TensorRT之条件控制

NVIDIA TensorRT 支持条件 if-then-else 流控制。 TensorRT 条件用于实现网络子图的条件执行。

11.1. Defining A Conditional

if-conditional 由条件边界层定义：

- `IConditionLayer`表示predicate 并指定条件是应该执行真分支（`then-branch`）还是假分支（`else-branch`）。
- `IIfConditionalInputLayer`指定两个条件分支之一的输入。
- `IIfConditionalOutputLayer`指定条件的输出。

每个边界层都继承自`IIfConditionalBoundaryLayer`类，该类具有获取其关联`IIfConditional`的方法`getConditional()`。 `IIfConditional`实例标识条件。所有具有相同`IIfConditional`的条件边界层都属于该条件。

条件必须恰好有一个`IConditionLayer`实例、零个或多个`IIfConditionalInputLayer`实例，以及至少一个`IIfConditionalOutputLayer`实例。

`IIfConditional`实现了一个`if-then-else`流控制结构，该结构提供基于动态布尔输入的网络子图的条件执行。它由一个布尔标量predicate condition和两个分支子图定义：一个`trueSubgraph`在`condition`评估为`true`时执行，一个`falseSubgraph`在`condition`评估为`false`时执行

```
If condition is true then:
    output = trueSubgraph(trueInputs);
Else
    output = falseSubgraph(falseInputs);
Emit output
```

真分支和假分支都必须定义，类似于许多编程语言中的三元运算符。

要定义`if-conditional`，使用方法`INetworkDefinition::addIfConditional`创建一个`IIfConditional`实例，然后添加边界层和分支层。

```
IIfConditional* simpleIf = network->addIfConditional();
```

`IIfConditional ::setCondition`方法接受一个参数：`条件张量`。这个 0D 布尔张量（标量）可以由网络中的早期层动态计算。它用于决定执行哪个分支。 `IConditionLayer`有一个输入（条件）并且没有输出，因为它由条件实现在内部使用。

```
// Create a condition predicate that is also a network input.
auto cond = network->addInput("cond", DataType::kBOOL, Dims{0});
IConditionLayer* condition = simpleIf->setCondition(*cond);
```

TensorRT 不支持实现条件分支的子图抽象，而是使用`IIfConditionalInputLayer`和`IIfConditionalOutputLayer`来定义条件的边界。

- `IIfConditionalInputLayer`将单个输入抽象为`IIfConditional`的一个或两个分支子图。特定`IIfConditionalInputLayer`的输出可以同时提供两个分支。`then-branch`和`else-branch`的

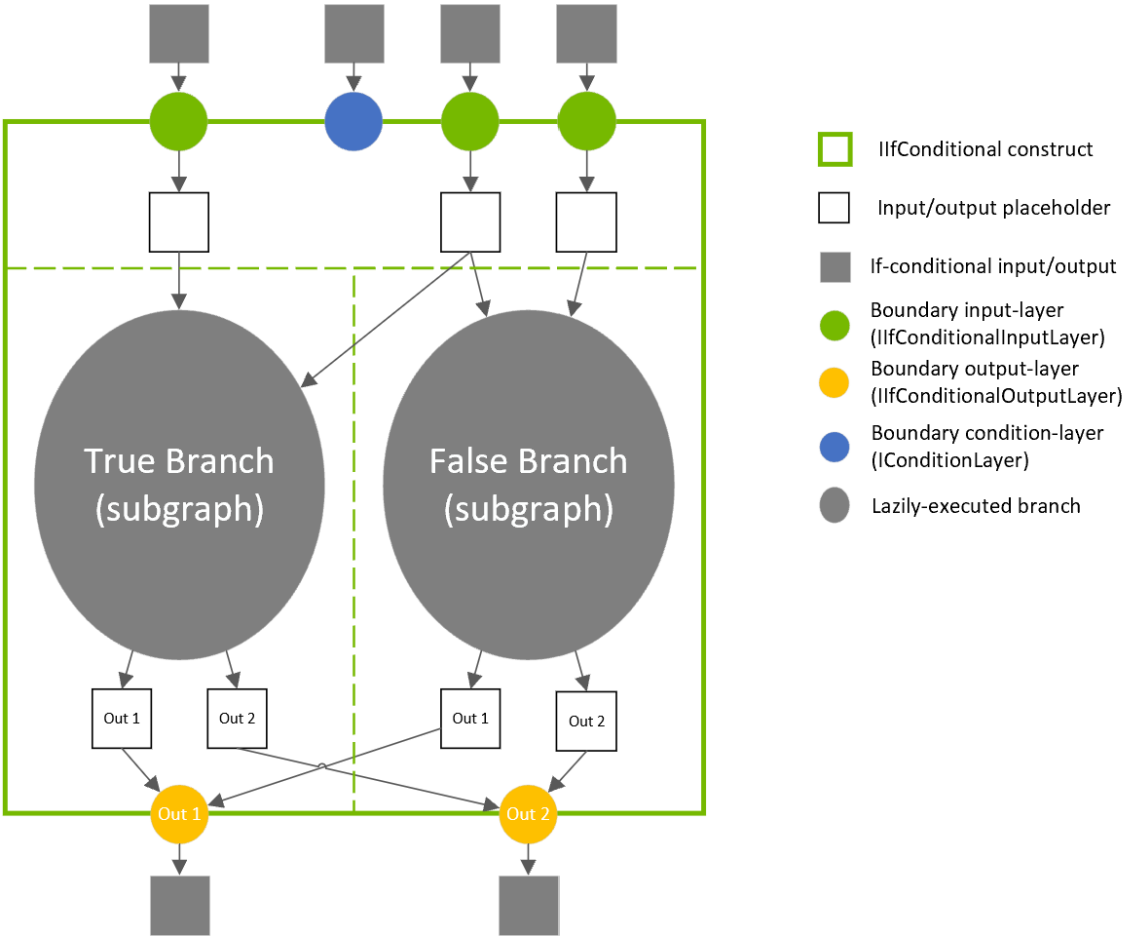
输入不需要是相同的类型和形状，每个分支可以独立地包含零个或多个输入。
`IIfConditionalInputLayer`是可选的，用于控制哪些层将成为分支的一部分（请参阅条件执行）。如果分支的所有输出都不依赖于`IIfConditionalInputLayer`实例，则该分支为空。当条件为false时没有要评估的层时，空的 `else-branch` 可能很有用，并且网络评估应按照条件进行（请参阅条件示例）。

```
// Create an if-conditional input.
// x is some arbitrary Network tensor.
IIfConditionalInputLayer* inputX = simpleIf->addInput(*x);
```

- `IIfConditionalOutputLayer`抽象了 `if` 条件的单个输出。它有两个输入：来自真子图的输出（输入索引 0）和来自假子图的输出（输入索引 1）。 `IIfConditionalOutputLayer`的输出可以被认为是最终输出的占位符，最终输出将在运行时确定。`IIfConditionalOutputLayer`的作用类似于传统 SSA 控制流图中的 $\Phi(Phi)$ 函数节点。它的语义是：选择真子图或假子图的输出。`IIfConditional`的所有输出都必须源自`IIfConditionalOutputLayer`实例。没有输出的 if 条件对网络的其余部分没有影响，因此，它被认为是病态的。两个分支（子图）中的每一个也必须至少有一个输出。 `if-conditional` 的输出可以标记为网络的输出，除非 `if-conditional` 嵌套在另一个 `if-conditional` 或循环中。

```
// trueSubgraph and falseSubgraph represent network subgraphs
IIfConditionalOutputLayer* outputLayer = simpleIf->addOutput(
    *trueSubgraph->getOutput(0),
    *falseSubgraph->getOutput(0));
```

下图提供了 if 条件抽象模型的图形表示。绿色矩形表示条件的内部，仅限于NVIDIA TensorRT 支持矩阵中的Layers For Flow-Control Constructs部分中列出的层类型。



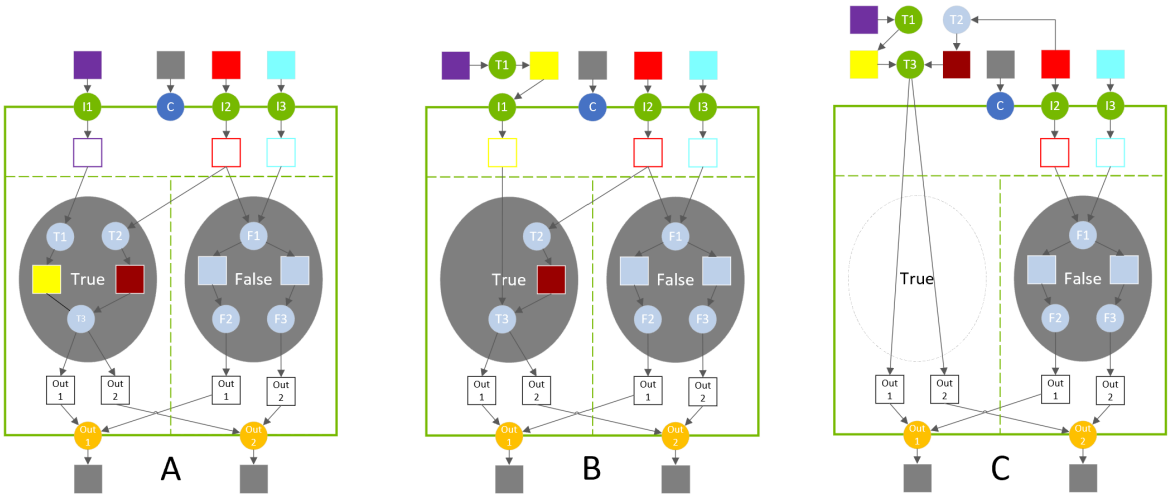
11.2. Conditional Execution

网络层的条件执行是一种网络评估策略，其中仅在需要分支输出的值时才执行分支层（属于条件子图的层）。在条件执行中，无论是真分支还是假分支都被执行并允许改变网络状态。

相反，在断定执行中，真分支和假分支都被执行，并且只允许其中之一改变网络评估状态，具体取决于条件断定的值（即仅其中一个的输出）子图被馈送到以下层。

条件执行有时称为惰性求值，断定执行有时称为急切求值。
`IIfConditionalInputLayer`的实例可用于指定急切调用哪些层以及延迟调用哪些层。这是通过从每个条件输出开始向后跟踪网络层来完成的。依赖于至少一个`IIfConditionalInputLayer`输出的数据层被认为是条件内部的，因此被延迟评估。在没有`IIfConditionalInputLayer`实例添加到条件条件的极端情况下，所有层都被急切地执行，类似于`ISelectLayer`。

下面的三个图表描述了`IIfConditionalInputLayer`放置的选择如何控制执行调度。



在图 A 中，真分支由 3 层（T1、T2、T3）组成。当条件评估为true时，这些层会延迟执行。

在图 B 中，输入层 I1 放置在层 T1 之后，它将 T1 移出真实分支。在评估 if 结构之前，T1 层急切地执行。

在图表 C 中，输入层 I1 被完全移除，这将 T3 移到条件之外。 T2 的输入被重新配置以创建合法网络，并且 T2 也移出真实分支。当条件评估为true时，条件不计算任何内容，因为输出已经被急切地计算（但它确实将条件相关输入复制到其输出）。

11.3. Nesting and Loops

条件分支可以嵌套其他条件，也可以嵌套循环。循环可以嵌套条件。与循环嵌套一样，TensorRT 从数据流中推断条件和循环的嵌套。例如，如果条件 B 使用在循环 A 内定义的值，则 B 被认为嵌套在 A 内。

真分支中的层与假分支中的层之间不能有交叉边，反之亦然。换句话说，一个分支的输出不能依赖于另一个分支中的层。

例如，请参阅条件示例以了解如何指定嵌套。

11.4. Limitations

两个真/假子图分支中的输出张量数必须相同。来自分支的每个输出张量的类型和形状必须相同。

请注意，这比 ONNX 规范更受限制，ONNX 规范要求真/假子图具有相同数量的输出并使用相同的输出数据类型，但允许不同的输出形状。

11.5. Conditional Examples

11.5.1. Simple If-Conditional

下面的例子展示了如何实现一个简单的条件，它有条件地对两个张量执行算术运算。
Conditional

```
condition = true
If condition is true:
        output = x + y
Else:
        output = x - y
```

Example

```
ITensor* addCondition(INetworkDefinition& n, bool predicate)
{
    // The condition value is a constant int32 input that is cast to boolean
because TensorRT doesn't support boolean constant layers.

    static const Dims scalarDims = Dims{0, {}};
    static float constexpr zero{0};
    static float constexpr one{1};

    float* const val = predicate ? &one : &zero;

    ITensor* cond =
        n.addConstant(scalarDims, DataType::kINT32, val, 1})->getOutput(0);

    auto* cast = n.addIdentity(cond);
    cast->setOutputType(0, DataType::kBOOL);
    cast->getOutput(0)->setType(DataType::kBOOL);

    return cast->getOutput(0);
}

IBuilder* builder = createInferBuilder(gLogger);
INetworkDefinition& n = *builder->createNetworkV2(0U);
auto x = n.addInput("x", DataType::kFLOAT, Dims{1, {5}});
auto y = n.addInput("y", DataType::kFLOAT, Dims{1, {5}});
ITensor* cond = addCondition(n, true);

auto* simpleIf = n.addIfConditional();
simpleIf->setCondition(*cond);

// Add input layers to demarcate entry into true/false branches.
x = simpleIf->addInput(*x)->getOutput(0);
y = simpleIf->addInput(*y)->getOutput(0);

auto* trueSubgraph = n.addElementWise(*x, *y, ElementWiseOperation::kSUM)-
>getOutput(0);
auto* falseSubgraph = n.addElementWise(*x, *y, ElementWiseOperation::kSUB)-
>getOutput(0);

auto* output = simpleIf->addOutput(*trueSubgraph, *falseSubgraph)->getOutput(0);
n.markOutput(*output);
```

11.5.2. Exporting from PyTorch

以下示例展示了如何将脚本化的 PyTorch 代码导出到 ONNX。函数 `sum_even` 中的代码执行嵌套在循环中的 if 条件。

```
import torch.onnx
import torch
import tensorrt as trt
import numpy as np

TRT_LOGGER = trt.Logger(trt.Logger.WARNING)
EXPLICIT_BATCH = 1 << (int)(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH)
```

```
@torch.jit.script
def sum_even(items):
    s = torch.zeros(1, dtype=torch.float)
    for c in items:
        if c % 2 == 0:
            s += c
    return s

class ExampleModel(torch.nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, items):
        return sum_even(items)

def build_engine(model_file):
    builder = trt.Builder(TRT_LOGGER)
    network = builder.create_network(EXPLICIT_BATCH)
    config = builder.create_builder_config()
    parser = trt.OnnxParser(network, TRT_LOGGER)

    with open(model_file, 'rb') as model:
        assert parser.parse(model.read())
        return builder.build_engine(network, config)

def export_to_onnx():
    items = torch.zeros(4, dtype=torch.float)
    example = ExampleModel()
    torch.onnx.export(example, (items), "example.onnx", verbose=False,
opset_version=13, enable_onnx_checker=False, do_constant_folding=True)

export_to_onnx()
build_engine("example.onnx")
```