transformers 操作篇

来自： AiGC面试宝典

宁静致远

2024年01月28日 13:20

1. 如何 利用 transformers 加载 Bert 模型?

```
import torch
from transformers import BertModel, BertTokenizer
# 这里我们调用bert-base模型，同时模型的词典经过小写处理
model_name = 'bert-base-uncased'
# 读取模型对应的tokenizer
tokenizer = BertTokenizer.from_pretrained(model_name)
# 载入模型
model = BertModel.from_pretrained(model_name)
# 输入文本
input_text = "Here is some text to encode"
# 通过tokenizer把文本变成 token_id
input_ids = tokenizer.encode(input_text, add_special_tokens=True)
# input_ids: [101, 2182, 2003, 2070, 3793, 2000, 4372, 16044, 102]
input_ids = torch.tensor([input_ids])
# 获得BERT模型最后一个隐层结果
with torch.no_grad():
    last_hidden_states = model(input_ids)[0]  # Models outputs are now tuples

""" tensor([[[-0.0549,  0.1053, -0.1065,  ..., -0.3550,  0.0686,  0.6506],
         [-0.5759, -0.3650, -0.1383,  ..., -0.6782,  0.2092, -0.1639],
         [-0.1641, -0.5597,  0.0150,  ..., -0.1603, -0.1346,  0.6216],
         ...,
         [ 0.2448,  0.1254,  0.1587,  ..., -0.2749, -0.1163,  0.8809],
         [ 0.0481,  0.4950, -0.2827,  ..., -0.6097, -0.1212,  0.2527],
         [ 0.9046,  0.2137, -0.5897,  ...,  0.3040, -0.6172, -0.1950]]])
    shape: (1, 9, 768)
"""
```

可以看到，包括import在内的不到十行代码，我们就实现了读取一个预训练过的BERT模型，来encode我们指定的一个文本，对文本的每一个token生成768维的向量。如果是二分类任务，我们

接下来就可以把第一个token也就是[CLS]的768维向量，接一个linear层，预测出分类的logits，或者根据标签进行训练。

2. 如何 利用 transformers 输出 Bert 指定 hidden_state?

Bert 默认是 十二层，但是有时候预训练时并不需要利用全部利用，而只需要预训练前面几层即可，此时该怎么做呢?
下载到bert-base-uncased的模型目录里面包含 配置文件 config.json, 该文件中包含 output_hidden_states，可以利用该参数来设置 编码器内隐藏层层数


```
{
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "type_vocab_size": 2,
  "vocab_size": 30522
}
```


3. BERT 获取最后一层或每一层网络的向量输出

• transformer 最后一层输出的结果

```
last_hidden_state：shape是(batch_size, sequence_length, hidden_size)，
hidden_size=768,它是模型最后一层输出的隐藏状态
pooler_output：shape是(batch_size, hidden_size)，这是序列的第一个
token(classification token)的最后一层的隐藏状态，它是由线性层和Tanh激活函数进一步处
理的，这个输出不是对输入的语义内容的一个很好的总结，对于整个输入序列的隐藏状态序列
的平均化或池化通常更好。
hidden_states：这是输出的一个可选项，如果输出，需要指定
config.output_hidden_states=True,它也是一个元组，它的第一个元素是embedding，其余元
素是各层的输出，每个元素的形状是(batch_size, sequence_length, hidden_size)
attentions：这也是输出的一个可选项，如果输出，需要指定
config.output_attentions=True,它也是一个元组，它的元素是每一层的注意力权重，用于计
算self-attention heads的加权平均值
```

• 获取每一层网络的向量输出

```
##最后一层的所有 token向量
outputs.last_hidden_state
## cls向量
```

```
outputs.pooler_output
## hidden_states，包括13层，第一层即索引0是输入embedding向量，后面1-12索引是每层的输出向量
hidden_states = outputs.hidden_states
embedding_output = hidden_states[0]
attention_hidden_states = hidden_states[1:]
```