2.TensorRT的功能

本章概述了您可以使用 TensorRT 做什么。它旨在对所有 TensorRT 用户有用。

2.1. C++ and Python APIs

TensorRT 的 API 具有 C++ 和 Python 的语言绑定，具有几乎相同的功能。 Python API 促进了与 Python 数据处理工具包和库（如 NumPy 和 SciPy）的互操作性。 C++ API 可以更高效，并且可以更好地满足某些合规性要求，例如在汽车应用中。
注意： Python API 并非适用于所有平台。有关详细信息，请参阅NVIDIA TensorRT 支持矩阵。

2.2. The Programming Model

TensorRT 构建阶段的最高级别接口是Builder （ C++ 、 Python ）。构建器负责优化模型并生成Engine 。

为了构建引擎，您需要：

- 创建网络定义
- 为builder指定配置
- 调用builder创建引擎

`NetworkDefinition`接口（ C++ 、 Python ）用于定义模型。将模型传输到 TensorRT 的最常见途径是以 ONNX 格式从框架中导出模型，并使用 TensorRT 的 ONNX 解析器来填充网络定义。但是，您也可以使用 TensorRT 的Layer (C++ , Python) 和Tensor (C++ , Python) 接口逐步构建定义。

无论您选择哪种方式，您还必须定义哪些张量是网络的输入和输出。未标记为输出的张量被认为是可以由构建器优化掉的瞬态值。输入和输出张量必须命名，以便在运行时，TensorRT 知道如何将输入和输出缓冲区绑定到模型。

`BuilderConfig`接口（ C++ 、 Python ）用于指定TensorRT如何优化模型。在可用的配置选项中，您可以控制 TensorRT 降低计算精度的能力，控制内存和运行时执行速度之间的权衡，以及限制对 CUDA ®内核的选择。由于构建器可能需要几分钟或更长时间才能运行，因此您还可以控制构建器搜索内核的方式，以及缓存搜索结果以供后续运行使用。

一旦有了网络定义和构建器配置，就可以调用构建器来创建引擎。构建器消除了无效计算、折叠常量、重新排序和组合操作以在 GPU 上更高效地运行。它可以选择性地降低浮点计算的精度，方法是简单地在 16 位浮点中运行它们，或者通过量化浮点值以便可以使用 8 位整数执行计算。它还使用不同的数据格式对每一层的多次实现进行计时，然后计算执行模型的最佳时间表，从而最大限度地降低内核执行和格式转换的综合成本。

构建器以称为计划的序列化形式创建引擎，该计划可以立即反序列化，或保存到磁盘以供以后使用。

注意：

- TensorRT 创建的引擎特定于创建它们的 TensorRT 版本和创建它们的 GPU。
- TensorRT 的网络定义不会深度复制参数数组（例如卷积的权重）。因此，在构建阶段完成之前，您不得释放这些阵列的内存。使用 ONNX 解析器导入网络时，解析器拥有权重，因此在构建阶段完成之前不得将其销毁。
- 构建器时间算法以确定最快的。与其他 GPU 工作并行运行构建器可能会扰乱时序，导致优化不佳。

2.2.2. The Runtime Phase

TensorRT 执行阶段的最高级别接口是`Runtime`（C++、Python）。
使用运行时时，您通常会执行以下步骤：

- 反序列化创建引擎的计划(plan 文件)
- 从引擎创建执行上下文(context)
  然后，反复：
- 填充输入缓冲区以进行推理
- 调用enqueue()或execute()以运行推理

`Engine`接口（C++、Python）代表一个优化模型。您可以查询引擎以获取有关网络输入和输出张量的信息——预期的维度、数据类型、数据格式等。

`ExecutionContext`接口（C++、Python）是调用推理的主要接口。执行上下文包含与特定调用关联的所有状态 - 因此您可以拥有与单个引擎关联的多个上下文，并并行运行它们。

调用推理时，您必须在适当的位置设置输入和输出缓冲区。根据数据的性质，这可能在 CPU 或 GPU 内存中。如果根据您的模型不明显，您可以查询引擎以确定在哪个内存空间中提供缓冲区。

设置缓冲区后，可以同步（执行）或异步（入队）调用推理。在后一种情况下，所需的内核在 CUDA 流上排队，并尽快将控制权返回给应用程序。一些网络需要在 CPU 和 GPU 之间进行多次控制传输，因此控制可能不会立即返回。要等待异步执行完成，请使用`cudaStreamSynchronize`在流上同步。

2.3. Plugins

TensorRT 有一个`Plugin`接口，允许应用程序提供 TensorRT 本身不支持的操作的实现。在转换网络时，ONNX 解析器可以找到使用 TensorRT 的`PluginRegistry`创建和注册的插件。

TensorRT 附带一个插件库，其中许多插件和一些附加插件的源代码可以在此处找到。

请参阅使用自定义层扩展 TensorRT一章。

2.4. Types and Precision

TensorRT 支持使用`FP32`、`FP16`、`INT8`、`Bool`和`INT32`数据类型的计算。
当 TensorRT 选择 CUDA 内核在网络中实现浮点运算时，它默认为`FP32`实现。有两种方法可以配置不同的精度级别：

- 为了在模型级别控制精度， BuilderFlag选项（C++、Python）可以向 TensorRT 指示它在搜索最快时可能会选择较低精度的实现（并且因为较低的精度通常更快，如果允许的话，它通常会）。因此，您可以轻松地指示 TensorRT 为您的整个模型使用 FP16 计算。对于输入动态范围约为 1 的正则化模型，这通常会产生显着的加速，而准确度的变化可以忽略不计。
- 对于更细粒度的控制，由于网络的一部分对数值敏感或需要高动态范围，因此层必须以更高的精度运行，可以为该层指定算术精度。

请参阅降低精度部分。

2.5. Quantization

TensorRT 支持量化浮点，其中浮点值被线性压缩并四舍五入为 8 位整数。这显着提高了算术吞吐量，同时降低了存储要求和内存带宽。在量化浮点张量时，TensorRT 需要知道它的动态范围——即表示什么范围的值很重要——量化时会钳制超出该范围的值。

动态范围信息可由构建器根据代表性输入数据计算（这称为校准--`calibration`）。或者，您可以在框架中执行量化感知训练，并将模型与必要的动态范围信息一起导入到 TensorRT。

请参阅使用 INT8章节。

2.6. Tensors and Data Formats

在定义网络时，TensorRT 假设张量由多维 C 样式数组表示。每一层对其输入都有特定的解释：例如，2D 卷积将假定其输入的最后三个维度是 CHW 格式 - 没有选项可以使用，例如 WHC 格式。有关每个层如何解释其输入，请参阅TensorRT 网络层一章。

请注意，张量最多只能包含 2^31-1 个元素。
在优化网络的同时，TensorRT 在内部执行转换（包括到 HWC，但也包括更复杂的格式）以使用尽可能快的 CUDA 内核。通常，选择格式是为了优化性能，而应用程序无法控制这些选择。然而，底层数据格式暴露在 I/O 边界（网络输入和输出，以及将数据传入和传出插件），以允许应用程序最大限度地减少不必要的格式转换。

请参阅I/O 格式部分

2.7. Dynamic Shapes

默认情况下，TensorRT 根据定义时的输入形状（批量大小、图像大小等）优化模型。但是，可以将构建器配置为允许在运行时调整输入维度。为了启用此功能，您可以在构建器配置中指定一个或多个 `OptimizationProfile` （ C++ 、 Python ）实例，其中包含每个输入的最小和最大形状，以及该范围内的优化点。

TensorRT 为每个配置文件创建一个优化的引擎，选择适用于 [最小、最大] 范围内的所有形状的 CUDA 内核，并且对于优化点来说是最快的——通常每个配置文件都有不同的内核。然后，您可以在运行时在配置文件中进行选择。

请参阅使用动态形状一章。

2.8. DLA

TensorRT 支持 NVIDIA 的深度学习加速器 (DLA)，这是许多 NVIDIA SoC 上的专用推理处理器，支持 TensorRT 层的子集。 TensorRT 允许您在 DLA 上执行部分网络，而在 GPU 上执行其余部分；对于可以在任一设备上执行的层，您可以在构建器配置中逐层选择目标设备。

请参阅使用 DLA章节。

2.9. Updating Weights

在构建引擎时，您可以指定它可能需要稍后更新其权重。如果您经常在不更改结构的情况下更新模型的权重，例如在强化学习中或在保留相同结构的同时重新训练模型时，这将很有用。权重更新是通过 `Refitter` (C++ , Python) 接口执行的。

请参阅Refitting An Engine 部分。

2.10. trtexec

示例目录中包含一个名为 `trtexec` 的命令行包装工具。 `trtexec` 是一种无需开发自己的应用程序即可快速使用 TensorRT 的工具。 trtexec工具有三个主要用途：

- 在随机或用户提供的输入数据上对网络进行基准测试。
- 从模型生成序列化引擎。
- 从构建器生成序列化时序缓存。

请参阅trtexec部分。

2.11. Polygraphy

Polygraphy 是一个工具包，旨在帮助在 TensorRT 和其他框架中运行和调试深度学习模型。它包括一个 Python API和一个使用此 API 构建的命令行界面 (CLI) 。

除此之外，使用 Polygraphy，您可以：

- 在多个后端之间运行推理，例如 TensorRT 和 ONNX-Runtime，并比较结果（例如API 、 CLI ）
- 将模型转换为各种格式，例如具有训练后量化的 TensorRT 引擎（例如API 、 CLI ）
- 查看有关各种类型模型的信息（例如CLI ）
- 在命令行上修改 ONNX 模型：
  - 提取子图（例如CLI ）
  - 简化和清理（例如CLI ）
- 隔离 TensorRT 中的错误策略（例如CLI ）

有关更多详细信息，请参阅Polygraphy 存储库。