预训练模型的本地部署实践

目前还存在的一些问题：

- 大模型的部署--多卡GPU部署的方法需要研究
- 半精度模型的推理与全精度的差异，即：精度损失需要量化
- 大模型的推理加速方法需要进一步研究

1、CodeGen

前期的准备工作：

1. 预训练模型的下载
2. 推理所需的依赖关系的安装

   Huggingface Transformers库的本地安装
3. 测试推理所需的硬件资源
   - CPU推理：
     - RAM: 13.03 GB
     - Disk: 44.34 GB
   - GPU推理：
     - GPU内存消耗：12041 MB
     - RAM：13.47 GB
     - Disk： 44.34 GB

Transformers：

1. Install with pip （已验证内网可用）

   ```
   pip install transformers
   ```

2. Install from source

   ```
   pip install git+https://github.com/huggingface/transformers
   ```

3. 离线安装

   注意事项：需要先将旧版本的transformers库去除，然后进行安装操作

   准备条件：https://github.com/huggingface/transformers下载离线安装包

   解压后放到需要放到需要安装的镜像内，然后使用命令进行安装：

```
# will install the package
python setup.py install
```

内网部署的注意点：

由于直接安装dev版本的transformers库会确实tokenizer相关文件，故目前的内网安装教程为：先安装内网pip 源自备的transformers库，然后再重新安装transfromers-main dev版本的库，即可解决目前存在的问题。

查询相关的安装指令：

```
python setup.py --help
```

使用镜像文件创建用户环境：

- 镜像选择

  因为需要使用到GPU进行推理，故需要配置CUDA的镜像docker image进行部署，使用JupyterLab进行用户交互操作；

- 镜像文件的部署指令

  由于需要使用到docker Hub的指令进行镜像的拖取，需要用到内网的特殊指令：

  ```
  docker pull hub.2980.com/dockerhub/xxxx
  ```

  部署：

  ```
  # 示例代码，在外网机3080上启动容器
  docker run --gpus all -d -it -p 8891:8888 -v
  /mnt/gameai/workspace/nlp:/home/jovyan/work -e GRANT_SUDO=yes -e
  JUPYTER_ENABLE_LAB=yes --user root yangdafu/jupyter_torch_1-10_py37_cuda11-3:test
  ```

  关于挂载地址：把需要挂载的地址的路径替代启动命令中的$(pwd)即可。

  NOTE: 镜像的内部的映射地址默认为8888不要随意更改，**前面为宿主机端口，后面为容器内部端口(不要搞错了！)**

  默认密码为：gpu-jupyter

模型的本地部署及加载：

```
import torch
# GPU推理
torch.set_default_tensor_type(torch.cuda.FloatTensor)
from transformers import AutoTokenizer, AutoModelForCausalLM
tokenizer = AutoTokenizer.from_pretrained("./drive/MyDrive/NLP/PreTrainedModel/")
model = AutoModelForCausalLM.from_pretrained("./drive/MyDrive/NLP/PreTrainedModel/")
inputs = tokenizer("# this function prints hello world", return_tensors="pt").to(0)
sample = model.generate(**inputs, max_length=128)
print(tokenizer.decode(sample[0], truncate_before_pattern=[r"\n\n^#", "^'''",
"\n\n\n"]))
```

2、InCoder

Demo地址：https://huggingface.co/spaces/facebook/incoder-demo

重要信息记录：

- Demo of the 6.7B parameter vision of InCoder.
- A decoder-only Transformer model that can both extend and insert/filling code.

模型的运行条件：

```
pip install torch
pip install "tokenizers>=0.12.1"
pip install transformers
```

Model

这个6B模型有两个版本：全精度权重（float32，存储在分支main）和半精度权重（float16，存储在分支float16）。这些版本可以按以下方式加载:

- *Full-precision* (float32)

  如果你要对模型进行微调，应该使用这个方法（注意：这将需要大量的GPU内存，可能是多个GPU，而且我们还没有尝试在transformers中训练模型---它是在Fairseq中训练的）。

  ```
  model = AutoModelForCausalLM.from_pretrained("facebook/incoder-6B")
  ```

- *Half-precision* (float16)

  如果你只做推理（即从模型生成），可以使用这个方法。它将使用更少的GPU内存，以及在加载模型时使用更少的RAM。在这个版本中，它应该能够在16GB的GPU上进行推理（批量大小为1，序列长度至少为256）。

  ```
  model = AutoModelForCausalLM.from_pretrained("facebook/incoder-6B",
  revision="float16", torch_dtype=torch.float16, low_cpu_mem_usage=True)
  ```

Tokenizer

```
tokenizer = AutoTokenizer.from_pretrained("facebook/incoder-6B")
```

Note: incoder-1B和incoder-6B标记器是相同的，所以也可以使用'facebook/incoder-1B'。

当调用tokenizer.decode时，重要的是传递clean_up_tokenization_spaces=False以避免删除标点符号后的空格：

```
tokenizer.decode(tokenizer.encode("from ."), clean_up_tokenization_spaces=False)
```

Note-1 :

编码时预留了<|endoftext|>标记，因为对于我们的模型来说，这标志着一个文档的开始。这个标记可以通过向tokenizer.decode传递skip_special_tokens=True来从解码后的输出中移除。

NOTE-2 :

使用"low_cpu_mem_usage=True"时，需要安装加速库：pip install accelerate

部署半精度float16模型的硬件资源消耗情况统计：

- 模型体积：　14.4 GB
- RAM：　2.53 GB
- GPU：　14343 MB
- Disk：　51.43 GB

部署验证代码：

```
%pip install git+https://github.com/huggingface/transformers.git
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

model = AutoModelForCausalLM.from_pretrained("facebook/incoder-6B", revision="float16",
torch_dtype=torch.float16, low_cpu_mem_usage=True)
model = model.half().to("cuda:0")
tokenizer = AutoTokenizer.from_pretrained("facebook/incoder-6B")


PAD = "<pad>"
tokenizer.pad_token = PAD
# tokenizer.padding_side = "left"

doc1 = "def count_words(filename):\n"
doc2 = 'def count_words(filename):\n    """Count the words in the file"""'

dct = tokenizer(doc1, padding="longest", truncation=True,
return_tensors="pt").to('cuda')
```

```
hypotheses_batch = model.generate(**dct, do_sample=True, temperature=0.2, top_p=0.95,
max_length=256)
tokenizer.decode(hypotheses_batch[0], clean_up_tokenization_spaces=False,
skip_special_tokens=True)
```

部署全精度模型

- 结论：在Colab Pro+上测试失败
  - RAM：51+ GB，未测试完成
  - 模型体积：24.8 GB

使用Flask进行NLP模型的本地部署的实践

参考资料：

- Flask官方文档：Quickstart
- Extending Flask-RESTful
- Example:
  - GitHub: pythondataanalysis
  - Medium: Docker Made Easy for Data Scientists
  -
- FastAPI
- 测试Flask遇到的问题
  - Using curl.GET request with multiple query parameters
  - Flask only sees first parameter from multiple parameters sent with curl
  - Send JSON-Request to Flask via Curl

预备条件：

```
pip install flask
pip install transformers
pip install flask_restful
```

Flask的模型部署脚本：

```
from flask import Flask, request
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)
```

```
# 模型加载
import torch
import time

# GPU加速推理
torch.set_default_tensor_type(torch.cuda.FloatTensor)
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained('./PreTrained/')
model = AutoModelForCausalLM.from_pretrained('./PreTrained/')

class CodeComplete(Resource):
    def put(self):
        codehints = request.form['data']
        print('CodeHints: ', codehints)
        inputs = tokenizer(codehints, return_tensors='pt').to(0)
        sample = model.generate(**inputs, max_length=128)
        prediction = tokenizer.decode(sample[0], truncate_before_pattern=[r"\n\n^#",
"^'''", "\n\n\n"])
        print(prediction)
        return prediction

api.add_resource(CodeComplete, '/')

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

参考地址：

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

使用 curl 进行测试：

```
curl http://127.0.0.1:5000 'data=# this function prints hello world'  -X  PUT
```

存在的问题：

无法正常解析host， 无法进行正常的交互操作。

修正之前的代码部署问题：

参考资料：

- How to get POSTed JOSN in Flask ?
- How to post JSON data using Curl ?

```
@app.route('/', method=['GET', 'POST'])
def CodeComplete():
    if request.method == 'POST':
        content = request.json
        codehints = content['data']
        inputs = tokenizer(codehints, return_tensors="pt")
        sample = model.generate(**inputs, max_length=128)
        prediction = tokenizer.decode(sample[0], truncate_before_pattern=[r"\n\n^#",
"^'''", "\n\n\n"])
        return prediction

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

```
from flask import Flask
from flask import request
import json

app = Flask(__name__)

@app.route("/",  methods = ['POST'])
def hello():
    print(request.get_json())
    return json.dumps({'success':True}), 200, {'ContentType':'application/json'}

if __name__ == "__main__":
    app.run()
```

访问服务：

```
curl -X POST http://10.17.68.105:5000 -H 'Content-Type: application/json' -d
'{"data":"#print hello world function"}'
```

预训练模型的重复加载问题：

- Why does running the Flask dev server running itself twice ?

  原因分析：

  The Werkzeug reloader spawns a child process so that it can restart that process each time your code changes. Werkzeug is the library that supplies Flask with the development server when you call `app.run()`.

  解决方法：

  If you set `use_reloader` to `False` you'll see the behaviour go away, but the you also lose the reloading functionlity.

  ```
  app.run(port=5000, debug=True, host='0.0.0.0', use_reloader=False)
  ```

后端对接细节

请求URL：

- http://10.17.68.105:5000

请求方式：

- POST

请求参数：

参数名	类型	默认值	说明
data	String	None	CodeHints, 代码上文语境

返回参数：

- 随意，确认成功接收即可。

注意事项：

1. 关于 flask 的错误：ImportError: cannot import name 'Flask'

   原因定位：文件名不能为flask，重命名即可。

2. 安装curl的报错："curl: Depends: libcurl4(=7.61.0-1ubuntu2.2) but it is not going to be installed on Ubuntu 18.10...."

   ```
   sudu apt remove libcurl4
   sudo apt install curl
   ```

3. 关于内网的transformers库的安装问题

   目前的解决方案：

   1. 先安装内置pip源的transformers库
   2. 然后再重装dev版本的transformers库

4. AssortionError: Unimplemented methods "GET", API Flask

   原因分析：This means your class needs a "GET" method. So instead of calling that method "getMemory" rename to "get".