TensorRT中的循环

NVIDIA TensorRT 支持循环结构，这对于循环网络很有用。 TensorRT 循环支持扫描输入张量、张量的循环定义以及“扫描输出”和“最后一个值”输出。

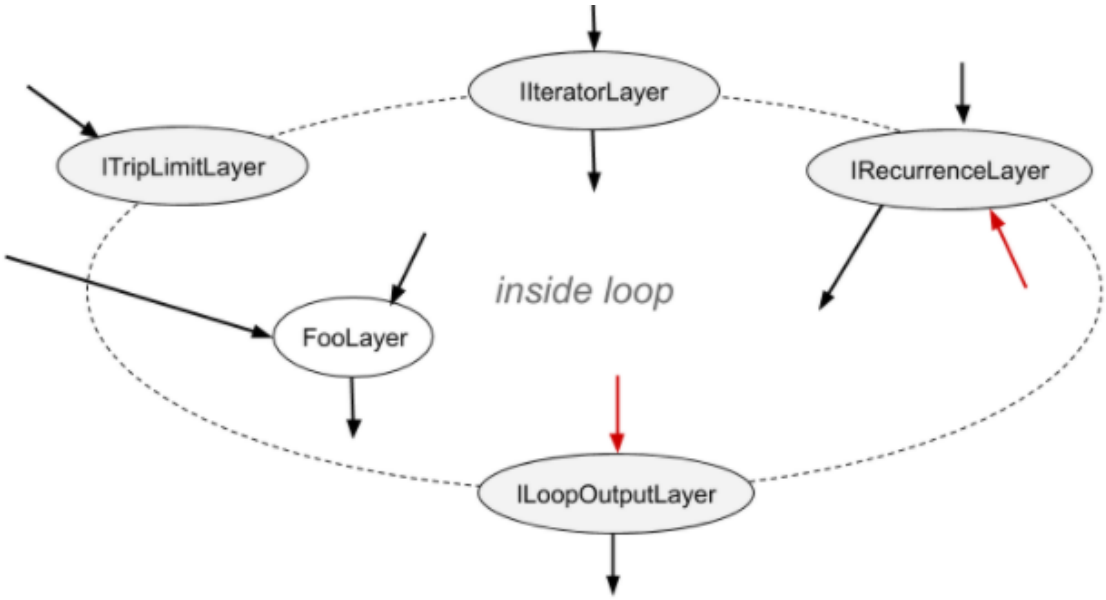
10.1. Defining A Loop

循环由循环边界层(loop boundary layers)定义。

- `ITripLimitLayer` 指定循环迭代的次数。
- `IIteratorLayer` 使循环能够迭代张量。
- `IRecurrenceLayer` 指定一个循环定义。
- `ILoopOutputLayer` 指定循环的输出。

每个边界层都继承自类 `ILoopBoundaryLayer` ，该类有一个方法 `getLoop()` 用于获取其关联的 `ILoop` 。 `ILoop` 对象标识循环。具有相同 `ILoop` 的所有循环边界层都属于该循环。

下图描绘了循环的结构和边界处的数据流。循环不变张量可以直接在循环内部使用，例如 FooLayer 所示。



一个循环可以有多个 `IteratorLayer` 、 `IRecurrenceLayer` 和 `ILoopOutputLayer` ，并且最多可以有两个 `ITripLimitLayer` ，如后面所述。没有 `ILoopOutputLayer` 的循环没有输出，并由 TensorRT 优化。

NVIDIA TensorRT 支持矩阵中的流控制结构层部分描述了可用于循环内部的 TensorRT 层。

内部层可以自由使用在循环内部或外部定义的张量。内部可以包含其他循环（请参阅嵌套循环）和其他条件构造（请参阅条件嵌套）。

要定义循环，首先，使用 `INetworkDefinition ::addLoop` 方法创建一个 `ILoop` 对象。然后添加边界层和内部层。本节的其余部分描述了边界层的特征，使用 `loop` 表示 `INetworkDefinition ::addLoop` 返回的 `ILoop*` 。

`ITripLimitLayer` 支持计数循环和 `while` 循环。

- `loop ->addTripLimit( t ,TripLimit::kCOUNT)` 创建一个 `ITripLimitLayer` ，其输入 `t` 是指定循环迭代次数的 0D INT32 张量。

- `loop ->addTripLimit( t ,TripLimit::kWHILE)` 创建一个 `ITripLimitLayer` ，其输入 `t` 是一个 0D Bool 张量，用于指定是否应该进行迭代。通常 `t` 要么是 `IRecurrenceLayer` 的输出，要么是基于所述输出的计算。

一个循环最多可以有一种限制。

`IIteratorLayer` 支持在任何轴上向前或向后迭代。

- `loop ->addIterator( t )` 添加一个 `IIteratorLayer` ，它在张量t的轴 0 上进行迭代。例如，如果输入是矩阵：

```
2 3 5
4 6 8
```

第一次迭代的一维张量{2, 3, 5}和第二次迭代的{4, 6, 8} 。超出张量范围的迭代是无效的。

- `loop ->addIterator( t , axis )` 类似，但该层在给定的轴上迭代。例如，如果 axis=1 并且输入是矩阵，则每次迭代都会传递矩阵的一列。
- `loop ->addIterator( t , axis,reverse )` 类似，但如果 `reverse =true` ，则该层以相反的顺序产生其输出。

`ILoopOutputLayer` 支持三种形式的循环输出：

- `loop ->addLoopOutput( t, LoopOutput::kLAST_VALUE)` 输出 `t` 的最后一个值，其中 `t` 必须是 `IRecurrenceLayer` 的输出。
- `loop-> addLoopOutput( t ,LoopOutput::kCONCATENATE, axis )` 将每次迭代的输入串联输出到 `t` 。例如，如果输入是一维张量，第一次迭代的值为 `{ a,b,c}` ，第二次迭代的值为 `{d,e,f}` ， axis =0 ，则输出为矩阵：

```
a b c
d e f
```

如果axis =1 ，则输出为：

```
a d
b e
c f
```

- `loop-> addLoopOutput( t ,LoopOutput::kREVERSE, axis )` 类似，但颠倒了顺序。

`kCONCATENATE` 和 `kREVERSE` 形式都需要第二个输入，这是一个 0D INT32 形状张量，用于指定新输出维度的长度。当长度大于迭代次数时，额外的元素包含任意值。第二个输入，例如 `u` ，应使用 `ILoopOutputLayer::setInput(1, u )` 设置。

最后，还有 `IRecurrenceLayer` 。它的第一个输入指定初始输出值，第二个输入指定下一个输出值。第一个输入必须来自循环外；第二个输入通常来自循环内部。例如，这个 C++ 片段的 TensorRT 模拟：

```
for (int32_t i = j; ...; i += k) ...
```

可以通过这些调用创建，其中j和k是 `ITensor*` 。

```
ILoop* loop = n.addLoop();
IRecurrenceLayer* iRec = loop->addRecurrence(j);
ITensor* i = iRec->getOutput(0);
ITensor* iNext = addElementWise(*i, *k,
    ElementWiseOperation::kADD)->getOutput(0);
iRec->setInput(1, *iNext);
```

第二个输入是TensorRT允许后沿的唯一情况。如果删除了这些输入，则剩余的网络必须是非循环的。

10.2. Formal Semantics

TensorRT 具有应用语义，这意味着除了引擎输入和输出之外没有可见的副作用。因为没有副作用，命令式语言中关于循环的直觉并不总是有效。本节定义了 TensorRT 循环结构的形式语义。

形式语义基于张量的惰性序列(lazy sequences)。循环的每次迭代对应于序列中的一个元素。循环内张量X的序列表示为`⟨ X 0, X 1, X 2, ... ⟩`。序列的元素被懒惰地评估，意思是根据需要。

`IIteratorLayer(X)` 的输出是`⟨ X[0], X[1], X[2], ... ⟩`其中`X[i]`表示在`IIteratorLayer`指定的轴上的下标。

`IRecurrenceLayer(X,Y)` 的输出是`⟨ X, Y0, Y1, Y2, ... ⟩`。
的输入和输出取决于LoopOutput的类型。

- `kLAST_VALUE`： 输入是单个张量X，对于 n-trip 循环，输出是X n 。
- `kCONCATENATE`： 第一个输入是张量`X`，第二个输入是标量形状张量`Y`。结果是`X0, X1, X2, ... Xn-1`与后填充（如有必要）连接到`Y`指定的长度。如果`Y < n`则为运行时错误。 `Y`是构建时间常数。注意与`IIteratorLayer`的反比关系。 `IIteratorLayer`将张量映射到一系列子张量；带有`kCONCATENATE`的`ILoopOutputLayer`将一系列子张量映射到一个张量。
- kREVERSE： 类似于`kCONCATENATE`，但输出方向相反。

`ILoopOutputLayer`的输出定义中的n值由循环的`ITripLimitLayer`确定：

- 对于计数循环，它是迭代计数，表示ITripLimitLayer的输入。
- 对于 while 循环，它是最小的 n 使得X_n为假，其中`X`是`ITripLimitLayer`的输入张量的序列。

非循环层的输出是层功能的顺序应用。例如，对于一个二输入非循环层`F(X,Y) = ⟨ f(X 0 , Y 0 ), f(X 1 , Y 1 ), f(X 2 , Y 2 )... ⟩`.如果一个张量来自循环之外，即是循环不变的，那么它的序列是通过复制张量来创建的。

10.3. Nested Loops

TensorRT 从数据流中推断出循环的嵌套。例如，如果循环 B 使用在循环 A 中定义的值，则 B 被认为嵌套在 A 中。

TensorRT 拒绝循环没有干净嵌套的网络，例如如果循环 A 使用循环 B 内部定义的值，反之亦然。

10.4. Limitations

引用多个动态维度的循环可能会占用意外的内存量。
在一个循环中，内存的分配就像所有动态维度都取任何这些维度的最大值一样。例如，如果一个循环引用两个维度为`[4,x,y]`和`[6,y]`的张量，则这些张量的内存分配就像它们的维度是`[4,max(x,y),max(x ,y)]`和`[6,max(x,y)]`。

带有`kLAST_VALUE`的`LoopOutputLayer`的输入必须是`IRecurrenceLayer`的输出。
循环 API 仅支持`FP32`和`FP16`精度。

10.5. Replacing IRNNv2Layer With Loops

`IRNNv2Layer`在 TensorRT 7.2.1 中已弃用，并将在 TensorRT 9.0 中删除。使用循环 API 合成循环子网络。例如，请参阅sampleCharRNN方法`SampleCharRNNLoop::addLSTMCell`。循环 API 让您可以表达一般的循环网络，而不是局限于IRNNLayer和IRNNv2Layer中的预制单元。

请参阅sampleCharRNN。