

武汉大学国家网络安全学院实验报告

课程名称	内容安全实验	实验日期	2024-4-22
实验名称	实验 4：文本生成与检测		
姓名	学号	专业	班级
邓鹏	2021302181152	网络空间安全	6 班
目录			
一、实验目的及实验内容..... 2			
1.1 实验目的..... 2			
1.2 实验内容..... 2			
1.2.1 GPT2 相关原理.....2			
1.2.2 temperature、Top-K、Top-P 相关原理.....3			
1.2.3 GPT-2 的注意力机制.....3			
1.2.4 GPT-2 Output Detector 相关原理.....4			
二、实验环境..... 5			
三、实验步骤及结果分析..... 6			
3.1 transformers 的 GPT-2 模型以任意主题生成一段文本.....6			
3.2 调整参数 temperature、Top-K、Top-P.....8			
3.3 结合 openAI/GPT2 源码分析 GPT2 的注意力机制和作用..... 12			
3.4 GPT-2 Output Detector 进行文本检测.....15			
3.5 基于 transformers 框架微调模型.....16			
3.6 基于 GPT-2 Output Detector trainer，训练一个中文模型.....19			
四、出现的问题及解决方法..... 24			
五、个人小结..... 26			
六、教师评语及评分..... 26			

一、实验目的及实验内容

(本次实验的具体内容；必要的原理分析)

1.1 实验目的

1. 以任意主题生成一段文本，用 openAI/GPT2 或者 transformers 的 GPT2 模型皆可。
2. 调整参数 temperature、Top-K、Top-P 并重新执行任务 1，分析调参之后的影响，最好结合相关源代码进行分析，用 openAI/GPT2 或者 transformers 的 GPT2 模型皆可。
3. 结合 openAI/GPT2 源码分析 GPT2 的注意力机制和作用。
4. 运行 GPT-2 Output Detector 进行文本检测，分析文本检测原理，分析文本长度对检测结果的影响及原因。
5. 自选应用场景和数据集，基于 transformers 框架微调模型，并做相关评价分析。
6. 基于 GPT-2 Output Detector trainer，训练一个中文模型，并做相关评价分析。

1.2 实验内容

1.2.1 GPT2 相关原理

openAI/GPT-2 是 openAI 开发的一个基于 transform 的开源深度学习架构，它只使用了 transform 的 decoding 部分。下面介绍一下 GPT-2 的基本原理：

(1) 预训练与微调：

- 预训练：GPT-2 首先在大规模的数据集上进行无监督学习。这一阶段的训练不是针对特定的任务，而是让模型学会理解和生成自然语言。

- 微调：虽然 GPT-2 通常被用作无微调的生成模型，但它也可以通过在特定任务的数据集上进行额外训练来进行微调，以提高在特定任务上的表现。

(2) Transformer 架构：

- GPT-2 基于 Transformer 模型，这是一种专为处理序列数据（如文本）设计的神经网络架构。它主要依靠自注意力（Self-Attention）机制来捕捉输入数据中的各种关系。

- 自注意力机制：允许模型在处理某一单词时，考虑到输入序列中的所有其他单词，从而理解上下文关系。这一机制通过计算不同单词之间的注意力分数来实现，分数高的单词对当前单词的影响更大。

(3) 层次结构：

GPT-2 包含多个 Transformer 层。每一层都包含多头注意力机制和前馈神经网络。通过这些层的堆叠，模型能够学习复杂的语言特征和层次结构。

(4) 大规模的参数：

GPT-2 有不同版本, 参数量从 1.17 亿到 15.54 亿不等。参数越多, 模型的容量越大, 理论上可以捕获更加复杂的语言现象和生成更流畅的文本。

(5) 生成文本:

在生成文本时, GPT-2 使用了名为“核采样”(Nucleus Sampling)的技术, 这种方法在生成每个新词时只考虑概率较高的候选词, 从而保持文本的连贯性和多样性。

1.2.2 temperature、Top-K、Top-P 相关原理

GPT-2 模型可以使用不同的解码方法来生成文本, 这些解码方法决定了如何从模型的概率输出中选择单词, 形成连贯的文本。下面我详细解释各种解码策略及其在参数调整中的作用:

(1) 贪婪搜索 (Greedy Search)

贪婪搜索是最简单的解码方法, 它在每个时间步选择概率最高的单词。这种方法的问题是它可能导致重复的或非常通用的文本输出, 因为它总是选择当前最可能的选项, 而不考虑其他可能的选择可能带来的长期利益。

(2) Beam 搜索 (Beam Search)

Beam 搜索是一种更复杂的方法, 它在每一步保持多个 (由参数 `num_beams` 定义) 最可能的候选输出。这种方法可以提高输出文本的多样性和质量。除了跟踪多个候选项外, 还可以设置 `no_repeat_ngram_size` 参数来避免生成重复的 `n-gram`, 从而防止文本陷入循环重复。

(3) 采样方法

与贪婪和 Beam 搜索的确定性输出不同, 采样方法通过随机选择下一个单词来增加输出的多样性, 这种选择是根据模型预测的概率分布加权的。

- 温度 (Temperature): 通过调整温度参数, 我们可以控制选择下一个词的随机性。温度较低 (<1) 使模型倾向于选择概率较高的词, 而温度较高 (>1) 则增加了低概率词被选中的机会。

- Top-k 采样: 这种方法只从最可能的 k 个词中选择下一个词。通过限制选择的范围, 可以避免低概率的、可能导致不连贯文本的词被选中。

- Top-p 采样: 与 Top-k 相似, Top-p 采样选择累计概率高于某个阈值 p 的最小集合。这样做可以动态调整候选词的数量, 适应模型在不同上下文下的预测差异。

(4) 联合使用采样技术

在实践中, 我们经常将上述技术组合使用来平衡随机性和连贯性。例如, 我们可以同时使用 Top-k 和 Top-p 采样, 加上一个温度参数, 以控制生成文本的多样性和质量。

通过这些解码技术和参数调整, 我们可以有效地控制语言模型的输出, 使其既多样化又符合语境。

1.2.3 GPT-2 的注意力机制

GPT-2 利用的核心技术是 Transformer 模型的注意力机制, 特别是自注意力 (self-attention) 机制。这种机制能够让模型在处理一个词时, 考虑到句子中的所有其他词, 从而有效地捕获语言中的复杂依赖关系。以下是自注意力机制的详细介绍:

(1) 自注意力机制

自注意力是一种特殊形式的注意力机制，它允许输入序列的每个元素都与序列中的其他所有元素进行比较。这种比较可以揭示元素之间的关系，是处理序列数据特别有效的方法。在 GPT-2 中，这种机制用于捕捉文本中的语义和语法依赖关系。

工作原理：

- 输入表示：首先，输入序列（如单词或词素）被转换为固定大小的向量，通常通过嵌入层完成。

- 查询（Q）、键（K）、值（V）：每个输入向量被转换生成三个不同的向量：查询向量（Q）、键向量（K）和值向量（V）。这些转换通常通过乘以训练得来的权重矩阵实现。

- 打分：自注意力通过计算每个查询向量与所有键向量的点积来计算分数，表示输入序列中每个元素对于序列中其他元素的重要性。

- 缩放点积注意力：得分会被缩放处理，通常是除以键向量维度的平方根，这有助于稳定梯度的学习。

- Softmax 归一化：通过 softmax 函数对这些分数进行归一化，使它们的和为 1。这样每个分数都可以被解释为一个概率，表示目标位置在生成每个单词时应该给予每个单词多少注意力。

- 加权和：将归一化的注意力分数与对应的值向量相乘，并将结果求和，得到该位置的输出向量。

(2) 多头注意力

GPT-2 在其自注意力层中使用了所谓的多头注意力，这意味着上述自注意力过程不是进行一次，而是独立地进行多次。每个“头”学习输入数据的不同方面，然后将这些学习的方面合并起来，以获得更全面的理解。

多样性：

- 每个注意力头可能会关注输入序列中的不同方面（例如，一个头可能专注于捕获语法结构，而另一个头可能探索词间的长距离依赖）。

- 输出向量是每个头输出的拼接或平均，这取决于具体实现的细节。

总之，自注意力机制是 GPT-2 强大能力的基础，使得它在处理各种语言模型任务时都表现出色，提供了深入且广泛的语义理解。

1.2.4 GPT-2 Output Detector 相关原理

openAI/GPT-2 Output Detector 是针对 GPT2 生成文本的检测器，该模型是对 RoBERTa 模型进行微调而获得。

其步骤主要包括：

- 数据准备：收集大量由 GPT-2 生成的文本和相应数量的真实人类生成文本。这些文本需要进行适当的预处理，以适配 RoBERTa 模型的输入需求。

- 模型微调：使用上述准备好的数据对 RoBERTa 模型进行微调。微调通常在有监督的学习框架中进行，即给定文本输入和标签（生成文本或真实文本）来训练模型。RoBERTa 模型在这一过程中学习区分两类文本的特征。

- 特征学习：由于 RoBERTa 模型是基于 Transformer 架构，它能够捕捉文本中复杂的依赖关系和上下文信息。在微调过程中，模型会学习到哪些特征更有可能表明文本是由机器生成的。例如，文本中的某些重复模式、非自然的语句结构或使用特定词汇的频率等。

- 概率输出：微调后的模型能够为每个输入文本输出一个概率分数，指示该文本被判定为机器生成的可能性。这使得用户可以根据具体应用场景和需要的置信度设置阈值。

通过这种方式，GPT-2 Output Detector 可以有效地识别由 GPT-2 或类似的语言模型生成的文本。

二、实验环境

（本次实验所使用的开发环境、依赖包等的情况）

Anaconda3 + jupyter + python==3.8.13 + tensorflow==2.10.0 + torch==2.2.2+cpu + transformers==4.40.0

Transformers 的版本选择是根据资料选取的,如下

<https://github.com/openai/gpt-2-output-dataset/issues/28>



三、实验步骤及结果分析

（详细描述实验步骤，并根据具体实验，记录、整理相应的数据表格等，对实验结果进行分析）

3.1 transformers 的 GPT-2 模型以任意主题生成一段文本

语言模型是一种机器学习模型，能够查看句子的一部分并预测下一个单词或单词序列。类似于 iPhone/Android 上的自动填充功能，GPT-2 可以在更大更复杂的范围内进行下一个词的预测。例如，最小的 GPT-2 有 1.17 亿参数，最大的（对公众不可见的）有超过 15 亿参数。对公众可用的最大模型是他们主要 GPT-2 模型的一半大小。

无论是在 PyTorch 还是 TensorFlow 中，Transformers 库使得 GPT-2 模型的导入变得非常简单——我将使用 TensorFlow。GPT-2 模型及其 Tokenizer 可以从 transformers 库中导入。

从输入序列开始：

```
In [1]: # 为了可重复性
        SEED = 34

        # 输出文本的最大单词数
        MAX_LEN = 70

In [2]: input_sequence = "I don't know about you, but there's only one thing I want to do after a long day of work"
```

我选择的是可用的最大 GPT-2 模型，当然也可以选择其他大小的模型。

```
In [3]: # 导入 transformers
        from transformers import TFGPT2LMHeadModel, GPT2Tokenizer

        # 获取大型 GPT2 tokenizer 和 GPT2 模型
        local_model_dir = 'D:/Content_Secu/task4/gpt2-large'

        # 从本地文件夹加载分词器和模型
        tokenizer = GPT2Tokenizer.from_pretrained(local_model_dir)
        GPT2 = TFGPT2LMHeadModel.from_pretrained(local_model_dir, pad_token_id=tokenizer.eos_token_id)

        #tokenizer = GPT2Tokenizer.from_pretrained("gpt2-medium")
        #GPT2 = TFGPT2LMHeadModel.from_pretrained("gpt2-medium", pad_token_id=tokenizer.eos_token_id)

        #tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
        #GPT2 = TFGPT2LMHeadModel.from_pretrained("gpt2", pad_token_id=tokenizer.eos_token_id)

        # 查看模型参数
        GPT2.summary()

All PyTorch model weights were used when initializing TFGPT2LMHeadModel.

All the weights of TFGPT2LMHeadModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFGPT2LMHeadModel for predictions without further training.

Model: "tfgpt2lm_head_model"

=====
Layer (type)           Output Shape           Param #
=====
transformer (TFGPT2MainLayer) multiple               774030080
=====

Total params: 774,030,080
Trainable params: 774,030,080
Non-trainable params: 0
=====
```

一般是可以直接从官网中将模型下载下来的，但是由于网络问题，我只能在 huggingface 中下载模型到本地调用。

接下来，我们选择最基础的解码方法，即贪婪搜索来生成文本。

第一次尝试（贪婪搜索）

使用贪婪搜索，概率最高的词被预测为下一个词，即每个时间步 t 更新下一个词为：

$$w_t = \operatorname{argmax}_w P(w|w_{1:t-1})$$

让我们看看这种简单方法的表现：

```
In [4]: # 导入深度学习基础库
import tensorflow as tf
tf.random.set_seed(SEED)

In [20]: # 用我们的开头来续写，使用 GPT-2 模型生成一个最大长度为 MAX_LEN 的文本
# 编码生成条件的上下文
input_ids = tokenizer.encode(input_sequence, return_tensors='tf')

# 生成文本直到输出长度（包括上下文长度）达到50
greedy_output = GPT2.generate(input_ids, max_length = MAX_LEN)

print("Output:\n" + 100 * '-')
print(tokenizer.decode(greedy_output[0], skip_special_tokens = True))
```

Output:

```
I don't know about you, but there's only one thing I want to do after a long day of work: go to the gym.

I'm not talking about the gym that's right next to my house. I'm talking about the gym that's right next to my office.

I'm not talking about the gym that's right next to my house. I'm talking about the gym that's right next to my office.

I'm not talking about the gym that's right next to my house. I'm talking about the gym that's right next to my office.

I'm not talking about the gym that's right next to my house. I'm talking about the gym that's right next to
```

可以发现文本的生成效果并不好，模型很快就开始重复自己了。

这也是贪婪搜索存在的问题，高概率的词可能被前面低概率的词所掩盖，所以模型无法探索更多样化的词组合。但是我们可以通过实施 Beam 搜索来防止这种情况发生。

Beam 搜索本质上是贪婪搜索，但模型在每个时间步跟踪并保留 `num_beams` 个假设，因此模型能够在生成文本时比较替代路径。我们还可以通过设置 `no_repeat_ngram_size = 2` 来包括 n-gram 惩罚，确保不会有 2-gram 重复出现。这里我们设置 `num_return_sequences = 5`，这样我们可以看到其他 5 个 beam 的样子。

带N-gram惩罚的Beam搜索

Beam搜索本质上是贪婪搜索，但模型在每个时间步跟踪并保留 `num_beams` 个假设，因此模型能够在生成文本时比较替代路径。我们还可以通过设置 `no_repeat_ngram_size = 2` 来包括n-gram惩罚，确保不会有2-gram重复出现。我们还将设置 `num_return_sequences = 5`，这样我们可以看到其他5个beam的样子。

修改 `generate` 函数中的一些参数以使用Beam搜索：

```
In [6]: # 修改 `generate` 函数中的一些参数以使用 Beam 搜索：
beam_outputs = GPT2.generate(
    input_ids,
    max_length = MAX_LEN,
    num_beams = 5,
    no_repeat_ngram_size = 2,
    num_return_sequences = 5,
    early_stopping = True
)

print('')
print("Output:\n" + 100 * '-')

# 使用了 GPT-2 模型的 Beam 搜索策略来生成文本，并展示五个不同的生成结果
for i, beam_output in enumerate(beam_outputs):
    print("{}: {}".format(i, tokenizer.decode(beam_output, skip_special_tokens=True)))
```

Output:

```
0: I don't know about you, but there's only one thing I want to do after a long day of work, and that's to sit down and watch a movie."

"I know, I know," you say. "But you're not going to like this one. It's not a good movie. I mean, it's
1: I don't know about you, but there's only one thing I want to do after a long day of work, and that's to sit down and watch a movie."

"I know, I know," you say. "But you're not going to like this one. It's about a guy who has a crush on a girl
2: I don't know about you, but there's only one thing I want to do after a long day of work, and that's to sit down and watch a movie."

"I know, I know," you say. "But you're not going to like this one. It's about a guy who has a crush on a woman
3: I don't know about you, but there's only one thing I want to do after a long day of work, and that's to sit down and watch a movie."

"I know, I know," you say. "But you're not going to like this one. It's about a guy who has a crush on a beautiful
4: I don't know about you, but there's only one thing I want to do after a long day of work, and that's to sit down and watch a movie."

"I know, I know," you say. "But you're not going to like this one. It's not a good movie. I'm not sure if
```

这样就好多了！5 个不同的 beam 假设几乎都是一样的，但如果我们增加 `num_beams`

的数量，我们会在不同的 beams 中看到更多的变化。当然，Beam 搜索也不是完美的。它在生成文本的长度大致恒定时表现得很好，比如翻译或总结这类问题，但对于像对话或故事生成这类开放式问题就不那么适用了（因为很难找到 num_beams 和 no_repeat_ngram_size 之间的平衡）。

3.2 调整参数 temperature、Top-K、Top-P

现在我们将探索非确定性解码——采样。与其沿着固定路径寻找概率最高的终点文本，不如随机根据其条件概率分布选择下一个词。

然而，当我们引入这种随机性时，生成的文本往往会变得不连贯，因此我们可以引入温度参数，这个参数可以增加高概率词出现的机会，同时减少低概率词的机会。

我们只需设置 do_sample = True 来实现随机采样，top_k = 0（暂不使用 Top-K 采样），temperature = 0.8 设置采样温度。

基本采样

现在我们将探索非确定性解码——采样。与其沿着固定路径寻找概率最高的终点文本，不如随机根据其条件概率分布选择下一个词：

$$w_t \sim P(w|w_{1:t-1})$$

然而，当我们引入这种随机性时，生成的文本往往会变得不连贯（更多信息见[这里](#)），因此我们可以引入温度参数，这个参数可以增加高概率词出现的机会，同时减少低概率词的机会：

我们只需设置 do_sample = True 来实现随机采样，top_k = 0（暂不使用 Top-K 采样），temperature = 0.8 设置采样温度，：

```
In [7]: # 温度参数用于控制输出文本的随机性。温度越高，输出越随机；温度越低，则趋于选择概率较高的词。
sample_output = GPT2.generate(
    input_ids,
    do_sample = True,
    max_length = MAX_LEN,
    top_k = 0,
    temperature = 0.8
)

print("Output:\n" + 100 * '-')
print(tokenizer.decode(sample_output[0], skip_special_tokens = True))
```

Output:

```
I don't know about you, but there's only one thing I want to do after a long day of work."

"Hmm. Must be quite the choice of words."

"Well, it's not a choice of words, but a need. I can't find the right answer until I find my answer."
```

可以发现生成的文本还是不太连贯。

接下来使用 Top-k 采样，这种方法只从最可能的 k 个词中选择下一个词。通过限制选择的范围，可以避免低概率的、可能导致不连贯文本的词被选中。

Top-K 采样

在 Top-K 采样中，选择最有可能的 k 个下一个词，并将整个概率质量转移到这些词上。因此我们不是增加高概率词出现的机会，而是完全移除低概率词：

设置top_k的值：

```
In [8]: # 只从最有可能的 k 个词中采样
sample_output = GPT2.generate(
    input_ids,
    do_sample = True,
    max_length = MAX_LEN,
    top_k = 50
)

print("Output:\n" + 100 * '-')
print(tokenizer.decode(sample_output[0], skip_special_tokens = True), '...')
```

Output:

```
I don't know about you, but there's only one thing I want to do after a long day of work. I want to get out of here and go jogging. To go jogging."

"That may be true, but I don't really have much money to spare!"

"That's true too. Why don ...
```

最后是 Top-p 采样，与 Top-k 相似，Top-p 采样选择累计概率高于某个阈值 p 的最小集合。这样做可以动态调整候选词的数量，适应模型在不同上下文下的预测差异。

Top-P 采样

Top-P 采样 (也称为核心采样) 类似于 **Top-K**, 但不是选择最有可能的 k 个词, 我们选择累积概率大于 p 的最小词集, 然后将整个概率质量转移到这个集合的词上:

top_k=0,设置**top_p**的值:

```
In [9]: # 只从最有可能的 80% 词中采样
sample_output = GPT2.generate(
    input_ids,
    do_sample = True,
    max_length = MAX_LEN,
    top_p = 0.8,
    top_k = 0
)

print("Output:\n" + 100 * '-')
print(tokenizer.decode(sample_output[0], skip_special_tokens = True), '...')
```

Output:

I don't know about you, but there's only one thing I want to do after a long day of work: try out some dessert! Today I've got a total of four different fruit ice creams from The Baker's Dozen. I'm going to share three of them with you, each with a twist.

One was made ...

然后尝试 Top-k 与 Top-p 采样结合起来

Top-K 和 Top-P 采样

我们可以在这里同时使用 **Top-K** 和 **Top-P** 采样。这样可以减少我们得到奇怪词 (低概率词) 的机会, 同时允许动态选择大小。我们只需要为两者设置一个值。如果我们想, 我们甚至可以包含最初的温度参数, 现在让我们看看在添加这些参数后, 我们的模型表现如何。我们将检查前 5 个返回结果, 看看我们的答案有多少变化:

```
In [10]: # 结合两种采样技术
sample_outputs = GPT2.generate(
    input_ids,
    do_sample = True,
    max_length = 2*MAX_LEN,
    #temperature = .7,
    top_k = 50,
    top_p = 0.85,
    num_return_sequences = 5
)

print("Output:\n" + 100 * '-')
for i, sample_output in enumerate(sample_outputs):
    print('{}: {}'.format(i, tokenizer.decode(sample_output, skip_special_tokens = True)))
    print('')
    print('')
```

Output:

0: I don't know about you, but there's only one thing I want to do after a long day of work and this is one of it. I have to do something else. It's been quite an exciting couple of weeks at the office, haven't I?

Makes you wonder about the people who didn't get the memo that a long day of work is about to turn into a long day of fun...

1: I don't know about you, but there's only one thing I want to do after a long day of work: watch some movies on my bed!

So, I took a trip to my local mall to check out a new line of "couples" furniture. It's the same type of furniture that I saw on an episode of The Bachelor. It's the kind of furniture that makes me think that if I were to be on a reality TV show, I would be dating one of the characters from that show.

The first thing I noticed about the furniture was that there are no chairs. There is only one bed, a desk, a table, and two chairs...

2: I don't know about you, but there's only one thing I want to do after a long day of work, and that's to go on an adventure! You know, to see if I can get myself to the bottom of the hole before some kind of horrible death and/or injury happens to me. The one thing I know for sure is that I have never, ever, played in a hole of my own creation. I can say that from experience. The hole is my work. I had no choice but to use it as a playground. I used to do some of my most interesting and imaginative work on the hole.

When I say "my hole," I mean the...

3: I don't know about you, but there's only one thing I want to do after a long day of work.

Oh, how my soul hurts to think of it.

The only reason I'm able to stand to look at myself in the mirror, with my face so distorted, is because I know it's not me.

I was a little sad to hear that a person would die in the car accident on the way to the hospital.

It's the only way to express how I feel when you don't say the right words at the right time.

I had the opportunity to visit some of the most amazing people in the world in New York...

4: I don't know about you, but there's only one thing I want to do after a long day of work. I want to relax and watch TV and not worry about work.

My first stop on my weekend road trip is the nearest shopping mall.

We are on the way to my aunt's house. She's a retired teacher. I can't wait to spend my weekends with her.

After walking to the mall, I walked around the city. I looked around the different shopping centers and shops. There's so much to do in Japan.

I picked out a few shops to visit in this shopping mall.

One of them was a souvenir...

然后尝试 temperature、Top-k 与 Top-p 采样结合起来

temperature、Top-K 和 Top-P 采样

我们可以在这里同时使用 temperature、Top-K 和 Top-P 采样。这样可以减少我们得到奇怪词（低概率词）的机会，同时允许动态选择大小。我们只需要为两者设置一个值。如果我们想，我们甚至可以包含最初的温度参数，现在让我们看看在添加这些参数后，我们的模型表现如何。我们将检查前 5 个返回结果，看看我们的答案有多少变化：

```
In [11]: # 结合两种采样技术
sample_outputs = GPT2.generate(
    input_ids,
    do_sample = True,
    max_length = 2*MAX_LEN,
    temperature = 0.8,
    top_k = 50,
    top_p = 0.85,
    num_return_sequences = 5
)

print("Output:\n" + 100 * '-')
for i, sample_output in enumerate(sample_outputs):
    print("{}: {}".format(i, tokenizer.decode(sample_output, skip_special_tokens = True)))
    print('')
    print('')
```

Output:

0: I don't know about you, but there's only one thing I want to do after a long day of work: drink beer. But, if you're not into that, I've got some great suggestions for you.

1. Try a beer made with a non-alcoholic malt.

For example, this is the first beer I've ever made with a pilsner malt. I wanted a beer that was a bit less bitter than what I was used to. But this beer was delicious.

2. Go with a stout.

I think the most underrated beer in the world. I've been drinking it for years and I've always been impressed by...

1: I don't know about you, but there's only one thing I want to do after a long day of work: I want to go to the movies."

The first person I told about this was the manager of the movie theater I used to frequent. He laughed and said, "You mean you want to go to the movies on a Friday night?"

"Yeah, I do," I said. "I mean, I'll be able to see a movie on a Friday night if I work for a movie theater."

"What?" he said. "You want to go to the movies on a Friday night? You're not going to work in a movie theater?..."

2: I don't know about you, but there's only one thing I want to do after a long day of work: eat a little breakfast."

The man sat up straight, his face contorted with confusion.

"What? What are you talking about?"

"Just... eat a little breakfast. It's not that bad. And it'll make your day better. Now, get up and go."

The man rose from his chair and made his way to the door. He didn't even glance at the man who had just entered. He didn't even look at the door again. He just walked straight out.

I was a bit startled. What...

3: I don't know about you, but there's only one thing I want to do after a long day of work: get home and play some video games!"

While he's not going to be playing Mario Kart 8, he is going to be playing the game's upcoming multiplayer mode. In it, he'll be able to battle other players, and in doing so, he's hoping to improve his skills and get better at the game.

"I've been playing a lot of multiplayer games, but I haven't had much luck at it," he said. "I was playing a couple of games recently that really made me feel like I had a good shot at beating the game..."

4: I don't know about you, but there's only one thing I want to do after a long day of work: play some videogames. So I decided to start a new project in the form of a virtual reality game.

How did it begin?

I was browsing the internet and I saw a project called "Reverse Chronos". It was a video game which would take the player back in time. I knew I wanted to make a virtual reality game and I wanted to create a game in which the player would experience the experience of the time that he or she was in. I started to think about what the player would do if he or she was in the past...

最后,尝试不同的输入文本,看看效果

基准提示

在这里，我们将看到当给定一些更有趣的输入时，GPT-2 模型的表现如何。

```
In [12]: MAX_LEN = 150
```

"In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English."

```
In [13]: prompt1 = 'In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes'
input_ids = tokenizer.encode(prompt1, return_tensors='tf')
```

```
In [14]: sample_outputs = GPT2.generate(
        input_ids,
        do_sample = True,
        max_length = MAX_LEN,
        temperature = 0.8,
        top_k = 50,
        top_p = 0.85
        #num_return_sequences = 5
    )

print("Output:\n" + 100 * '-')
for i, sample_output in enumerate(sample_outputs):
    print("{}: {}".format(i, tokenizer.decode(sample_output, skip_special_tokens = True)))
    print('')
```

Output:

0: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

The discovery came when scientists headed to the area in 2011 to take a look for water. When they didn't find anything, they headed back to the location to get more water samples, which was when the strange creatures were spotted.

The team of researchers, led by Dr. Tom Tarnita of the National Geographic Society, says the animals were not a typical herd of unicorn s that could be found in many areas of the world, where they are thought to live. The animals had a white coat,...

"Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today."

```
In [21]: prompt2 = 'Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today.'
input_ids = tokenizer.encode(prompt2, return_tensors='tf')
```

```
In [22]: sample_outputs = GPT2.generate(
        input_ids,
        do_sample = True,
        max_length = MAX_LEN,
        temperature = 0.8,
        top_k = 50,
        top_p = 0.85
        #num_return_sequences = 5
    )

print("Output:\n" + 100 * '-')
for i, sample_output in enumerate(sample_outputs):
    print("{}: {}".format(i, tokenizer.decode(sample_output, skip_special_tokens = True)))
    print('')
```

Output:

0: Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today. The singer was caught in the act after a security guard saw her walk out of the store with a bag full of clothes. She was later arrested on suspicion of shoplifting.

The 21-year-old pop star was released on \$5,000 bond.

PHOTOS: The '90s' Most Shocking Celebrity Mug Shots

It is unclear whether the incident was a one-time incident or if Cyrus has a history of shoplifting in the past. She was seen walking out of the store with the bag full of clothes.

Cyrus was arrested last month after she was spotted in a Beverly Hills mall with a...

"Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry."

```
In [23]: prompt3 = 'Legolas and Giali advanced on the orcs, raising their weapons with a harrowing war cry.'
input_ids = tokenizer.encode(prompt3, return_tensors='tf')
```

```
In [24]: sample_outputs = GPT2.generate(
        input_ids,
        do_sample = True,
        max_length = MAX_LEN,
        temperature = 0.8,
        top_k = 50,
        top_p = 0.85
        #num_return_sequences = 5
    )

print("Output:\n" + 100 * '-')
for i, sample_output in enumerate(sample_outputs):
    print("{}: {}".format(i, tokenizer.decode(sample_output, skip_special_tokens = True)))
    print('')
```

Output:

0: Legolas and Gimli advanced on the orcs, raising their weapons with a harrowing war cry. As they passed through the battle, Giali took a step back, and saw that the orcs had a large group of humans, their heads covered with a red and black cloak. "We will not let them pass," he said. "This is the land of the orcs, and I will not let them pass." He then turned to the humans and spoke: "The people of the north have spoken. They have said that you are not welcome here." They turned to look at him. "Who are you?" they asked. "I am the lord of the north," he said. "The north is mine. The north is my home...."

"For today's homework assignment, please describe the reasons for the US Civil War."

```
In [25]: prompt4 = "For today's homework assignment, please describe the reasons for the US Civil War."
input_ids = tokenizer.encode(prompt4, return_tensors='tf')
```

```
In [26]: sample_outputs = GPT2.generate(
        input_ids,
        do_sample = True,
        max_length = MAX_LEN,
        temperature = 0.8,
        top_k = 50,
        top_p = 0.85
        #num_return_sequences = 5
    )

print("Output:\n" + 100 * '-')
for i, sample_output in enumerate(sample_outputs):
    print("{}: {}".format(i, tokenizer.decode(sample_output, skip_special_tokens = True)))
    print('')
```

Output:

0: For today's homework assignment, please describe the reasons for the US Civil War.

What do you think would have happened if the Civil War never happened?

Share your answer in the comments section below!...

3.3 结合 openAI/GPT2 源码分析 GPT2 的注意力机制和作用

将源码下载下来后,可以发现有关注意力机制的内容在 src/model.py 中,故重点分析这个文件的内容。

有关注意力机制的代码主要在 attn 函数中

```
# 注意力机制
1个用法
def attn(x, scope, n_state, *, past, hparams):
    assert x.shape.ndims == 3 # Should be [batch, sequence, features]
    assert n_state % hparams.n_head == 0
    if past is not None:
        assert past.shape.ndims == 5 # Should be [batch, 2, heads, sequence, features], where 2 is [k, v]

    # 分割头部
    def split_heads(x):
        # From [batch, sequence, features] to [batch, heads, sequence, features]
        return tf.transpose(split_states(x, hparams.n_head), [0, 2, 1, 3])

    # 合并头部
    def merge_heads(x):
        # Reverse of split_heads
        return merge_states(tf.transpose(x, [0, 2, 1, 3]))

    # 掩码注意力权重
    def mask_attn_weights(w):
        # w has shape [batch, heads, dst_sequence, src_sequence], where information flows from src to dst.
        _, _, nd, ns = shape_list(w)
        b = attention_mask(nd, ns, dtype=w.dtype)
        b = tf.reshape(b, [1, 1, nd, ns])
        w = w*b - tf.cast(1e10, w.dtype)*(1-b)
        return w
```

```

# 多头注意力
def multihead_attn(q, k, v):
    # q, k, v have shape [batch, heads, sequence, features]
    w = tf.matmul(q, k, transpose_b=True)
    w = w * tf.rsqrt(tf.cast(v.shape[-1].value, w.dtype))

    w = mask_attn_weights(w)
    w = softmax(w)
    a = tf.matmul(w, v)
    return a

with tf.variable_scope(scope):
    c = conv1d(x, scope='c_attn', n_state * 3) # 一次性计算 Q, K, V
    q, k, v = map(split_heads, tf.split(c, 3, axis=2)) # 分割 Q, K, V 到不同头
    present = tf.stack([k, v], axis=1)
    if past is not None:
        pk, pv = tf.unstack(past, axis=1)
        k = tf.concat([pk, k], axis=-2) # 连接 K 维度
        v = tf.concat([pv, v], axis=-2) # 连接 V 维度
    a = multihead_attn(q, k, v) # 计算多头注意力
    a = merge_heads(a) # 合并头部
    a = conv1d(a, scope='c_proj', n_state) # 最后的投影
    return a, present

```

其中，注意力机制的实现核心分为几个步骤：

(1) 线性转换 (Q, K, V 生成)

```

with tf.variable_scope(scope):
    c = conv1d(x, scope='c_attn', n_state * 3) # 一次性计算 Q, K, V
    q, k, v = map(split_heads, tf.split(c, 3, axis=2)) # 分割 Q, K, V 到不同头
    present = tf.stack([k, v], axis=1)
    if past is not None:

```

这里，conv1d 函数通过一个 1D 卷积层将输入 x 转换成三个不同的输出：查询 (Q)、键 (K) 和值 (V)。这三个输出的维度之和是输入维度的三倍，随后被等分，对应于 Q, K, 和 V。每个输出接着通过 split_heads 函数被切分成多个注意力头。

(2) 分割头部

```

# 分割头部
def split_heads(x):
    # From [batch, sequence, features] to [batch, heads, sequence, features]
    return tf.transpose(split_states(x, hparams.n_head), [0, 2, 1, 3])

```

在 split_heads 函数中，原始的批处理和序列维度被保持，而特征维度被分割成多个头。这允许模型并行地处理不同的表示子空间，每个头捕获输入数据的不同方面。

(3) 缩放点积注意力

```

# 多头注意力
def multihead_attn(q, k, v):
    # q, k, v have shape [batch, heads, sequence, features]
    w = tf.matmul(q, k, transpose_b=True)
    w = w * tf.rsqrt(tf.cast(v.shape[-1].value, w.dtype))

    w = mask_attn_weights(w)
    w = softmax(w)
    a = tf.matmul(w, v)
    return a

```


在这一步，通过对查询 (Q) 和键 (K) 的矩阵乘法计算注意力得分，然后使用特征维度的平方根进行缩放，这有助于避免训练期间梯度消失问题。

(4) 应用掩码

```
# 多头注意力
def multihead_attn(q, k, v):
    # q, k, v have shape [batch, heads, sequence, features]
    w = tf.matmul(q, k, transpose_b=True)
    w = w * tf.rsqrt(tf.cast(v.shape[-1].value, w.dtype))

    w = mask_attn_weights(w)
    w = softmax(w)
    a = tf.matmul(w, v)
    return a
```

`mask_attn_weights` 函数应用掩码来防止信息向错误的方向流动（如前面提到的，避免位置 i 注意到位置 $j > i$ 的信息）。这通过将掩码位置的得分减去一个非常大的数实现，使得 `softmax` 后这些位置的权重接近于零。

(5) 注意力权重归一化和输出

```
# 多头注意力
def multihead_attn(q, k, v):
    # q, k, v have shape [batch, heads, sequence, features]
    w = tf.matmul(q, k, transpose_b=True)
    w = w * tf.rsqrt(tf.cast(v.shape[-1].value, w.dtype))

    w = mask_attn_weights(w)
    w = softmax(w)
    a = tf.matmul(w, v)
    return a
```

归一化权重通过 `softmax` 计算得到，然后用这些权重来加权值 (V)，得到每个头的输出。这些输出代表了模型对输入数据的不同方面的加权表示。

(6) 合并头部

```
with tf.variable_scope(scope):
    c = conv1d(x, scope='c_attn', n_state * 3) # 一次性计算 Q, K, V
    q, k, v = map(split_heads, tf.split(c, 3, axis=2)) # 分割 Q, K, V 到不同头
    present = tf.stack([k, v], axis=1)
    if past is not None:
        pk, pv = tf.unstack(past, axis=1)
        k = tf.concat([pk, k], axis=-2) # 连接 K 维度
        v = tf.concat([pv, v], axis=-2) # 连接 V 维度
    a = multihead_attn(q, k, v) # 计算多头注意力
    a = merge_heads(a) # 合并头部
    a = conv1d(a, scope='c_proj', n_state) # 最后的投影
    return a, present
```

`merge_heads` 函数将多个头的输出合并回单个特征空间，然后通过另一个 1D 卷积 (`c_proj`) 投影回原始的特征维度。

具体来说,注意力机制会计算当前输入特征与每一个历史像素之间的相似度或相关性,相似度越高的像素被赋予更大的权重,表示其对当前输入特征更重要,需要更多注意;相似度越低的像素被赋予更小的权重,表示其影响较小,可以被过滤。

3.4 GPT-2 Output Detector 进行文本检测

下载源码后,根据 README.md 文件操作。

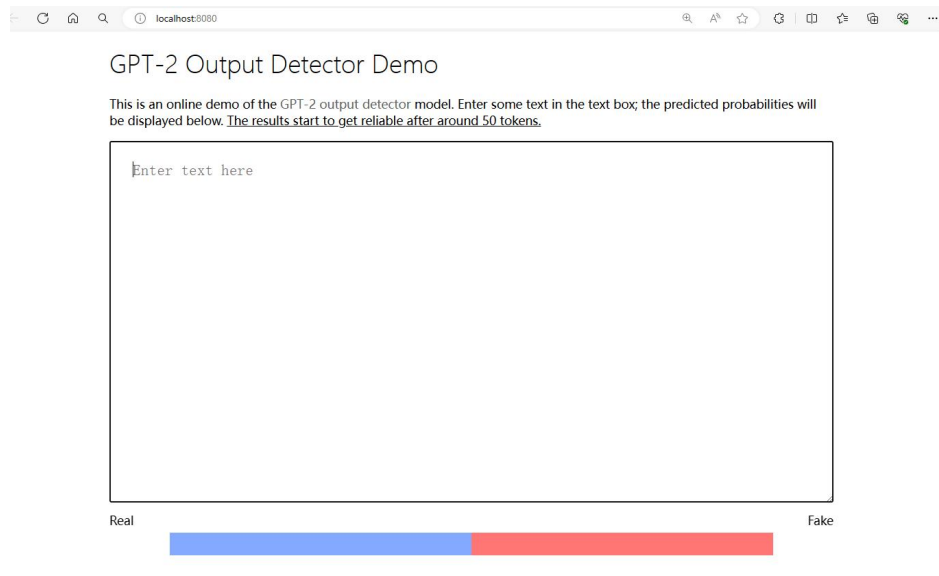
需要相应网站下载对应的模型 detector_dataset.pt,然后调试运行,中间遇到的问题,详见第四部分。

终端输入 `python -m detector.server detector-base.pt` 运行。

可以得到如下界面

```
(tensorflow) PS D:\Content_Secu\task4\gpt-2-output-dataset-master> python -m detector.server detector-base.pt
Loading checkpoint from detector-base.pt
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Starting HTTP server on port 8080
[ ] Process has started; loading the model ...
[ ] Ready to serve at http://localhost:8080
```

点击进入网页,如下



要求输入至少 50token,把我们在第一部分生成的文本拿来测试,如下

Top-K 和 Top-P 采样

我们可以在这里同时使用 Top-K 和 Top-P 采样。这样可以减少我们得到奇怪词（低概率词）的机会，同时允许动态选择大小。我们只需要为两者设置一个值。如果我们想，我们甚至可以包含最初的温度参数，现在让我们看看在添加这些参数后，我们的模型表现如何。我们将检查的 5 个返回结果，看看我们的答案有多少变化：

```
In [10]: # 结合两种采样技术
sample_outputs = GPT2.generate(
    input_ids,
    do_sample = True,
    max_length = 2*MAX_LEN,
    #temperature = .7,
    top_k = 50,
    top_p = 0.85,
    max_return_sequences = 5
)

print("Output:\n" + 100 * '-')
for i, sample_output in enumerate(sample_outputs):
    print('[]: {}'.format(i, tokenizer.decode(sample_output, skip_special_tokens = True)))
    print('')
    print('')
```

Output:

0: I don't know about you, but there's only one thing I want to do after a long day of work and this is one of it. I have to do something else. It's been quite an exciting couple of weeks at the office, haven't I?

Makes you wonder about the people who didn't get the memo that a long day of work is about to turn into a long day of fun...

1: I don't know about you, but there's only one thing I want to do after a long day of work: watch some movies on my bed!

So, I took a trip to my local mall to check out a new line of "couples" furniture. It's the same type of furniture that I saw on an episode of The Bachelor. It's the kind of furniture that makes me think that if I were to be on a reality TV show, I would be dating one of the characters from that show.

The first thing I noticed about the furniture was that there are no chairs. There is only one bed, a desk, a table, and two chairs...

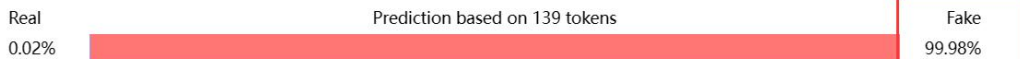
GPT-2 Output Detector Demo

This is an online demo of the GPT-2 output detector model. Enter some text in the text box; the predicted probabilities will be displayed below. The results start to get reliable after around 50 tokens.

I don't know about you, but there's only one thing I want to do after a long day of work: watch some movies on my bed!

So, I took a trip to my local mall to check out a new line of "couples" furniture. It's the same type of furniture that I saw on an episode of The Bachelor. It's the kind of furniture that makes me think that if I were to be on a reality TV show, I would be dating one of the characters from that show.

The first thing I noticed about the furniture was that there are no chairs. There is only one bed, a desk, a table, and two chairs...|



可见,预测成功!!!

3.5 基于 transformers 框架微调模型

这里选用的是之前舆情分析课程的大作业中的一部分——情感分析部分。

采用的数据集是之前小组收集标注的数据。

首先处理数据

```
# 加载分词器和模型
model_dir = 'F:/yuqing/Final/models/bert-sentiment-20231203-184759'
tokenizer = BertTokenizer.from_pretrained(model_dir)
model = BertForSequenceClassification.from_pretrained(model_dir)

# 使用gpu
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

1个用法
def extract_sentiments(texts):
    model.eval()
    sentiments = []
    for text in texts:
        inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=128)
        inputs = {k: v.to(device) for k, v in inputs.items()}

        with torch.no_grad():
            outputs = model(**inputs)

        prediction = torch.argmax(outputs.logits, dim=1).item()
        label_map = {0: 'neutral', 1: 'positive', 2: 'negative'}
        sentiments.append(label_map[prediction])
    return sentiments
```



```

# 加载数据
file_path = 'F:/yuqing/Final/data_process/processed_data.csv'
data = pd.read_csv(file_path)

# 提取文本并进行情感分析
texts = data['Text'].tolist()
sentiments = extract_sentiments(texts)

# 解析实体和关系
1个用法
def parse_entities_relations(row):
    if pd.isna(row['Entities']):
        return []
    else:
        entities = row['Entities'].split('; ')
        relations = row['Relations'].split('; ') if pd.notna(row['Relations']) else ['']*len(entities)
        return list(zip(entities, relations))

# 结合实体、关系和情感分析结果
rows = []
for index, row in data.iterrows():
    sentiment = sentiments[index]
    for entity, relation in parse_entities_relations(row):
        entity_type, entity_name = entity.split(': ')
        new_row = {
            'Entity': entity_name.strip(),
            'Relation': relation,
            'Sentiment': sentiment
        }
        rows.append(new_row)

graph_data = pd.DataFrame(rows)

# 显示结果
print(graph_data.head())

# 保存到新的CSV文件
output_file_path = 'F:/yuqing/Final/data_process/processed_data_sentiment.csv'
graph_data.to_csv(output_file_path, index=False)

```

接下来，就是利用处理好的数据训练模型

```

# 检查CUDA是否可用
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# 加载数据
file_path = 'F:/yuqing/Final/data/sentiment1.csv'
data = pd.read_csv(file_path)

# 情感标签映射
label_map = {'neutral': 0, 'positive': 1, 'negative': 2}
data['label'] = data['label'].map(label_map)

```

```

# 数据集类
2个用法
class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

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

    def __getitem__(self, item):
        text = str(self.texts[item])
        label = self.labels[item]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            return_attention_mask=True,
            return_tensors='pt',
            truncation=True
        )

        return {
            'text': text,
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# 分割数据集
train_data, test_data = train_test_split(*arrays: data, test_size=0.1)
train_texts = train_data['text'].tolist()
train_labels = train_data['label'].tolist()
test_texts = test_data['text'].tolist()
test_labels = test_data['label'].tolist()

# 参数设置
MAX_LEN = 128
BATCH_SIZE = 16
EPOCHS = 4

# 加载预训练模型和分词器
tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
model = BertForSequenceClassification.from_pretrained(pretrained_model_name_or_path: 'bert-base-chinese', num_labels=3)

# 将模型移动到GPU
model = model.to(device)

# 创建数据加载器
train_dataset = SentimentDataset(train_texts, train_labels, tokenizer, MAX_LEN)
train_data_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE)

validation_dataset = SentimentDataset(test_texts, test_labels, tokenizer, MAX_LEN)
validation_data_loader = DataLoader(validation_dataset, batch_size=BATCH_SIZE)

```

```

# 训练模型
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)

for epoch in range(EPOCHS):
    model.train()
    total_train_loss = 0
    for batch in train_data_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        model.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_train_loss += loss.item()
        loss.backward()
        optimizer.step()

    avg_train_loss = total_train_loss / len(train_data_loader)
    print(f"Epoch {epoch + 1}/{EPOCHS}, Average Training Loss: {avg_train_loss:.4f}")

# 在验证集上评估模型
model.eval()
total_eval_loss = 0
for batch in validation_data_loader:
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    labels = batch['labels'].to(device)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        total_eval_loss += outputs.loss.item()

avg_val_loss = total_eval_loss / len(validation_data_loader)
print(f"Epoch {epoch + 1}/{EPOCHS}, Average Validation Loss: {avg_val_loss:.4f}")

# 当前日期和时间
current_time = datetime.now().strftime('%Y%m%d-%H%M%S')
model_save_path = f'F:/yuqing/Final/models/bert-sentiment-{current_time}'

# 保存模型和分词器
model.save_pretrained(model_save_path)
tokenizer.save_pretrained(model_save_path)

```

具体的完整代码，我已经上传到 [github](#) 上，参见链接

<https://github.com/cainiaopppppppp/NLP/tree/main>

3.6 基于 GPT-2 Output Detector trainer，训练一个中文模型

观察原来的 data 数据结构，可以发现都是 `**train/valid/test` 这样命名的。而前面的名称对应这是人为生成的文本还是 GPT 生成的文本。

因为我们缺乏 GPT 生成的中文数据集，而老师不要求模型的准确性，所以我们从平台上任意下载 6 个文件，并重命名为 `fake_data`、`real_data` 即可，如下图

```

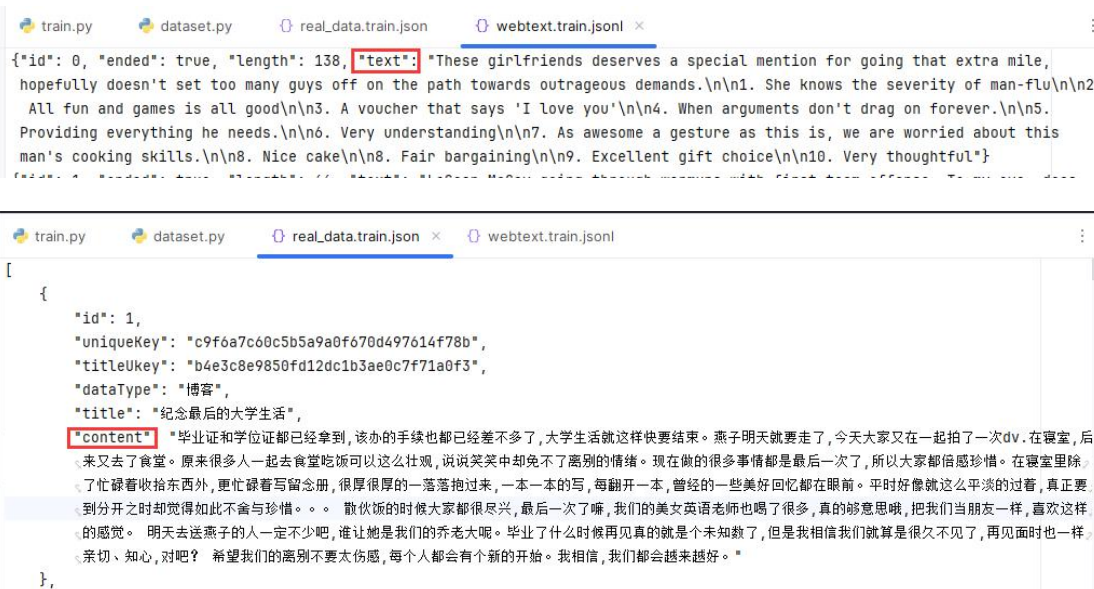
└─ data
  ├── fake_data.test.json
  ├── fake_data.train.json
  ├── fake_data.valid.json
  ├── real_data.test.json
  ├── real_data.train.json
  └── real_data.valid.json

```

我们需要修改的代码有 detector 目录下的 dataset.py、train.py、server.py

首先，修改 dataset.py 文件。

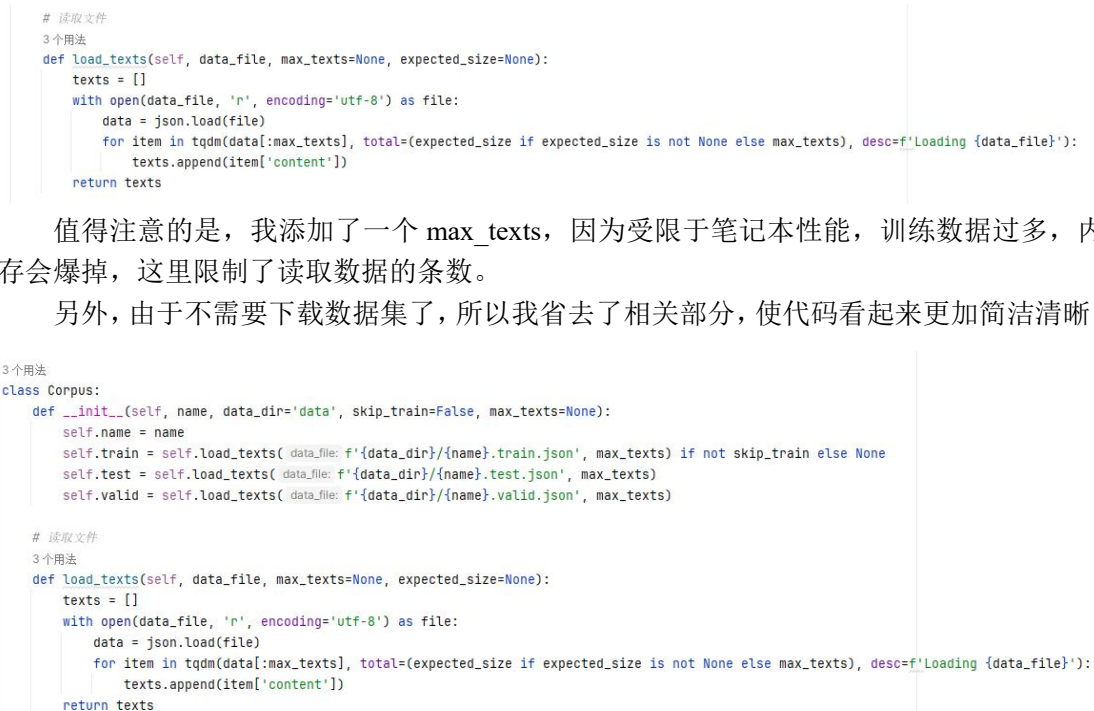
因为我们的中文数据与之前的英文数据结构不同，文件的后缀也不同



```
{
  "id": 0,
  "ended": true,
  "length": 138,
  "text": "These girlfriends deserves a special mention for going that extra mile, hopefully doesn't set too many guys off on the path towards outrageous demands.\n\n1. She knows the severity of man-flu\n\n2. All fun and games is all good\n\n3. A voucher that says 'I love you'\n\n4. When arguments don't drag on forever.\n\n5. Providing everything he needs.\n\n6. Very understanding\n\n7. As awesome a gesture as this is, we are worried about this man's cooking skills.\n\n8. Nice cake\n\n8. Fair bargaining\n\n9. Excellent gift choice\n\n10. Very thoughtful"}

[
  {
    "id": 1,
    "uniqueKey": "c9f6a7c60c5b5a9a0f670d497614f78b",
    "titleUKey": "b4e3c8e9850fd12dc1b3ae0c7f71a0f3",
    "dataType": "博客",
    "title": "纪念最后的大学生活",
    "content": "毕业证和学位证都已经拿到,该办的手续也都已经差不多了,大学生活就这样快要结束。燕子明天就要走了,今天大家又在一起拍了一次dv。在寝室,后来又去了食堂。原来很多人一起去食堂吃饭可以这么壮观,说说笑笑中却免不了离别的情绪。现在做的很多事情都是最后一次了,所以大家都倍感珍惜。在寝室除了忙碌着收拾东西外,更忙碌着写留念册,很厚很厚的一落笔抱过来,一本一本的写,每翻开一本,曾经的一些美好回忆都在眼前。平时好像就这么平淡的过着,真正要分开之时却觉得如此不舍与珍惜。。。 散伙饭的时候大家都很尽兴,最后一次了嘛,我们的美女英语老师也喝了很多,真的够意思哦,把我们当朋友一样,喜欢这样的感觉。 明天去送燕子的人一定不少吧,谁让她是我们的乔老大呢。毕业了什么时候再真的就是个未知数了,但是我相信我们就算是很久不见了,再见面时也一样。亲切、知心,对吧? 希望我们的离别不要太伤感,每个人都会有个新的开始。我相信,我们都会越来越好。"
  },
  ...
]
```

所以我们需要修改 load_texts，如下（我将 load_texts 直接放到 Corpus 下了）



```
# 读取文件
3个用法
def load_texts(self, data_file, max_texts=None, expected_size=None):
    texts = []
    with open(data_file, 'r', encoding='utf-8') as file:
        data = json.load(file)
        for item in tqdm(data[:max_texts], total=(expected_size if expected_size is not None else max_texts), desc=f'Loading {data_file}'):
            texts.append(item['content'])
    return texts

class Corpus:
    def __init__(self, name, data_dir='data', skip_train=False, max_texts=None):
        self.name = name
        self.train = self.load_texts( data_file: f'{data_dir}/{name}.train.json', max_texts) if not skip_train else None
        self.test = self.load_texts( data_file: f'{data_dir}/{name}.test.json', max_texts)
        self.valid = self.load_texts( data_file: f'{data_dir}/{name}.valid.json', max_texts)

# 读取文件
3个用法
def load_texts(self, data_file, max_texts=None, expected_size=None):
    texts = []
    with open(data_file, 'r', encoding='utf-8') as file:
        data = json.load(file)
        for item in tqdm(data[:max_texts], total=(expected_size if expected_size is not None else max_texts), desc=f'Loading {data_file}'):
            texts.append(item['content'])
    return texts
```


3个用法

```
class EncodedDataset(Dataset):
    def __init__(self, real_texts: List[str], fake_texts: List[str], tokenizer: BertTokenizer,
                 max_sequence_length: int = 512, epoch_size: int = None, token_dropout: float = None, seed: int = None):
        self.real_texts = real_texts
        self.fake_texts = fake_texts
        self.tokenizer = tokenizer
        self.max_sequence_length = max_sequence_length
        self.epoch_size = epoch_size
        self.token_dropout = token_dropout
        self.random = np.random.RandomState(seed)

    def __len__(self):
        return self.epoch_size or len(self.real_texts) + len(self.fake_texts)

    def __getitem__(self, index):
        label = self.random.randint(2) if self.epoch_size is not None else (1 if index < len(self.real_texts) else 0)
        text = self.real_texts[index] if label == 1 else self.fake_texts[index - len(self.real_texts)]

        tokens = self.tokenizer.encode(text, truncation=True, max_length=self.max_sequence_length, padding='max_length')
        mask = [float(token != self.tokenizer.pad_token_id) for token in tokens]

        return torch.tensor(tokens), torch.tensor(mask), label
```

接下来，修改 train.py

一样，省去下载文件的部分，另外，由于我们的数据集文件没有那么多，并且命名指出了哪个是 fake 哪个是 real，所以可以直接省去原有代码的繁琐部分，浓缩为下面两行代码

加载真实和生成的文本数据集

```
real_corpus = Corpus(args.real_dataset, args.data_dir, max_texts=50)
fake_corpus = Corpus(args.fake_dataset, args.data_dir, max_texts=50)
```

另外，因为我只有一个 gpu，所以我把 rank 啥的也省略了，所以修改后的代码如下

```
def train(model, device, train_loader, optimizer):
    model.train()
    total_loss, total_accuracy, total_count = 0, 0, 0
    for idx, (input_ids, attention_mask, labels) in enumerate(tqdm(train_loader)):
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        logits = outputs.logits

        preds = torch.argmax(logits, dim=-1)
        accuracy = (preds == labels).float().mean()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        total_accuracy += accuracy.item()
        total_count += 1

    avg_loss = total_loss / total_count
    avg_accuracy = total_accuracy / total_count
    return avg_loss, avg_accuracy
```

```

def validate(model, device, validation_loader):
    model.eval()
    total_loss, total_accuracy, total_count = 0, 0, 0
    with torch.no_grad():
        for idx, (input_ids, attention_mask, labels) in enumerate(tqdm(validation_loader)):
            input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

            outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
            loss = outputs.loss
            logits = outputs.logits

            preds = torch.argmax(logits, dim=-1)
            accuracy = (preds == labels).float().mean()

            total_loss += loss.item()
            total_accuracy += accuracy.item()
            total_count += 1

    avg_loss = total_loss / total_count
    avg_accuracy = total_accuracy / total_count
    return avg_loss, avg_accuracy

def save_model(model, optimizer, epoch, loss, accuracy, save_path):
    """ 保存模型和优化器的状态字典以及其他训练参数 """
    torch.save({
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'epoch': epoch,
        'loss': loss,
        'accuracy': accuracy
    }, save_path)

def main(args):
    # 加载分词器和模型
    tokenizer = BertTokenizer.from_pretrained(args.bert_model)
    model = BertForSequenceClassification.from_pretrained(args.bert_model)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(device)
    model.to(device)

    # 加载真实和生成的文本数据集
    real_corpus = Corpus(args.real_dataset, args.data_dir, max_texts=50)
    fake_corpus = Corpus(args.fake_dataset, args.data_dir, max_texts=50)

    # 创建训练数据集和验证数据集
    train_dataset = EncodedDataset(real_corpus.train, fake_corpus.train, tokenizer, max_sequence_length=512)
    train_loader = DataLoader(train_dataset, batch_size=args.batch_size, shuffle=True)

    validation_dataset = EncodedDataset(real_corpus.valid, fake_corpus.valid, tokenizer, max_sequence_length=512)
    validation_loader = DataLoader(validation_dataset, batch_size=args.batch_size, shuffle=False)

    # 设置优化器
    optimizer = Adam(model.parameters(), lr=args.learning_rate)
    best_validation_accuracy = 0

    # 进行训练和验证
    for epoch in range(args.epochs):
        train_loss, train_accuracy = train(model, device, train_loader, optimizer)
        print(f"Train Loss: {train_loss}, Train Accuracy: {train_accuracy}")

        validation_loss, validation_accuracy = validate(model, device, validation_loader)
        print(f"Validation Loss: {validation_loss}, Validation Accuracy: {validation_accuracy}")

        if validation_accuracy > best_validation_accuracy:
            best_validation_accuracy = validation_accuracy
            save_path = os.path.join('models', f'model_epoch-{epoch}_val_acc_{validation_accuracy:.4f}.pt')
            save_model(model, optimizer, epoch, validation_loss, validation_accuracy, save_path)
            print(f"Model saved to {save_path}")

```

注意，因为原来使用的 Roberta 词向量模型是处理英文的，所以我改用了 bert-base-chinese，用的是本地加载的方法。

```
(tensorflow) PS D:\Content_Secu\Task4\gpt-2-output-dataset-chinese> python -m detector.train
```

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at D:/Content_Secu/task4/bert-base-chinese 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.

```
cuda
Loading data/real_data.train.json: 100% |██████████████████████████████████████████████████| 50/50 [00:00<, ?it/s]
Loading data/real_data.test.json: 100% |██████████████████████████████████████████████████| 50/50 [00:00:00.00, 498135.87it/s]
Loading data/real_data.valid.json: 100% |██████████████████████████████████████████████████| 50/50 [00:00<, ?it/s]
Loading data/fake_data.train.json: 100% |██████████████████████████████████████████████████| 50/50 [00:00<, ?it/s]
Loading data/fake_data.test.json: 100% |██████████████████████████████████████████████████| 50/50 [00:00<, ?it/s]
Loading data/fake_data.valid.json: 100% |██████████████████████████████████████████████████| 50/50 [00:00<, ?it/s]
100% |██████████████████████████████████████████████████| 50/50 [01:51:00:00, 2.23s/it]
Train Loss: 0.4854792460799217, Train Accuracy: 0.79
100% |██████████████████████████████████████████████████| 50/50 [00:31:00:00, 1.61it/s]
Validation Loss: 0.21362854078412055, Validation Accuracy: 0.95
Model saved to models\model_epoch_0_val_acc_0.9500.pt
100% |██████████████████████████████████████████████████| 50/50 [01:34:00:00, 1.89s/it]
Train Loss: 0.22920498039233683, Train Accuracy: 0.93
100% |██████████████████████████████████████████████████| 50/50 [00:29:00:00, 1.72it/s]
Validation Loss: 0.057377680093050006, Validation Accuracy: 1.0
Model saved to models\model_epoch_1_val_acc_1.0000.pt
100% |██████████████████████████████████████████████████| 50/50 [01:36:00:00, 1.93s/it]
Train Loss: 0.10171952055767179, Train Accuracy: 0.97
100% |██████████████████████████████████████████████████| 50/50 [00:27:00:00, 1.81it/s]
Validation Loss: 0.020011453106999397, Validation Accuracy: 1.0
```

```

models
  model_epoch_0_val_acc_0.9500.pt
  model_epoch_1_val_acc_1.0000.pt

```

因为用的是 bert-base-chinese，所以一开始的声明要修改

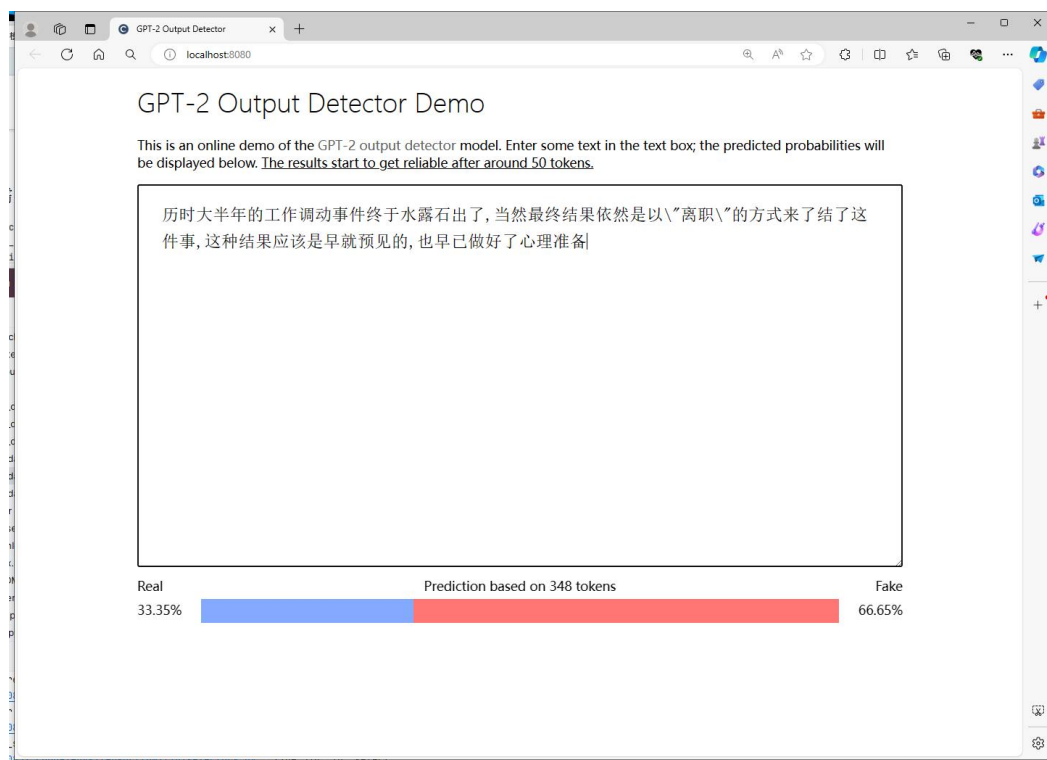
另外，我们只需要修改 `main` 函数部分即可，因为我们只需要修改加载模型的部分

```
def main(checkpoint, port=8080, device='cuda' if torch.cuda.is_available() else 'cpu'):  
    model_path = 'D:/Content_Secu/task4/bert-base-chinese'  
    tokenizer = BertTokenizer.from_pretrained(model_path)  
  
    # 明确地设置 BOS 和 EOS 令牌为 CLS 和 SEP 令牌  
    tokenizer.bos_token = tokenizer.cls_token  
    tokenizer.eos_token = tokenizer.sep_token  
  
    print(f'Loading checkpoint from {checkpoint}')  
    data = torch.load(checkpoint, map_location=device)  
  
    model = BertForSequenceClassification.from_pretrained(model_path)  
    model.load_state_dict(data['model_state_dict'], strict=False)  
    model.to(device)  
    model.eval()  
  
    print(f'Starting HTTP server on port {port}', file=sys.stderr)  
    server = HTTPServer((server_address, port), RequestHandler)  
    serve_forever(server, model, tokenizer, device)
```

最后，我们就可以运行代码了，`python -m detector.server models/我们的模型路径`

```
(tensorflow) PS D:\Content_Secu\task4\gpt-2-output-dataset-chinese> python -m detector.server models/model_epoch_1_val_acc_1.0000.pt
Loading checkpoint from models/model_epoch_1_val_acc_1.0000.pt
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at D:\Content_Secu\task4\bert-base-chinese 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.
Starting HTTP server on port 8080
[] Process has started; loading the model ...
[] Ready to serve at http://localhost:8080
```

尝试输入中文语句



值得注意的是，由于缺乏真正的由 GPT-2 生成的中文数据集，我们的效果是可以忽略不计，这里只是走了一遍流程。

局限性：

- 1.GPT-2 生成文本的中文数据集不足，无法训练。
- 2.内存受限，无法训练大量数据

四、出现的问题及解决方法

（详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法）

问题 1：运行 `gpt-2-output-dataset` 报错

终端 本地 × + ▾

```
File "C:\Users\35083\.conda\envs\tensorflow\lib\socketserver.py", line 747, in __init__
    self.handle()
File "C:\Users\35083\.conda\envs\tensorflow\lib\http\server.py", line 435, in handle
    self.handle_one_request()
File "C:\Users\35083\.conda\envs\tensorflow\lib\http\server.py", line 423, in handle_one_request
    method()
File "D:\Content_Secu\task4\gpt-2-output-dataset-master\detector\server.py", line 37, in do_GET
    tokens = tokens[:tokenizer.max_len - 2]
AttributeError: 'RobertaTokenizer' object has no attribute 'max_len'
-----
Traceback (most recent call last):
  File "C:\Users\35083\.conda\envs\tensorflow\lib\runpy.py", line 194, in _run_module_as_main
```

解决方法：参见 <https://github.com/openai/gpt-2-output-dataset/issues/35>



SachinGanesh commented on Jan 15, 2023

replace

```
self.model.load_state_dict(data['model_state_dict'])
```

with

```
self.model.load_state_dict(data['model_state_dict'], strict=False)
```



修改对应代码，如下

```
35     tokens = tokenizer.encode(query)
36     all_tokens = len(tokens)
37     tokens = tokens[:tokenizer.model_max_length - 2]
38     used_tokens = len(tokens)
39     tokens = torch.tensor([tokenizer.bos_token_id] + tokens + [tokenizer.eos_token_id]).unsqueeze(0)
40     mask = torch.ones_like(tokens)
```

问题 2：训练 GPT-2 Output Detector trainer 中文模型时遇到问题

```
File "C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\torch\nn\modules\node.py", line 1501, in _call_impl
    return forward_call(*args, **kwargs)
File "C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\torch\nn\modules\dropout.py", line 402, in forward
    self.outputs = self.self(
File "C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\torch\nn\modules\node.py", line 1501, in _call_impl
    return forward_call(*args, **kwargs)
File "C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\torch\nn\modules\dropout.py", line 334, in forward
    attention_probs = self.dropout(attention_probs)
File "C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\torch\nn\modules\node.py", line 1501, in _call_impl
    return forward_call(*args, **kwargs)
File "C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\torch\nn\modules\dropout.py", line 59, in forward
    return F.dropout(input, self.p, self.training, self.inplace)
File "C:\Users\35083\.conda\envs\tensorflow\lib\site-packages\torch\nn\functional.py", line 1252, in dropout
    return _VF.dropout(input, p, training) if inplace else _VF.dropout(input, p, training)
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 48.00 MiB (GPU 0; 4.00 GiB total capacity; 3.39 GiB already allocated; 0 bytes free; 3.44 GiB reserved in total by PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid fragmentation.  See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF (tensorflow) PS D:\Content_Secu\task4\gpt-2-output-dataset-chinese>
```

解决方法：限制数据读取条目

```
# 加载真实和生成的文本数据集
real_corpus = Corpus(args.real_dataset, args.data_dir, max_texts=50)
fake_corpus = Corpus(args.fake_dataset, args.data_dir, max_texts=50)
```

五、个人小结

(描述实验心得, 可提出实验的改进意见)

在这次实验中, 我深入探索了基于 Transformer 的语言模型 GPT-2, 并进行了文本生成与真实性检测的多项实践操作。这一过程不仅加深了我对自然语言处理当前技术的理解, 也锻炼了我解决实际问题 and 调优模型的能力。

学习理论知识是一回事, 将这些理论应用到实际中去又是另一回事。在实验中, 我亲手操作了复杂的 NLP 模型, 并尝试了各种技术来优化模型性能。通过亲自设定参数, 观察不同参数配置下模型的表现, 我对模型背后的数学原理和机制有了更深刻的理解。例如, 通过调整温度、Top-K 和 Top-P 参数, 我直观地看到了这些参数如何影响 GPT-2 生成文本的连贯性和多样性。

通过微调 BERT 模型, 我学习到了如何根据特定任务调整预训练模型。这一部分让我认识到了大规模预训练模型在特定领域应用的强大能力, 同时也让我意识到微调过程中需要精细操作的重要性。选择合适的学习率、批次大小和训练周期是实现良好训练效果的关键。

在实验过程中, 我遇到了多个技术问题, 例如模型加载失败、内存溢出等。这些问题迫使我必须深入到错误信息的背后逻辑, 逐一解决。通过查询文献、论坛和官方文档, 我学习到了很多实用的调试技巧和问题解决策略。

另外, 在文本真实性检测实验中, 我实现的中文模型表现并不好, 这可能是由于缺乏训练数据、训练数据过少。

六、教师评语及评分

教师签名:

年 月 日

