模型推理服务化框架Triton保姆式教程（三）：开发实践

吃果冻不吐果冻皮

前面给大家分享了模型推理+服务化框架Triton保姆式教程系列的**快速入门**和 **架构解析**，本文进行Triton开发实践的分享。

环境准备

准备镜像及依赖库安装

首先，拉取推理系统的服务端镜像(`tritonserver`)。

```
# tritonserver服务端镜像
# nvcr.io/nvidia/tritonserver:<yy.mm>-py3
docker pull nvcr.io/nvidia/tritonserver:23.04-py3
```

然后，拉取Pytorch官方镜像作为推理系统的客户端同时进行一些预处理操作（当然也可以直接拉取tritonserver客户端SDK镜像）。

```
docker pull pytorch/pytorch:2.0.0-cuda11.7-cudnn8-devel

# tritonserver客户端SDK镜像
# nvcr.io/nvidia/tritonserver:<yy.mm>-py3-sdk
# docker pull nvcr.io/nvidia/tritonserver:23.04-py3-sdk
```

接下来，基于官方Pytorch镜像创建一个容器客户端。

```
docker run -dt --name pytorch200_cu117_dev --restart=always --gpus all \
--network=host \
--shm-size 4G \
-v /home/gdong/workspace:/workspace \
-w /workspace \
pytorch/pytorch:2.0.0-cuda11.7-cudnn8-devel \
/bin/bash
```

进入容器。

```
docker exec -it pytorch200_cu117_dev bash
```

安装datasets、transformer以及tritonclient库。

```
pip install datasets transformers -i https://pypi.tuna.tsinghua.edu.cn/simple  --trust

pip install tritonclient[all] -i https://pypi.tuna.tsinghua.edu.cn/simple  --trusted-h
```

准备模型

本文将基于 PyTorch 后端使用 resnet50 模型来进行图片分类，因此，需预先下载 resnet50 模型，然后将其转换为torchscript格式。具体代码（ `resnet50_convert_torchscript.py` ）如下所示：

知乎 首发于 AI工程（MLOps）

```
resnet50 = models.resnet50(pretrained=True)
resnet50.eval()
image = torch.randn(1, 3, 244, 244)
resnet50_traced = torch.jit.trace(resnet50, image)
resnet50(image)
# resnet50_traced.save('/workspace/model/resnet50/model.pt')
torch.jit.save(resnet50_traced, "/workspace/model/resnet50/model.pt")
```

最后，拉取Triton Server 代码库。

```
git clone -b r23.04 https://github.com/triton-inference-server/server.git
```

一些常见后端backend的配置都在 `server/docs/examples` 目录下。

```
tree docs/examples -L 2
docs/examples
|-- README.md
|-- fetch_models.sh
|-- jetson
|   |-- README.md
|   `-- concurrency_and_dynamic_batching
`-- model_repository
    |-- densenet_onnx
    |-- inception_graphdef
    |-- simple
    |-- simple_dyna_sequence
    |-- simple_identity
    |-- simple_int8
    |-- simple_sequence
    `-- simple_string

11 directories, 3 files
```

拉取Triton Tutorials库，该仓库中包含Triton的教程和样例，本文使用 `Quick_Deploy/PyTorch` 下部署一个Pytorch模型进行讲解。

```
git clone https://github.com/triton-inference-server/tutorials.git
```

ok，目前为止，前期的准备工作已经好了，下面进行具体的开发实践。

开发实践

创建一个模型仓库

首先，构建一个模型仓库，仓库的目录结构+如下所示：

```
model_repository/
`-- resnet50
    |-- 1
    |   `-- model.pt
    `-- config.pbtxt
```

其中， `config.pbtxt` 是模型配置文件； `1` 表示模型版本号； `resnet50` 表示模型名，需要与 `config.pbtxt` 文件中的 `name` 字段保存一致； `model.pt` 为模型权重+（即上面转换后的模型权重）。

`config.pbtxt` 具体内容如下所示：

```
max_batch_size : 0
input [
  {
    name: "input__0"
    data_type: TYPE_FP32
    dims: [ 3, 224, 224 ]
    reshape { shape: [ 1, 3, 224, 224 ] }
  }
]
output [
  {
    name: "output__0"
    data_type: TYPE_FP32
    dims: [ 1, 1000 ,1, 1]
    reshape { shape: [ 1, 1000 ] }
  }
]
```

重要字段说明如下：

- name：模型名
- platform：用于指定模型对应的后端（backend），比如：pytorch_libtorch、onnxruntime_onnx、tensorrt_plan等
- max_batch_size：模型推理在batch模式下支持的最大batch数
- input：模型输入属性配置。
- output：模型输出属性配置。

模型仓库构建好之后，接下来启动Triton推理服务端。

启动推理服务

启动推理服务启动服务的方法有两种：一种是用 docker 启动并执行命令，一种是进入 docker 中然后手动调用命令。

本文直接使用 Docker 启动并执行命令，命令如下所示：

```
docker run --gpus all --rm \
-p 8000:8000 \
-p 8001:8001 \
-p 8002:8002 \
-v ${PWD}/model_repository:/models \
nvcr.io/nvidia/tritonserver:23.04-py3 \
tritonserver --model-repository=/models
```

参数说明：

- -p：宿主机与容器内端口映射
- -v：将宿主机存储挂载进容器，这里将模型仓库挂载进容器
- --model-repository：指定Triton服务模型仓库的地址

启动过程：

```
docker run --gpus all --rm \
> -p 8000:8000 \
> -p 8001:8001 \
> -p 8002:8002 \
> -v ${PWD}/model_repository:/models \
> nvcr.io/nvidia/tritonserver:23.04-py3 \
> tritonserver --model-repository=/models

==========================
```

```
NVIDIA Release 23.04 (build 58408265)
Triton Server Version 2.33.0

Copyright (c) 2018-2023, NVIDIA CORPORATION & AFFILIATES.  All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION & AFFILIATES.  All rights r

This container image and its contents are governed by the NVIDIA Deep Learning Contain
By pulling and using the container, you accept the terms and conditions of this licens
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

NOTE: CUDA Forward Compatibility mode ENABLED.
  Using CUDA 12.1 driver version 530.30.02 with kernel driver version 525.105.17.
  See https://docs.nvidia.com/deploy/cuda-compatibility/ for details.

I0529 16:17:07.342510 1 pinned_memory_manager.cc:240] Pinned memory pool is created at
I0529 16:17:07.356934 1 cuda_memory_manager.cc:105] CUDA memory pool is created on dev
I0529 16:17:07.356947 1 cuda_memory_manager.cc:105] CUDA memory pool is created on dev
I0529 16:17:07.356951 1 cuda_memory_manager.cc:105] CUDA memory pool is created on dev
I0529 16:17:07.356956 1 cuda_memory_manager.cc:105] CUDA memory pool is created on dev
I0529 16:17:09.509579 1 model_lifecycle.cc:459] loading: resnet50:1
I0529 16:17:48.920259 1 libtorch.cc:2008] TRITONBACKEND_Initialize: pytorch
I0529 16:17:48.920338 1 libtorch.cc:2018] Triton TRITONBACKEND API version: 1.12
I0529 16:17:48.920368 1 libtorch.cc:2024] 'pytorch' TRITONBACKEND API version: 1.12
I0529 16:17:50.186661 1 libtorch.cc:2057] TRITONBACKEND_ModelInitialize: resnet50 (ver
W0529 16:17:50.188218 1 libtorch.cc:284] skipping model configuration auto-complete fo
I0529 16:17:50.188786 1 libtorch.cc:313] Optimized execution is enabled for model inst
I0529 16:17:50.188800 1 libtorch.cc:332] Cache Cleaning is disabled for model instance
I0529 16:17:50.188806 1 libtorch.cc:349] Inference Mode is disabled for model instance
I0529 16:17:50.188812 1 libtorch.cc:443] NvFuser is not specified for model instance '
I0529 16:17:50.188965 1 libtorch.cc:2101] TRITONBACKEND_ModelInstanceInitialize: resne
I0529 16:17:51.451587 1 libtorch.cc:2101] TRITONBACKEND_ModelInstanceInitialize: resne
I0529 16:17:53.696469 1 libtorch.cc:2101] TRITONBACKEND_ModelInstanceInitialize: resne
I0529 16:17:55.208326 1 libtorch.cc:2101] TRITONBACKEND_ModelInstanceInitialize: resne
I0529 16:17:57.150685 1 model_lifecycle.cc:694] successfully loaded 'resnet50' version
I0529 16:17:57.150981 1 server.cc:583]
+------------------+------+
| Repository Agent | Path |
+------------------+------+
+------------------+------+

I0529 16:17:57.151158 1 server.cc:610]
+---------+-------------------------------------------------------------+-----------------
| Backend | Path                                                        | Config
+---------+-------------------------------------------------------------+-----------------
| pytorch | /opt/tritonserver/backends/pytorch/libtriton_pytorch.so | {"cmdline":{"aut
+---------+-------------------------------------------------------------+-----------------

I0529 16:17:57.151242 1 server.cc:653]
+----------+---------+--------+
| Model    | Version | Status |
+----------+---------+--------+
| resnet50 | 1       | READY  |
+----------+---------+--------+

I0529 16:17:57.279760 1 metrics.cc:808] Collecting metrics for GPU 0: NVIDIA A800 80(
I0529 16:17:57.279796 1 metrics.cc:808] Collecting metrics for GPU 1: NVIDIA A800 80GB
I0529 16:17:57.279805 1 metrics.cc:808] Collecting metrics for GPU 2: NVIDIA A800 80GB
I0529 16:17:57.279812 1 metrics.cc:808] Collecting metrics for GPU 3: NVIDIA A800 80GB
I0529 16:17:57.280398 1 metrics.cc:701] Collecting CPU metrics
I0529 16:17:57.280604 1 tritonserver.cc:2387]
+----------------------------------+------------------------------------------------
| Option                           | Value
+----------------------------------+------------------------------------------------
| server_id                        | triton
```

```
| model_repository_path[0]         | /models
| model_control_mode               | MODE_NONE
| strict_model_config              | 0
| rate_limit                       | OFF
| pinned_memory_pool_byte_size     | 268435456
| cuda_memory_pool_byte_size{0}    | 67108864
| cuda_memory_pool_byte_size{1}    | 67108864
| cuda_memory_pool_byte_size{2}    | 67108864
| cuda_memory_pool_byte_size{3}    | 67108864
| min_supported_compute_capability | 6.0
| strict_readiness                 | 1
| exit_timeout                     | 30
| cache_enabled                    | 0
+----------------------------------+------------------------------------------------

I0529 16:17:57.282917 1 grpc_server.cc:2450] Started GRPCInferenceService at 0.0.0.0:8
I0529 16:17:57.283185 1 http_server.cc:3555] Started HTTPService at 0.0.0.0:8000
I0529 16:17:57.325114 1 http_server.cc:185] Started Metrics Service at 0.0.0.0:8002
```

启动完成之后，我们会在日志中看到模型处于READY状态。

默认情况下，triton 会在每张 GPU 卡上面启动一个实例。

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     12054      C   tritonserver                     1658MiB |
|    1   N/A  N/A     12054      C   tritonserver                     1830MiB |
|    2   N/A  N/A     12054      C   tritonserver                     1830MiB |
|    3   N/A  N/A     12054      C   tritonserver                     1658MiB |
+-----------------------------------------------------------------------------+
```

接下来，进入推理系统客户端，发送推理请求。

发送推理请求

首先，创建客户端脚本 `client.py`：

```
import numpy as np
from torchvision import transforms
from PIL import Image
import tritonclient.http as httpclient
from tritonclient.utils import triton_to_np_dtype

# 图片预处理
# preprocessing function
def rn50_preprocess(img_path="img1.jpg"):
    img = Image.open(img_path)
    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    return preprocess(img).numpy()

transformed_img = rn50_preprocess()

# 设置连接到Triton服务端
# Setting up client
```

知乎 首发于 AI工程（MLOps）

```
# 指定resnet50模型的输入和输出
inputs = httpclient.InferInput("input__0", transformed_img.shape, datatype="FP32")
inputs.set_data_from_numpy(transformed_img, binary_data=True)

# class_count表示获取 TopK 分类预测结果。如果没有设置这个选项，默认值为0，那么将会得到一个 100
outputs = httpclient.InferRequestedOutput("output__0", binary_data=True, class_count=1

# 发送一个推理请求到Triton服务端
# Querying the server
results = client.infer(model_name="resnet50", inputs=[inputs], outputs=[outputs])
inference_output = results.as_numpy('output__0')
print(inference_output[:5])
```

预先下载好，用于推理请求的图片：

```
wget  -O img1.jpg "https://www.hakaimagazine.com/wp-content/uploads/header-gulf-birds.
```

执行客户端脚本发送请求：

```
> python client.py

[b'12.474869:90' b'11.527128:92' b'9.659309:14' b'8.408504:136'
 b'8.216769:11']
```

输出的格式为 `<confidence_score>:<classification_index>` 。

如果模型具有与每个分类索引关联的标签，Triton 也会返回这些标签，具体格式如下所示。

```
<confidence_score>:<classification_index>:<classification_index_label>
```

那么config.pbtxt文件中，需要添加label_filename配置，具体如下所示：

```
output [
  {
    name: "output__0"
    data_type: TYPE_FP32
    dims: [ 1, 1000 ,1, 1]
    reshape+ { shape: [ 1, 1000 ] }
    label_filename: "labels.txt"
  }
]
```

推理请求结果如下所示：

```
> python client.py
[b'12.474869:90:LORIKEET' b'11.527128:92:BEE EATER'
 b'9.659309:14:INDIGO FINCH']
(3,)
```

结语

本文演示了如何使用Triton封装一个模型推理服务，相关代码放置在**Github**，希望能够给你带来帮助。

参考文档：

- **Triton User Guide**

知乎 首发于 AI工程（MLOps）

编辑于 2023-06-10 08:40 · IP 属地四川

大模型推理 大模型 人工智能

理性发言，友善互动

1 条评论 默认 最新

咿呀咿呀

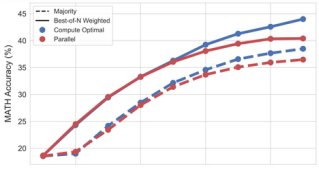
如果想要多节点部署，有没有一些参考。题主尝试过吗

2023-06-27 · 上海 回复 喜欢

文章被以下专栏收录

AI工程（MLOps）
AI System/MLOps

推荐阅读



小模型越级挑战14倍参数大模型，谷歌开启Test-Time端新...

量子位 发表于量子位

Stacking 模型融合+集成模型

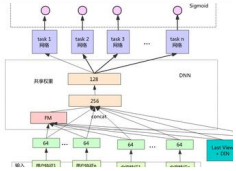
Stacking 概念stacking 是一种模型融合算法，基本思路是通过一个模型融合若干单模型的预测结果，目的是降低单模型的泛化误差。Stacking 原理单模型——可以称作一级模型 Stacking——可以...

爱吃牛油果的璐璐

无监督主题模型

作者|Zolzaya Luvsandorj 编译|VK 来源|Towards Data Science 原文链接：https://towardsdatascience.com/i to-nlp-part-5a-unsupervised-topic-model-in-python-...

灰灰 发表于磐创AI



聊聊知乎的推荐页排序模目标模型

养生的控制... 发表于建模