transformers-keras

发布 *0.0.1*

ZhouYang Luo

2021 年 09 月 07 日

Contents:

CHAPTER 1

安装

`transformers-keras` 是使用 `Keras` 实现的基于 `Transformer` 模型的库，它可以加载预训练模型的权重，也实现了多个下游的 NLP 任务。

项目主页：transformers-keras

1.1 使用 pip 安装

transformers-keras 可以直接使用 pip 安装：

```
pip install -U transformers-keras
```

使用 pip 安装的方式会自动把所需要的依赖都安装好。

如果你需要手动安装这些依赖，可以查看下面的依赖列表：

- tensroflow>=2.0.0
- tensorflow-addons
- tokenizers
- seqeval

CHAPTER 2

快速入门

本章节将会带领大家快速入门 `transformers-keras` 的使用。

`transformers-keras` 功能强大，它可以：

- 加载不同的预训练模型权重
- 使用预训练模型微调下游任务

与其它类似的库相比，`transformers-keras` 有以下优势：

- 清晰简单的 API，使用纯粹的 `keras` 构建模型，没有任何多余的封装
- 常用任务的数据管道构建，准备好数据就可以开始训练模型，不需要担心数据处理逻辑
- 可以直接导出 `SavedModel` 格式的模型，使用 tensorflow/serving 直接部署
- 最小依赖，除了 `tensorflow`，不附带任何其它庞大的第三方库

2.1 加载预训练模型的权重

基于 **BERT** 的模型支持加载以下预训练权重：

- 所有使用 google-research/bert 训练的 **BERT** 模型
- 所有使用 ymcui/Chinese-BERT-wwm 训练的 **BERT** 和 **RoBERTa** 模型

基于 **ALBERT** 的模型支持加载以下预训练权重:

- 所有使用 google-research/albert 训练的 **ALBERT** 模型

这里是基于 **BERT** 的模型的使用示例。所有基于 **ALBERT** 的模型用法和 BERT 类似，所以这里只使用 **BERT** 为例，不再重复用 ALBERT 举例。

2.1.1 BERT 特征抽取示例

我们这里直接加载预训练的 BERT 权重，来做句子特征的抽取。

```
from transformers_keras import Bert

# 加载预训练模型权重
model = Bert.from_pretrained('/path/to/pretrained/bert/model')
input_ids = tf.constant([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
segment_ids = tf.constant([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
attention_mask = tf.constant([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
sequence_output, pooled_output = model(inputs=[input_ids, segmet_ids, attention_mask],
↪ training=False)
```

通过上述代码，就可以抽取出模型的 `sequence_output` 和 `pooled_output` 特征向量。

其中:

- `sequence_output` 是 BERT 模型最后的输出状态，是一个形状为 `(batch_size, sequence_length, hidden_size)` 的张量。
- `pooled_output` 是 BERT 的 `[CLS]` 位置的向量经过 Dense 层得到的 `pooling` 输出。它是一个形状为 `(batch_size, hidden_size)` 的张量。

你可以通过这两个输出，采取不同的手段，来获取输入**句子**的向量。例如:

- 使用 `mean-pooling` 策略计算句子向量
- 使用 `[CLS]` 策略来获取句子向量 (`sequence_output[:, 0, )]` 即为 `[CLS]` 的向量表示)
- 使用 `pooled_output` 直接作为句子的向量

另外，可以通过构造器参数 `return_states=True` 和 `return_attention_weights=True` 来获取每一层的 `hidden_states` 和 `attention_weights` 输出:

```
from transformers_keras import Bert

# 加载预训练模型权重
model = Bert.from_pretrained(
    '/path/to/pretrained/bert/model',
    return_states=True,
    return_attention_weights=True)
input_ids = tf.constant([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
segment_ids = tf.constant([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

(下页继续)

(续上页)

```
attention_mask = tf.constant([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
sequence_outputs, pooled_output, hidden_states, attn_weights = model(inputs=[input_
↪ids, segment_ids, attention_mask], training=False)
```

其中:

- `hidden_states` 就是每一层 `hidden_state` stack 在一起的输出。它是一个形状为 `(batch_size, num_layers, sequence_length, hiddeb_size)` 的张量。
- `attn_weights` 就是每一层 `attention_weights` stack 在一起的输出。它是一个形状为 `(batch_size, num_layers, num_attention_heads, sequence_length, sequence_length)` 的张量。

2.2 微调下游任务示例

这里有以下几个示例:

- 使用 BERT 微调 **文本分类**任务
- 使用 BERT 微调 **问答**任务

2.2.1 使用 BERT 微调文本分类任务

你可以使用 BERT 构建序列的二分类网络:

```
from transformers_keras import BertForSequenceClassification

model = BertForSequenceClassification.from_pretrained('/path/to/pretrained/model')
model.summary()

model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['acc']
)
```

可以得到下面的模型输出:

```
Model: "bert_for_sequence_classification"
_________________________________________________________________________________
↪_____________
Layer (type)                 Output Shape              Param #     Connected to
↪
```

(下页继续)

(续上页)

```
==================================================================================================
input_ids (InputLayer)          [(None, None)]       0
__________________________________________________________________________________________________
segment_ids (InputLayer)        [(None, None)]       0
__________________________________________________________________________________________________
attention_mask (InputLayer)     [(None, None)]       0
__________________________________________________________________________________________________
bert (BertModel)                ((None, None, 768),  59740416    input_ids[0][0]
                                                                 segment_ids[0][0]
                                                                 attention_mask[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 2)            1538        bert[0][1]
==================================================================================================
Total params: 59,741,954
Trainable params: 59,741,954
Non-trainable params: 0
__________________________________________________________________________________________________
```

要训练网络，你需要准备训练数据。训练数据的格式采用 JSONL 格式，即文件的每一行都是一个 JSON。例如：

```
{"sequence": "我喜欢自然语言处理(NLP)", "label": 1}
{"sequence": "我不喜欢自然语言处理(NLP)", "label": 0}
```

需要注意的是，每个 **JSON** 都需要包含两个字段：

- sequence，即文本序列
- label，即文本的类别 ID

然后，就可以开始构造数据集，训练模型了：

```
from transformers_keras import SequenceClassificationDataset

input_files = [
    "filea.jsonl",
    "fileb.jsonl"
]

dataset = SequenceClassificationDataset.from_jsonl_files(
```

(下页继续)

(续上页)

```
    input_files=input_files,
    batch_size=32,
)
# 你可以查看dataset长什么样
print(next(iter(dataset)))

model.fit(
    dataset,
    epochs=10,
)
```

2.2.2 使用 BERT 微调问答任务

另一个例子，使用 BERT 来做 Question Answering：

```
from transformers_keras import BertForQuestionAnswering

model = BertForQuestionAnswering.from_pretrained('/path/to/pretrained/model')
model.summary()

model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['acc']
)
```

可以得到下面的模型输出：

```
Model: "bert_for_question_answering"
__________________________________________________________________________________________________
 ↪_______________
Layer (type)                    Output Shape         Param #     Connected to                     ␣
 ↪
==================================================================================================
input_ids (InputLayer)          [(None, None)]       0
__________________________________________________________________________________________________
 ↪_______________
segment_ids (InputLayer)        [(None, None)]       0
__________________________________________________________________________________________________
 ↪_______________
attention_mask (InputLayer)     [(None, None)]       0
```

(下页继续)

(续上页)

```
__________________________________________________________________________________________
↪______________
bert (BertModel)                ((None, None, 768),  59740416    input_ids[0][0]
                                                                 segment_ids[0][0]
                                                                 attention_mask[0][0]
__________________________________________________________________________________________
↪______________
dense (Dense)                   (None, None, 2)      1538        bert[0][0]
__________________________________________________________________________________________
↪______________
head (Lambda)                   (None, None)         0           dense[0][0]
__________________________________________________________________________________________
↪______________
tail (Lambda)                   (None, None)         0           dense[0][0]
==========================================================================================
Total params: 59,741,954
Trainable params: 59,741,954
Non-trainable params: 0
__________________________________________________________________________________________
↪______________
```

同样的，训练数据使用 JSONL 格式。每一行都是一个 JSON，例如：

```
{"question": "距离地球最近的天体是", "context": "月亮,你非要选的话,选c好了", "answer
↪": "月亮"}
{"question": "从月球上看地球的唯一建筑物是", "context":
↪"从月球上看地球的唯一建筑物是:中国的万里长城", "answer": "万里长城"}
```

其中，每个 JSON 需要包含以下字段：

- context，即上下文
- question，即问题
- answer，即答案

准备好数据集，就可以开始训练了：

```
from transformers_keras import QuestionAnsweringDataset

input_files = ["filea.jsonl", "fileb.jsonl"]

dataset = QuestionAnsweringDataset.from_jsonl_files(
    input_files=input_files,
    batch_size=32,
)
```

(下页继续)

(续上页)

```
# 你可以查看dataset长什么样
print(next(iter(dataset)))

model.fit(
    dataset,
    epochs=10,
)
```

transformers-keras 还支持其它不同的任务：

- 序列标注，例如 NER、POS
- 句向量，例如 SimCSE

这些任务会在下面的文档里介绍 ~。

CHAPTER 3

下游任务模型概述

3.1 统一的 API

3.2 Question Answering 模型介绍

3.3 Sentence Embedding 模型介绍

3.4 Sentiment Analysis 模型介绍

细粒度的情感分析，即 `Aspect-based Sentiment Analysis`，通常会把它作为一个**序列标注任务**。也就是说，会对一个序列的 **Aspect Term** 和 **Opinion Term** 进行序列标注，例如使用常见的 **B-I-O** 标注。标注过程中，可以同时对 Aspect 进行分类，对 Opinion 进行情感极性分类。

这种方式有一个缺点就是，对于类别经常变化的情况不太友好，一般来说需要重新训练模型，或者干脆对每一个领域训练一个单独的序列标注模型。总之，这种方式还是显得比较麻烦。

我本人更喜欢以下的方式：把 Aspect Term Extraction、Opinion Term Extraction、Opinion Sentiment Classification 三个任务，都使用 Question Answering 的方式来处理。

也就是说，Aspect-based Sentiment Analysis 可以分成两个部分：

- Aspect Term Extraction，只需要抽出 Aspect Term，不需要进行分类
- Opinion Term Extraction & Classification，需要抽取出 Opinion Term，同时进行情感极性分类

这两个部分使用两个独立的模型来处理，但是都是使用 Question Answering 的方式基于 BERT 模型实现。

3.4.1 Aspect Term Extraction

3.4.2 Opinion Term Extraction and Classification

3.5 Sequence Classification 模型介绍

3.6 Token Classification 模型介绍

CHAPTER 4

训练数据输入管道

本库提供了**方便**、**统**一、**高性能**的数据输入管道，只要按照一定格式准备好文件，即可构建出训练数据输入管道，立即开始模型训练。

未完待续…

4.1 统一的 API

4.2 Question Answering 模型的数据输入管道

4.3 Sentence Embedding 模型的数据输入管道

4.4 Sentiment Analysis 模型的数据输入管道

4.5 Sequence Classification 模型的数据输入管道

4.6 Token Classification 模型的输入输入管道

CHAPTER 5

完整解决方案示例

CHAPTER 6

进阶使用

支持的高级使用方法:

- 加载预训练模型权重的过程中跳过一些参数的权重
- 加载第三方实现的模型的权重

6.1 跳过一些预训练模型的权重

有些情况下，你可能会在加载预训练权重的过程中，跳过一些权重的加载。这个过程很简单。

这里是一个示例:

```
from transformers_keras import Bert, Albert

ALBERT_MODEL_PATH = '/path/to/albert/model'
albert = Albert.from_pretrained(
    ALBERT_MODEL_PATH,
    # return_states=False,
    # return_attention_weights=False,
    skip_token_embedding=True,
    skip_position_embedding=True,
    skip_segment_embedding=True,
    skip_pooler=True,
    ...
    )
```

(下页继续)

(续上页)

```
BERT_MODEL_PATH = '/path/to/bert/model'
bert = Bert.from_pretrained(
    BERT_MODEL_PATH,
    # return_states=False,
    # return_attention_weights=False,
    skip_token_embedding=True,
    skip_position_embedding=True,
    skip_segment_embedding=True,
    skip_pooler=True,
    ...
    )
```

所有支持跳过加载的权重如下:

- `skip_token_embedding`, 跳过加载 ckpt 的 `token_embedding` 权重
- `skip_position_embedding`, 跳过加载 ckpt 的 `position_embedding` 权重
- `skip_segment_embedding`, 跳过加载 ckpt 的 `token_type_emebdding` 权重
- `skip_embedding_layernorm`, 跳过加载 ckpt 的 `layer_norm` 权重
- `skip_pooler`, 跳过加载 ckpt 的 `pooler` 权重

6.2 加载第三方实现的模型的权重

在有一些情况下，第三方实现了一些模型，它的权重的结构组织和官方的实现不太一样。对于一般的预训练加载库，实现这个功能是需要库本身修改代码来实现的。本库通过 **适配器模式**提供了这种支持。用户只需要继承 **AbstractAdapter** 即可实现自定义的权重加载逻辑。

```
from transformers_keras.adapters import AbstractAdapter
from transformers_keras import Bert, Albert

# 自定义的BERT权重适配器
class MyBertAdapter(AbstractAdapter):

    def adapte_config(self, config_file, **kwargs):
        # 在这里把配置文件的配置项，转化成本库的BERT需要的配置
        # 本库实现的BERT所需参数都在构造器里，可以简单方便得查看
        pass

    def adapte_weights(self, model, config, ckpt, **kwargs):
        # 在这里把ckpt的权重设置到model的权重里
```

(下页继续)

(续上页)

```
        # 可以参考BertAdapter的实现过程
        pass

# 加载预训练权重的时候，指定自己的适配器 `adapter=MyBertAdapter()`
bert = Bert.from_pretrained('/path/to/your/bert/model', adapter=MyBertAdapter())

# 自定义的ALBERT权重适配器
class MyAlbertAdapter(AbstractAdapter):

    def adapte_config(self, config_file, **kwargs):
        # 在这里把配置文件的配置项，转化成本库的BERT需要的配置
        # 本库实现的ALBERT所需参数都在构造器里，可以简单方便得查看
        pass

    def adapte_weights(self, model, config, ckpt, **kwargs):
        # 在这里把ckpt的权重设置到model的权重里
        # 可以参考AlbertAdapter的实现过程
        pass

# 加载预训练权重的时候，指定自己的适配器 `adapter=MyAlbertAdapter()`
albert = Albert.from_pretrained('/path/to/your/albert/model',␣
↪adapter=MyAlbertAdapter())
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search