TensorRT&Triton学习笔记(一)：triton和模型部署+client

Keep Learning

前言

先介绍TensorRT、Triton的关系和区别：

TensorRT：为inference（推理）为生，是NVIDIA研发的一款针对深度学习模型在GPU上的计算，显著提高GPU上的模型推理性能。即一种专门针对高性能推理的模型框架，也可以解析其他框架的模型如tensorflow、torch。

主要优化手段如下：

Triton：类似于TensorFlow Serving，但triton包括server和client。

triton serving能够实现不同模型的统一部署和服务，提供http和grpc协议，给triton client请求模型推理。

----------分割线----------

如果是要将模型和推理嵌入在服务或软硬件中，那么TensorRT是很好的选择，使用它来加载模型进行推理，提升性能（tensorrt runtime）；

不然，常规的做法是模型推理和其他业务隔离，模型统一部署在triton server，然后其他业务通过triton client来进行模型推理的请求。

声明

这篇文章的主题会先主要介绍Triton的入门内容，TensorRT的内容后续会持续更新。

（代码较多，可以根据目录选择感兴趣的内容观看）

实验环境：Ubuntu18.04， GeForce RTX 2080Ti

Triton部署

安装

通过docker的形式，首先拉取镜像

```
# <xx.yy>为Triton的版本
docker pull nvcr.io/nvidia/tritonserver:<xx.yy>-py3

# 例如，拉取 20.12
docker pull nvcr.io/nvidia/tritonserver:20.12-py3
```

例如，20.12的版本需要NVIDIA Driver需要455以上，支持TensorRT 7.2.2。TensorRT版本要对应，不然模型可能会无法部署。

其他版本信息可以前往官网查看：docs.nvidia.com/deeplea...

启动

CPU版本的启动

```
docker run --rm -p8000:8000 -p8001:8001 -p8002:8002 -v/full/path/to/docs/examples/mode
```

GPU版本的启动，使用1个gpu

```
docker run --gpus=1 --rm -p8000:8000 -p8001:8001 -p8002:8002 -v/full/path/to/docs/exam
```

1. /full/path/to/docs/examples/model_repository：模型仓库*的路径。除了本地文件系统，还支持Google Cloud、S3、Azure这些云存储：github.com/triton-infer...
2. --rm：表示容器停止运行时会删除容器
3. 8000为http端口，8001为grpc端口

正常启动的话，可以看到部署的模型运行状态，以及对外提供的服务端口

```
+----------------------+---------+--------+
| Model                | Version | Status |
+----------------------+---------+--------+
| simple               | 1       | READY  |
| simple_dyna_sequence | 1       | READY  |
| simple_identity      | 1       | READY  |
| simple_int8          | 1       | READY  |
| simple_sequence      | 1       | READY  |
| simple_string        | 1       | READY  |
| tf_graphdef          | 1       | READY  |
| tf_onnx              | 1       | READY  |
| tf_savemodel         | 1       | READY  |
| torch_model          | 1       | READY  |
| torch_onnx           | 1       | READY  |
+----------------------+---------+--------+
```



```
I0310 13:23:53.765879 1 grpc_server.cc:4195] Started GRPCInferenceService at 0.0.0.0:8001
I0310 13:23:53.766505 1 http_server.cc:2857] Started HTTPService at 0.0.0.0:8000
I0310 13:23:53.816034 1 http_server.cc:167] Started Metrics Service at 0.0.0.0:8002
```

验证服务

也可以通过以下命令来验证服务是否正常运行

```
curl -v localhost:8000/v2/health/ready
```

知乎 首发于 TensorFlow及其他深度学习框架

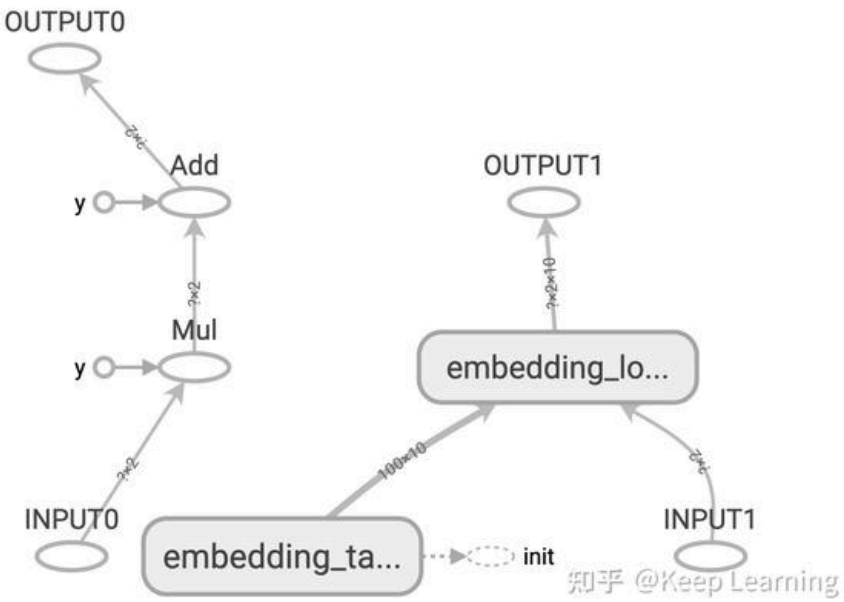
```
* Connected to localhost (127.0.0.1) port 8000 (#0)
> GET /v2/health/ready HTTP/1.1
> Host: localhost:8000
> User-Agent: curl/7.61.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 0
< Content-Type: text/plain
<
* Connection #0 to host localhost left intact
```


模型生成

Triton支持以下模型：TensorRT、ONNX、TensorFlow、Torch、OpenVINO、DALI，还有Python backend自定义生成的Python模型。

我们以一个简单的模型结构来演示：

1. INPUT0节点通过四则运算✦得到OUTPUT0节点；
2. INPUT1节点通过embedding table映射为OUTPUT1。



完整代码参考：github.com/QunBB/DeepLe...

TensorFlow

tensorflow可以生成SavedModel或者GraphDef的模型格式

SavedModel模型需要按照以下的目录结构✦进行存储：

```
<model-repository-path>/
    <model-name>/
      config.pbtxt
      1/
        model.savedmodel/
           <saved-model files>
```

GraphDef：

```
    config.pbtxt
    1/
      model.graphdef
```

```python
import os
import tensorflow as tf
from tensorflow.python.framework import graph_io


def create_modelfile(model_version_dir, max_batch,
                     save_type="graphdef",
                     version_policy=None):
    # your model net
    input0_shape = [None, 2]
    input1_shape = [None, 2]
    x1 = tf.placeholder(tf.float32, input0_shape, name='INPUT0')
    inputs_id = tf.placeholder(tf.int32, input1_shape, name='INPUT1')

    out = tf.add(tf.multiply(x1, 0.5), 2)

    embedding = tf.get_variable("embedding_table", shape=[100, 10])
    pre = tf.nn.embedding_lookup(embedding, inputs_id)

    out0 = tf.identity(out, "OUTPUT0")
    out1 = tf.identity(pre, "OUTPUT1")

    try:
        os.makedirs(model_version_dir)
    except OSError as ex:
        pass  # ignore existing dir

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        if save_type == 'graphdef':
            create_graphdef_modelfile(model_version_dir, sess,
                                      outputs=["OUTPUT0", "OUTPUT1"])
        elif save_type == 'savemodel':
            create_savedmodel_modelfile(model_version_dir,
                                        sess,
                                        inputs={
                                            "INPUT0": x1,
                                            "INPUT1": inputs_id
                                        },
                                        outputs={
                                            "OUTPUT0": out,
                                            "OUTPUT1": pre
                                        })
        else:
            raise ValueError("save_type must be one of ['tensorflow_graphdef', 'tensor

    create_modelconfig(models_dir=os.path.dirname(model_version_dir),
                       max_batch=max_batch,
                       save_type=save_type,
                       version_policy=version_policy)


def create_graphdef_modelfile(model_version_dir, sess, outputs):
    """
    tensorflow graphdef只能保存constant，无法保存Variable
    可以借助tf.graph_util.convert_variables_to_constants将Variable转化为constant
    :param model_version_dir:
    :param sess:
    :return:
```

```
    new_graph = tf.graph_util.convert_variables_to_constants(sess=sess,
                                                              input_graph_def=graph,
                                                              output_node_names=outputs
    graph_io.write_graph(new_graph,
                         model_version_dir,
                         "model.graphdef",
                         as_text=False)


def create_savedmodel_modelfile(model_version_dir, sess, inputs, outputs):
    """

    :param model_version_dir:
    :param sess:
    :param inputs: dict, {input_name: input_tensor}
    :param outputs: dict, {output_name: output_tensor}
    :return:
    """
    tf.saved_model.simple_save(sess,
                               model_version_dir + "/model.savedmodel",
                               inputs=inputs,
                               outputs=outputs)
```

torch

pytorch模型的目录结构格式：

```
<model-repository-path>/
    <model-name>/
      config.pbtxt
      1/
        model.pt
```

```
import os
import torch
from torch import nn


class MyNet(nn.Module):

    def __init__(self):
        super(MyNet, self).__init__()

        self.embedding = nn.Embedding(num_embeddings=100,
                                      embedding_dim=10)

    def forward(self, input0, input1):
        # tf.add(tf.multiply(x1, 0.5), 2)
        output0 = torch.add(torch.multiply(input0, 0.5), 2)

        output1 = self.embedding(input1)

        return output0, output1


def create_modelfile(model_version_dir, max_batch,
                     version_policy=None):
    # your model net

    # 定义输入的格式
    example_input0 = torch.zeros([2], dtype=torch.float32)
    example_input1 = torch.zeros([2], dtype=torch.int32)
```

```
    traced = torch.jit.trace(my_model, (example_input0, example_input1))

    try:
        os.makedirs(model_version_dir)
    except OSError as ex:
        pass  # ignore existing dir

    traced.save(model_version_dir + "/model.pt")
```

ONNX

ONNX的目录结构：

```
<model-repository-path>/
    <model-name>/
      config.pbtxt
      1/
        model.onnx
```

ONNX提供一种开源的深度学习和传统的机器学习模型格式，目的在于模型在不同框架之间进行转移。

下面我们介绍最常用的tensorflow和torch模型转成ONNX的方法。

tensorflow模型 --> ONNX

```
pip install -U tf2onnx+

# savedmodel
python -m tf2onnx.convert --saved-model tensorflow-model-path --output model.onnx

# checkpoint
python -m tf2onnx.convert --checkpoint tensorflow-model-meta-file-path --output model.

# graphdef
python -m tf2onnx.convert --graphdef tensorflow-model-graphdef-file --output model.onn
```

torch --> ONNX

```
import os
import torch
import torch.onnx


def torch2onnx(model_version_dir, max_batch):
    # 定义输入的格式
    example_input0 = torch.zeros([max_batch, 2], dtype=torch.float32)
    example_input1 = torch.zeros([max_batch, 2], dtype=torch.int32)

    my_model = MyNet()

    try:
        os.makedirs(model_version_dir)
    except OSError as ex:
        pass  # ignore existing dir

    torch.onnx.export(my_model,
                      (example_input0, example_input1),
                      os.path.join(model_version_dir, 'model.onnx'),
                      # 输入节点的名称
```

```
                output_names=("OUTPUT0", "OUTPUT1"),
                # 设置batch_size的维度
                dynamic_axes={"INPUT0": [0], "INPUT1": [0], "OUTPUT0": [0], "OUT
                verbose=True)
```

TensorRT

需要注意：TensorRT仅支持GPU。

```
<model-repository-path>/
    <model-name>/
      config.pbtxt
      1/
        model.plan
```

比较推荐的方式是从ONNX解析得到TensorRT模型（TensorRT）

```
import tensorrt as trt
import os


def onnx2trt(model_version_dir, onnx_model_file, max_batch):
    logger = trt.Logger(trt.Logger.WARNING)

    builder = trt.Builder(logger)

    # The EXPLICIT_BATCH flag is required in order to import models using the ONNX par
    network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLIC

    parser = trt.OnnxParser(network, logger)

    success = parser.parse_from_file(onnx_model_file)
    for idx in range(parser.num_errors):
        print(parser.get_error(idx))

    if not success:
        pass  # Error handling code here

    profile = builder.create_optimization_profile()
    # INPUT0可以接收[1, 2] -> [max_batch, 2]的维度
    profile.set_shape("INPUT0", [1, 2], [1, 2], [max_batch, 2])
    profile.set_shape("INPUT1", [1, 2], [1, 2], [max_batch, 2])

    config = builder.create_builder_config()
    config.add_optimization_profile(profile)

    # tensorrt 8.x
    # config.set_memory_pool_limit(trt.MemoryPoolType.WORKSPACE, 1 << 20)  # 1 MiB

    # tensorrt 7.x
    config.max_workspace_size = 1 << 20

    try:
        engine_bytes = builder.build_serialized_network(network, config)
    except AttributeError:
        engine = builder.build_engine(network, config)
        engine_bytes = engine.serialize()
        del engine

    with open(os.path.join(model_version_dir, 'model.plan'), "wb") as f:
        f.write(engine_bytes)
```

知乎 首发于 TensorFlow及其他深度学习框架

其他框架的模型参考：github.com/triton-infer...

模型配置文件

```
name: "tf_savemodel"
platform: "tensorflow_savedmodel"
max_batch_size: 8
version_policy: { latest { num_versions: 1 }}
input [
  {
    name: "INPUT0"
    data_type: TYPE_FP32
    dims: [ 2 ]
  },
  {
    name: "INPUT1"
    data_type: TYPE_INT32
    dims: [ 2 ]
  }
]
output [
  {
    name: "OUTPUT0"
    data_type: TYPE_FP32
    dims: [ 2 ]
  },
  {
    name: "OUTPUT1"
    data_type: TYPE_FP32
    dims: [ 2,10 ]
  }
]
```

name：模型名称，要跟模型路径对应。

platform：不同的模型存储格式都有自己对应的值。

max_batch_size：最大的batch_size，客户端超过这个batch_size的请求会报错。

version_policy：版本控制，这里是使用最新的一个版本。

input、output：输入和输出节点的名称，数据类型，维度。

维度一般不包括batch_size这个维度；

下表为不同框架对应的platform：

框架名称	platform值
TensorRT	tensorrt_plan
TensorFlow SavedModel	tensorflow_savedmodel
TensorFlow GraphDef	tensorflow_graphdef
ONNX	onnxruntime_onnx
Torch	pytorch_libtorch

下表是不同框架的数据类型对应关系：Model Config是配置文件的，API是triton client。其他框架是c++源码的命名空间，不过很好理解，主要包括16位和32位的int和float等等。

			Runtime			
TYPE_BOOL	kBOOL	DT_BOOL	BOOL	kBool	BOOL	bool
TYPE_UINT8		DT_UINT8	UINT8	kByte	UINT8	uint8
TYPE_UINT16		DT_UINT16	UINT16		UINT16	uint16
TYPE_UINT32		DT_UINT32	UINT32		UINT32	uint32
TYPE_UINT64		DT_UINT64	UINT64		UINT64	uint64
TYPE_INT8	kINT8	DT_INT8	INT8	kChar	INT8	int8
TYPE_INT16		DT_INT16	INT16	kShort	INT16	int16
TYPE_INT32	kINT32	DT_INT32	INT32	kInt	INT32	int32
TYPE_INT64		DT_INT64	INT64	kLong	INT64	int64
TYPE_FP16	kHALF	DT_HALF	FLOAT16		FP16	float16
TYPE_FP32	kFLOAT	DT_FLOAT	FLOAT	kFloat	FP32	float32
TYPE_FP64		DT_DOUBLE	DOUBLE	kDouble	FP64	float64
TYPE_STRING		DT_STRING	STRING		BYTES	dtype(object)

Triton Client

上述提到了，我们可以通过triton client来进行模型推理的请求，并且提供了http和grpc两种协议。

接下来，将以python来演示，仍然是上面那个简单的模型请求例子。

```
# 安装依赖包
pip install tritonclient[all]
```

```
import gevent.ssl
import numpy as np
import tritonclient.http as httpclient


def client_init(url="localhost:8000",
                ssl=False, key_file=None, cert_file=None, ca_certs=None, insecure=Fals
                verbose=False):
    """

    :param url:
    :param ssl: Enable encrypted link to the server using HTTPS
    :param key_file: File holding client private key
    :param cert_file: File holding client certificate
    :param ca_certs: File holding ca certificate
    :param insecure: Use no peer verification in SSL communications. Use with cautior
    :param verbose: Enable verbose output
    :return:
    """
    if ssl:
        ssl_options = {}
        if key_file is not None:
            ssl_options['keyfile'] = key_file
        if cert_file is not None:
            ssl_options['certfile'] = cert_file
```

```
        ssl_context_factory = None
        if insecure:
            ssl_context_factory = gevent.ssl._create_unverified_context
        triton_client = httpclient.InferenceServerClient(
            url=url,
            verbose=verbose,
            ssl=True,
            ssl_options=ssl_options,
            insecure=insecure,
            ssl_context_factory=ssl_context_factory)
    else:
        triton_client = httpclient.InferenceServerClient(
            url=url, verbose=verbose)

    return triton_client


def infer(triton_client, model_name,
          input0='INPUT0', input1='INPUT1',
          output0='OUTPUT0', output1='OUTPUT1',
          request_compression_algorithm=None,
          response_compression_algorithm=None):
    """

    :param triton_client:
    :param model_name:
    :param input0:
    :param input1:
    :param output0:
    :param output1:
    :param request_compression_algorithm: Optional HTTP compression algorithm to use f
           Currently supports "deflate", "gzip" and None. By default, no compression
    :param response_compression_algorithm:
    :return:
    """
    inputs = []
    outputs = []
    # batch_size=8
    # 如果batch_size超过配置文件的max_batch_size，infer则会报错
    # INPUT0、INPUT1为配置文件中的输入节点名称
    inputs.append(httpclient.InferInput(input0, [8, 2], "FP32"))
    inputs.append(httpclient.InferInput(input1, [8, 2], "INT32"))

    # Initialize the data
    # np.random.seed(2022)
    inputs[0].set_data_from_numpy(np.random.random([8, 2]).astype(np.float32), binary_
    # np.random.seed(2022)
    inputs[1].set_data_from_numpy(np.random.randint(0, 20, [8, 2]).astype(np.int32), b

    # OUTPUT0、OUTPUT1为配置文件中的输出节点名称
    outputs.append(httpclient.InferRequestedOutput(output0, binary_data=False))
    outputs.append(httpclient.InferRequestedOutput(output1,
                                                   binary_data=False))
    query_params = {'test_1': 1, 'test_2': 2}
    results = triton_client.infer(
        model_name=model_name,
        inputs=inputs,
        outputs=outputs,
        request_compression_algorithm=request_compression_algorithm,
        response_compression_algorithm=response_compression_algorithm)
    print(results)
    # 转化为numpy格式
    print(results.as_numpy(output0))
    print(results.as_numpy(output1))
```

总结

这篇文章非常基础，算是个人对tensorrt和triton的入门学习笔记了。后面会继续深入学习，然后更新文章，暂时安排的是这几个主题：

1. 在triton中，不同框架生成的模型推理性能比较，以及与tensorflow serving的比较；
2. triton serving 模型的定制化配置；
3. triton client进一步使用，例如对模型的加载等；
4. tensorrt runtime，即使用tensorrt加载和推理模型。

编辑于 2023-11-27 22:03 · IP 属地广东