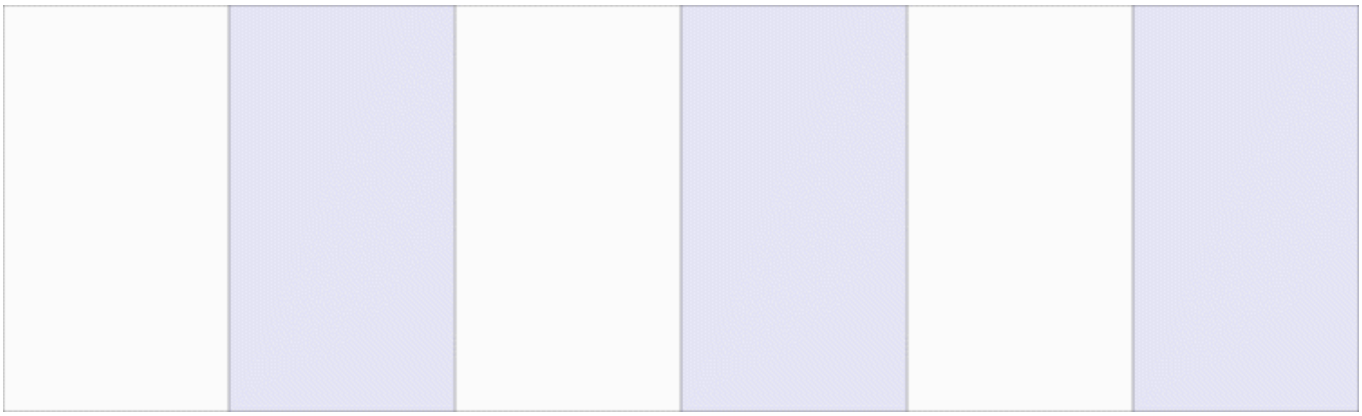
在C++平台上部署PyTorch模型流程+踩坑实录

CV开发者都爱看的 极市平台 2023-02-04 22:00:12 发表于广东 手机阅读



作者丨火星少女@知乎

来源丨https://zhuanlan.zhihu.com/p/146453159

编辑丨极市平台

极市导读

本文主要讲解如何将pytorch的模型部署到c++平台上的模型流程，按顺序分为四大块详细说明了模型转换、保存序列化模型、C ++中加载序列化的PyTorch模型以及执行Script Module。>>加入极市CV技术交流群，走在计算机视觉的最前沿

最近因为工作需要，要把pytorch的模型部署到c++平台上，基本过程主要参照官网的教学示例，期间发现了不少坑，特此记录。

1.模型转换

libtorch不依赖于python，python训练的模型，需要转换为script model才能由libtorch加载，并进行推理。在这一步官网提供了两种方法：

方法一：Tracing

这种方法操作比较简单，只需要给模型一组输入，走一遍推理网络，然后由torch.ji.trace记录一下路径上的信息并保存即可。示例如下：

```
import torch
import torchvision
```

```
# An instance of your model.
model = torchvision.models.resnet18()

# An example input you would normally provide to your model's forwarc
example = torch.rand(1, 3, 224, 224)

# Use torch.jit.trace to generate a torch.jit.ScriptModule via tracir
traced_script_module = torch.jit.trace(model, example)
```

缺点是如果模型中存在控制流比如if-else语句，一组输入只能遍历一个分支，这种情况下就没办法完整的把模型信息记录下来。

方法二：Scripting

直接在Torch脚本中编写模型并相应地注释模型，通过 `torch.jit.script` 编译模块，将其转换为 `ScriptModule` 。示例如下：

```
class MyModule(torch.nn.Module):
    def __init__(self, N, M):
        super(MyModule, self).__init__()
        self.weight = torch.nn.Parameter(torch.rand(N, M))

    def forward(self, input):
        if input.sum() > 0:
          output = self.weight.mv(input)
        else:
          output = self.weight + input
        return output

my_module = MyModule(10,20)
sm = torch.jit.script(my_module)
```

forward方法会被默认编译，forward中被调用的方法也会按照被调用的顺序被编译

如果想要编译一个forward以外且未被forward调用的方法，可以添加 **@torch.jit.export.**

如果想要方法不被编译，可使用

@torch.jit.ignore

（https://pytorch.org/docs/master/generated/torch.jit.ignore.html#torch.jit.ignore）

或者 **@torch.jit.unused**

（https://pytorch.org/docs/master/generated/torch.jit.unused.html#torch.jit.unused）

```
# Same behavior as pre-PyTorch 1.2
@torch.jit.script
def some_fn():
    return 2

# Marks a function as ignored, if nothing
# ever calls it then this has no effect
@torch.jit.ignore
def some_fn2():
    return 2

# As with ignore, if nothing calls it then it has no effect.
# If it is called in script it is replaced with an exception.
@torch.jit.unused
def some_fn3():
  import pdb; pdb.set_trace()
  return 4

# Doesn't do anything, this function is already
# the main entry point
@torch.jit.export
def some_fn4():
    return 2
```

在这一步遇到好多坑，主要原因可归为一下两点

1. 不支持的操作

TorchScript支持的操作是python的子集，大部分torch中用到的操作都可以找到对应实现，但也存在一些尴尬的不支持操作，详细列表可见 https://pytorch.org/docs/master/jit_unsupported.html#jit-unsupported，下面列一些我自己遇到的操作：

1）参数/返回值不支持可变个数，例如

```
def __init__(self, **kwargs):
```

或者

```
if output_flag == 0:
    return reshape_logits
else:
    loss = self.loss(reshape_logits, term_mask, labels_id)
    return reshape_logits, loss
```

2)各种iteration操作

eg1.

```
layers = [int(a) for a in layers]
```

报错torch.jit.frontend.UnsupportedNodeError: ListComp aren't supported

可以改成：

```
for k in range(len(layers)):
    layers[k] = int(layers[k])
```

eg2.

```
seq_iter = enumerate(scores)
try:
    _, inivalues = seq_iter.__next__()
except:
    _, inivalues = seq_iter.next()
```

eg3.

```
line = next(infile)
```

3)不支持的语句

eg1. 不支持continue

torch.jit.frontend.UnsupportedNodeError: continue statements aren't supported

eg2. 不支持try-catch

torch.jit.frontend.UnsupportedNodeError: try blocks aren't supported

eg3. 不支持with语句

4）其他常见op/module

eg1. torch.autograd.Variable

解决：使用torch.ones/torch.randn等初始化+.float()/.long()等指定数据类型。

eg2. torch.Tensor/torch.LongTensor etc.

解决：同上

eg3. requires_grad参数只在torch.tensor中支持，torch.ones/torch.zeros等不可用

eg4. tensor.numpy()

eg5. tensor.bool()

解决：tensor.bool()用tensor>0代替

eg6. self.seg_emb(seg_fea_ids).to(embeds.device)

解决：需要转gpu的地方显示调用.cuda()

总之一句话：**除了原生python和pytorch以外的库，比如numpy什么的能不用就不用，尽量用pytorch的各种API。**

2. 指定数据类型

1）属性，大部分的成员数据类型可以根据值来推断，空的列表/字典则需要预先指定

```
from typing import Dict

class MyModule(torch.nn.Module):
    my_dict: Dict[str, int]

    def __init__(self):
        super(MyModule, self).__init__()
        # This type cannot be inferred and must be specified
        self.my_dict = {}

        # The attribute type here is inferred to be `int`
        self.my_int = 20

    def forward(self):
        pass

m = torch.jit.script(MyModule())
```

2）常量，使用*Final*关键字

```
try:
    from typing_extensions import Final
except:
    # If you don't have `typing_extensions` installed, you can use a
    # polyfill from `torch.jit`.
    from torch.jit import Final

class MyModule(torch.nn.Module):

    my_constant: Final[int]

    def __init__(self):
        super(MyModule, self).__init__()
```

```
        self.my_constant = 2

    def forward(self):
        pass

m = torch.jit.script(MyModule())
```

3）变量。默认是tensor类型且不可变，所以**非tensor类型必须要指明**

```
def forward(self, batch_size:int, seq_len:int, use_cuda:bool):
```

方法三：Tracing and Scriptin混合

一种是在trace模型中调用script，适合模型中只有一小部分需要用到控制流的情况，使用实例如下：

```
import torch

@torch.jit.script
def foo(x, y):
    if x.max() > y.max():
        r = x
    else:
        r = y
    return r


def bar(x, y, z):
    return foo(x, y) + z

traced_bar = torch.jit.trace(bar, (torch.rand(3), torch.rand(3), torc
```

另一种情况是在script module中用tracing生成子模块，对于一些存在script module不支持的python feature的layer，就可以把相关layer封装起来，用trace记录相关layer流，其他layer不用修改。使用示例如下：

```
import torch
import torchvision

class MyScriptModule(torch.nn.Module):
    def __init__(self):
        super(MyScriptModule, self).__init__()
        self.means = torch.nn.Parameter(torch.tensor([103.939, 116.77
                                        .resize_(1, 3, 1, 1))
```

```
        self.resnet = torch.jit.trace(torchvision.models.resnet18(),
                                      torch.rand(1, 3, 224, 224))

    def forward(self, input):
        return self.resnet(input - self.means)

my_script_module = torch.jit.script(MyScriptModule())
```

2.保存序列化模型

如果上一步的坑都踩完，那么模型保存就非常简单了，只需要调用save并传递一个文件名即可，需要注意的是如果想要**在gpu上训练模型，在cpu上做inference**，一定要在模型save之前转化，再就是记得调用model.eval(),形如

```
gpu_model.eval()
cpu_model = gpu_model.cpu()
sample_input_cpu = sample_input_gpu.cpu()
traced_cpu = torch.jit.trace(traced_cpu, sample_input_cpu)
torch.jit.save(traced_cpu, "cpu.pth")

traced_gpu = torch.jit.trace(traced_gpu, sample_input_gpu)
torch.jit.save(traced_gpu, "gpu.pth")
```

3.C++ load训练好的模型

要在C ++中加载序列化的PyTorch模型，必须依赖于PyTorch C ++ API（也称为LibTorch）。libtorch的安装非常简单，只需要在pytorch官网（https://pytorch.org/）下载对应版本，解压即可。会得到一个结构如下的文件夹。

```
libtorch/
  bin/
  include/
  lib/
  share/
```

然后就可以构建应用程序了，一个简单的示例目录结构如下：

```
example-app/
  CMakeLists.txt
  example-app.cpp
```

example-app.cpp和CMakeLists.txt的示例代码分别如下：

```
#include <torch/script.h> // One-stop header.
#include <iostream>#include <memory>
int main(int argc, const char* argv[]) {
  if (argc != 2) {
    std::cerr << "usage: example-app <path-to-exported-script-module>
    return -1;
  }


  torch::jit::script::Module module;
  try {
    // Deserialize the ScriptModule from a file using torch::jit::loa
    module = torch::jit::load(argv[1]);
  }
  catch (const c10::Error& e) {
    std::cerr << "error loading the model\n";
    return -1;
  }

  std::cout << "ok\n";
}
```

```
cmake_minimum_required(VERSION 3.0 FATAL_ERROR)
project(custom_ops)

find_package(Torch REQUIRED)

add_executable(example-app example-app.cpp)
target_link_libraries(example-app "${TORCH_LIBRARIES}")
set_property(TARGET example-app PROPERTY CXX_STANDARD 14)
```

至此，就可以运行以下命令从 example-app/ 文件夹中构建应用程序啦：

```
mkdir build
cd build
cmake -DCMAKE_PREFIX_PATH=/path/to/libtorch ..
cmake --build . --config Release
```

其中/path/to/libtorch是之前下载后的libtorch文件夹所在的路径。这一步如果顺利能够看到编译完成100%的提示，下一步运行编译生成的可执行文件，会看到“ok”的输出，可喜可贺！

4. 执行Script Module

终于到最后一步啦！下面只需要按照构建输入传给模型，执行forward就可以得到输出啦。一个简单的示例如下：

```
// Create a vector of inputs.
std::vector<torch::jit::IValue> inputs;
inputs.push_back(torch::ones({1, 3, 224, 224}));

// Execute the model and turn its output into a tensor.
at::Tensor output = module.forward(inputs).toTensor();
std::cout << output.slice(/*dim=*/1, /*start=*/0, /*end=*/5) << '\n';
```

前两行创建一个 torch::jit::IValue 的向量，并添加单个输入. 使用 torch::ones() 创建输入张量，等效于C ++ API中的 torch.ones 。然后，运行 script::Module 的 forward 方法，通过调用 toTensor() 将返回的IValue值转换为张量。C++对torch的各种操作还是比较友好的，通过torch::或者后加_的方法都可以找到对应实现，例如

```
torch::tensor(input_list[j]).to(at::kLong).resize_({batch, 128}).clor
//torch::tensor对应pytorch的torch.tensor; at::kLong对应torch.int64;res
```

最后check一下确保c++端的输出和pytorch是一致的就大功告成啦~

踩了无数坑，薅掉了无数头发，很多东西也是自己一点点摸索的，如果有错误欢迎指正！

参考资料：

PyTorch C++ API - PyTorch master document

Torch Script - PyTorch master documentation

文章地址：

https://pytorch.org/cppdocs/

https://pytorch.org/tutorials/advanced/cpp_export.html

极市干货

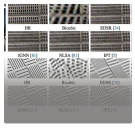
技术干货：损失函数技术总结及Pytorch使用示例 | 深度学习有哪些trick？ | 目标检测正负样本区分策略和平衡策略总结

实操教程：GPU多卡并行训练总结（以pytorch为例） | CUDA WarpReduce 学习笔记 | 卷积神经网络压缩方法总结

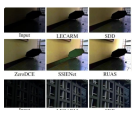
喜欢此内容的人还喜欢

ICCV 2023 | 南开程明明团队提出适用于SR任务的新颖注意力机制（已开源）

极市平台



ICCV23 | 将隐式神经表征用于低光增强，北大张健团队提出NeRCo



极市平台

YOLOv5帮助母猪产仔？南京农业大学研发母猪产仔检测模型并部署到Jetson Nano开发板

极市平台