可以使模型在不同的表示空间中捕获不同的语义信息。
3. **位置编码（Positional Encoding）**： 由于自注意力机制本身不具备序列位置信息，需要将位置编码加入输入嵌入向量中，以便模型能够理解输入序列中各元素的顺序。
4. **编码器-解码器架构**： Transformers 常用于序列到序列的任务，如机器翻译。在这种情况下，模型通常采用编码器来处理输入序列，并使用解码器生成输出序列。编码器和解码器都包含多个自注意力层和前馈神经网络层。
5. **残差连接与层归一化**： Transformers 使用残差连接（residual connections）和层归一化（layer normalization）来帮助训练更深的网络。残差连接使梯度能够更轻松地通过深层网络传播，而层归一化有助于稳定训练过程。
6. **位置编码**： 位置编码用于将输入序列中元素的位置信息嵌入到词嵌入向量中。这允许模型在没有显式顺序输入的情况下了解序列的相对位置。

7. **前馈神经网络（Feedforward Neural Networks）**： Transformers 的每个子层都包括一个前馈神经网络层，用于在注意力机制之后对特征进行进一步的映射和转换。
8. **位置编码**： Transformers 中的位置编码是为了在输入嵌入中引入序列的位置信息。一般情况下，可采用不同的位置编码方式，如正弦和余弦函数编码等。
9. **学习率调度器**： 为了有效地训练 Transformers 模型，通常会使用学习率调度器来动态调整学习率。这有助于在训练初期更快地收敛，在训练后期更稳定地收敛。
10. **遮蔽的自注意力（Masked Self-Attention）**： 在序列生成任务中，解码器需要逐步生成输出序列。为了不让模型在生成某个位置的输出时看到后续位置的信息，会使用遮蔽的自注意力机制。
11. **Token Embeddings**: 输入序列中的每个单词或标记需要被嵌入为向量。这些向量可以是预训练的词向量，也可以在训练过程中一同学习。
12. **注意力权重可视化**： Transformers 的自注意力机制使得可以计算每个输入位置对其他位置的注意力权重。这种权重的可视化有助于理解模型在处理不同输入时的关注点。

Transformers 网络是一种用于处理序列数据的深度学习架构，最初在 "Attention is All You Need" 这篇论文中被提出，主要用于解决自然语言处理（NLP）任务。下面是 Transformers 网络的详细介绍：

1. **输入表示**： Transformers 网络的输入是一个由词嵌入向量组成的序列，每个词嵌入向量捕捉了输入序列中单词的语义信息。这些嵌入可以是从头开始训练的，也可以是预训练的词向量，如 Word2Vec、GloVe 或 BERT。
2. **位置编码**： 在输入序列中，位置信息对于序列的理解至关重要。为了将位置信息引入模型，通常会在词嵌入向量中添加位置编码，以便模型能够区分不同位置的单词。位置编码可以使用正弦和余弦函数、绝对位置嵌入等方式实现。
3. **编码器-解码器结构**： Transformers 网络通常包含编码器和解码器两部分。在序列到序列的任务（如机器翻译）中，编码器用于处理输入序列，解码器用于生成输出序列。每个编码器和解码器都由多个层组成。
4. **自注意力机制**： 自注意力机制是 Transformers 的核心机制之一。在自注意力中，每个输入位置都可以与序列中的其他位置进行交互，产生一个加权的表示，其中权重由输入之间的关系决定。多头注意力则是在不同的表示子空间上执行多个注意力计算。
5. **前馈神经网络**： 在自注意力层之后，还有一个前馈神经网络层。这个层对每个位置的表示进行全连接操作，增加了非线性变换能力。
6. **残差连接与层归一化**： 在每个子层中，都会应用残差连接和层归一化。残差连接将原始输入与子层的输出相加，有助于梯度传播和模型训练。层归一化则有助于稳定训练过程。
7. **遮蔽的自注意力**： 在解码器中，为了避免在生成输出时泄露未来信息，会使用遮蔽的自注意力机制。这确保了在生成序列的每一步时，模型只能看到已生成的部分。
8. **多任务学习和预训练**： Transformers 在 NLP 领域取得了巨大成功，部分原因在于它们可以通过多任务学习和预训练来学习通用的语义表示。预训练的模型可以在特定任务上进行微调，从而提高性能。
9. **学习率调度器**： Transformers 网络通常使用学习率调度器，如学习率衰减或变化的学习率，以便在训练过程中有效地调整学习率，以实现更好的收敛和性能。
10. **输出层**： 在解码器中，输出层通常是一个全连接层，将模型的输出映射为词汇表中的单词或标记。在不同的任务中，输出层的设计会有所不同。

总的来说，Transformers 网络通过自注意力机制和编码器-解码器架构，使得模型能够更好地捕捉序列数据中的关系和语义信息。这使得 Transformers 在各种 NLP 任务中都取得了卓越的成绩，并且已经被广泛应用于其他领域的序列数据处理任务。

Transformers 网络是一种用于处理序列数据的深度学习架构，最初在 "Attention is All You Need" 这篇论文中被提出，主要用于解决自然语言处理（NLP）任务。下面是 Transformers 网络的详细介绍：

1. **输入表示**： Transformers 网络的输入是一个由词嵌入向量组成的序列，每个词嵌入向量捕捉了输入序列中单词的语义信息。这些嵌入可以是从头开始训练的，也可以是预训练的词向量，如 Word2Vec、GloVe 或 BERT。
2. **位置编码**： 在输入序列中，位置信息对于序列的理解至关重要。为了将位置信息引入模型，通常会在词嵌入向量中添加位置编码，以便模型能够区分不同位置的单词。位置编码可以使用正弦和余弦函数、绝对位置嵌入等方式实现。
3. **编码器-解码器结构**： Transformers 网络通常包含编码器和解码器两部分。在序列到序列的任务（如机器翻译）中，编码器用于处理输入序列，解码器用于生成输出序列。每个编码器和解码器都由多个层组成。
4. **自注意力机制**： 自注意力机制是 Transformers 的核心机制之一。在自注意力中，每个输入位置都可以与序列中的其他位置进行交互，产生一个加权的表示，其中权重由输入之间的关系决定。多头注意力则是在不同的表示子空间上执行多个注意力计算。
5. **前馈神经网络**： 在自注意力层之后，还有一个前馈神经网络层。这个层对每个位置的表示进行全连接操作，增加了非线性变换能力。
6. **残差连接与层归一化**： 在每个子层中，都会应用残差连接和层归一化。残差连接将原始输入与子层的输出相加，有助于梯度传播和模型训练。层归一化则有助于稳定训练过程。
7. **遮蔽的自注意力**： 在解码器中，为了避免在生成输出时泄露未来信息，会使用遮蔽的自注意力机制。这确保了在生成序列的每一步时，模型只能看到已生成的部分。
8. **多任务学习和预训练**： Transformers 在 NLP 领域取得了巨大成功，部分原因在于它们可以通过多任务学习和预训练来学习通用的语义表示。预训练的模型可以在特定任务上进行微调，从而提高性能。
9. **学习率调度器**： Transformers 网络通常使用学习率调度器，如学习率衰减或变化的学习率，以便在训练过程中有效地调整学习率，以实现更好的收敛和性能。
10. **输出层**： 在解码器中，输出层通常是一个全连接层，将模型的输出映射为词汇表中的单词或标记。在不同的任务中，输出层的设计会有所不同。

总的来说，Transformers 网络通过自注意力机制和编码器-解码器架构，使得模型能够更好地捕捉序列数据中的关系和语义信息。这使得 Transformers 在各种 NLP 任务中都取得了卓越的成绩，并且已经被广泛应用于其他领域的序列数据处理任务。