自动求导机制

本说明将概述Autograd如何工作并记录操作。了解这些并不是绝对必要的，但我们建议您熟悉它，因为它将帮助您编写更高效，更简洁的程序，并可帮助您进行调试。

从后向中排除子图

每个变量都有两个标志：`requires_grad`和`volatile`。它们都允许从梯度计算中精细地排除子图，并可以提高效率。

`requires_grad`

如果有一个单一的输入操作需要梯度，它的输出也需要梯度。相反，只有所有输入都不需要梯度，输出才不需要。如果其中所有的变量都不需要梯度进行，后向计算不会在子图中执行。

```
>>> x = Variable(torch.randn(5, 5))
>>> y = Variable(torch.randn(5, 5))
>>> z = Variable(torch.randn(5, 5), requires_grad=True)
>>> a = x + y
>>> a.requires_grad
False
>>> b = a + z
>>> b.requires_grad
True
```

这个标志特别有用，当您想要冻结部分模型时，或者您事先知道不会使用某些参数的梯度。例如，如果要对预先训练的CNN进行优化，只要切换冻结模型中的`requires_grad`标志就足够了，直到计算到最后一层才会保存中间缓冲区，其中的仿射变换将使用需要梯度的权重并且网络的输出也将需要它们。

```
model = torchvision.models.resnet18(pretrained=True)
for param in model.parameters():
    param.requires_grad = False
# Replace the last fully-connected layer
# Parameters of newly constructed modules have requires_grad=True by default
model.fc = nn.Linear(512, 100)

# Optimize only the classifier
optimizer = optim.SGD(model.fc.parameters(), lr=1e-2, momentum=0.9)
```

`volatile`

纯粹的inference模式下推荐使用`volatile`，当你确定你甚至不会调用`.backward()`时。它比任何其他自动求导的设置更有效——它将使用绝对最小的内存来评估模型。`volatile`也决定了`require_grad is False`。

`volatile`不同于`require_grad`的传递。如果一个操作甚至只有有一个`volatile`的输入，它的输出也将是`volatile`。`Volatility`比“不需要梯度”更容易传递——只需要一个`volatile`的输入即可得到一个`volatile`的输出，相对的，需要所有的输入“不需要梯度”才能得到不需要梯度的输出。使用volatile标志，您不需要更改模型参数的任何设置来用于inference。创建一个`volatile`的输入就够了，这将保证不会保存中间状态。

```
>>> regular_input = Variable(torch.randn(5, 5))
>>> volatile_input = Variable(torch.randn(5, 5), volatile=True)
>>> model = torchvision.models.resnet18(pretrained=True)
>>> model(regular_input).requires_grad
True
>>> model(volatile_input).requires_grad
False
>>> model(volatile_input).volatile
True
>>> model(volatile_input).creator is None
True
```

自动求导如何编码历史信息

每个变量都有一个`.creator`属性，它指向把它作为输出的函数。这是一个由`Function`对象作为节点组成的有向无环图（DAG）的入口点，它们之间的引用就是图的边。每次执行一个操作时，一个表示它的新`Function`就被实例化，它的`forward()`方法被调用，并且它输出的`Variable`的创建者被设置为这个`Function`。然后，通过跟踪从任何变量到叶节点的路径，可以重建创建数据的操作序列，并自动计算梯度。

需要注意的一点是，整个图在每次迭代时都是从头开始重新创建的，这就允许使用任意的Python控制流语句，这样可以在每次迭代时改变图的整体形状和大小。在启动训练之前不必对所有可能的路径进行编码——what you run is what you differentiate.

Variable上的In-place操作

在自动求导中支持in-place操作是一件很困难的事情，我们在大多数情况下都不鼓励使用它们。Autograd的缓冲区释放和重用非常高效，并且很少场合下in-place操作能实际上明显降低内存的使用量。除非您在内存压力很大的情况下，否则您可能永远不需要使用它们。

限制in-place操作适用性主要有两个原因：

1．覆盖梯度计算所需的值。这就是为什么变量不支持`log_`。它的梯度公式需要原始输入，而虽然通过计算反向操作可以重新创建它，但在数值上是不稳定的，并且需要额外的工作，这往往会与使用这些功能的目的相悖。

2．每个in-place操作实际上需要实现重写计算图。不合适的版本只需分配新对象并保留对旧图的引用，而in-place操作则需要将所有输入的`creator`更改为表示此操作的`Function`。这就比较棘手，特别是如果有许多变量引用相同的存储（例如通过索引或转置创建的），并且如果被修改输入的存储被任何其他`Variable`引用，则in-place函数实际上会抛出错误。

In-place正确性检查

每个变量保留有version counter，它每次都会递增，当在任何操作中被使用时。当`Function`保存任何用于后向的tensor时，还会保存其包含变量的version counter。一旦访问`self.saved_tensors`，它将被检查，如果它大于保存的值，则会引起错误。