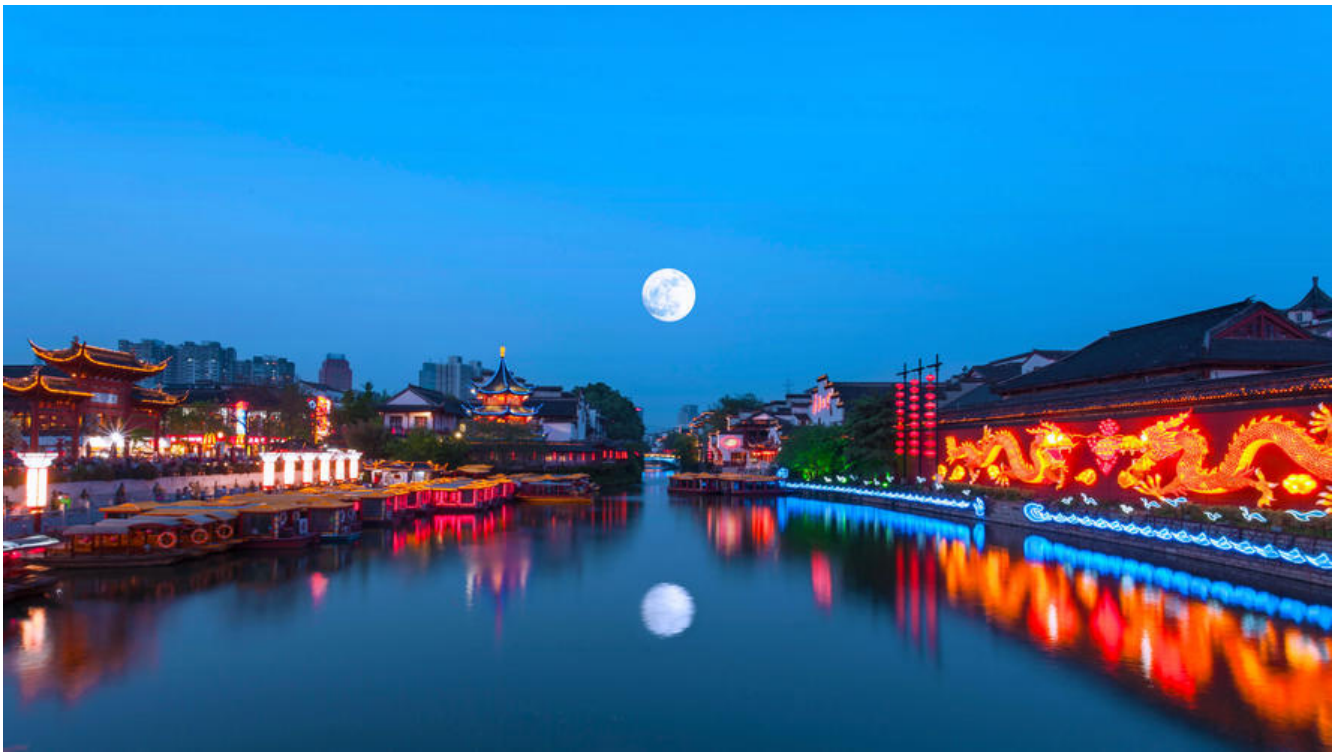
链滴

pytorch 入门笔记 -02- 自动求导

作者：zyk

原文链接：https://ld246.com/article/1637720533115

来源网站：链滴



Autograd: 自动求导机制

PyTorch 中所有神经网络的核心是 autograd 包。

我们先简单介绍一下这个包，然后训练第一个简单的神经网络。

autograd 包为张量上的所有操作提供了自动求导。

它是一个在运行时定义的框架，这意味着反向传播是根据你的代码来确定如何运行，并且每次迭代可是不同的。

示例

张量（Tensor）

torch.Tensor 是这个包的核心类。如果设置

.requires_grad 为 True，那么将会追踪所有对于该张量的操作。

当完成计算后通过调用 .backward()，自动计算所有的梯度，

这个张量的所有梯度将会自动积累到 .grad 属性。

要阻止张量跟踪历史记录，可以调用 .detach() 方法将其与计算历史记录分离，并禁止跟踪它将来的算记录。

为了防止跟踪历史记录（和使用内存），可以将代码块包装在 with torch.no_grad(): 中。

在评估模型时特别有用，因为模型可能具有 requires_grad = True 的可训练参数，但是我们不需要度计算。

在自动梯度计算中还有另外一个重要的类 Function.

Tensor and Function are interconnected and build up an acyclic

graph, that encodes a complete history of computation. Each tensor has a .grad_fn attribute that references a Function that has created the Tensor (except for Tensors created by the user - their grad_fn is None).

Tensor 和 Function互相连接并生成一个非循环图，它表示和存储了完整的计算历史。每个张量都有一个.grad_fn属性，这个属性引用了一个创建了Tensor的Function（除非这个张量是用手动创建的，即，这个张量的grad_fn 是 None）。

如果需要计算导数，你可以在 Tensor 上调用 .backward()。
如果 Tensor 是一个标量（即它包含一个元素数据）则不需要为 backward() 指定任何参数，
但是如果它有更多的元素，你需要指定一个 gradient 参数来匹配张量的形状。

```
import torch
```

创建一个张量并设置 requires_grad=True 用来追踪他的计算历史

```
x = torch.ones(2,2, requires_grad=True)
print(x)
```

```
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
```

对张量进行操作：

```
y = x + 2
print(y)
```

```
tensor([[3., 3.],
        [3., 3.]], grad_fn=<AddBackward0>)
```

结果 y 已经被计算出来了，所以 grad_fn 已经被自动生成了。

```
print(y.grad_fn)
```

```
<AddBackward0 object at 0x7fb188253280>
```

对 y 进行一个操作：

```
z = y * y * 3
out = z.mean()
print(z, out)
```

```
tensor([[27., 27.],
        [27., 27.]], grad_fn=<MulBackward0>) tensor(27., grad_fn=<MeanBackward0>)
```

.requires_grad_(...) 可以改变现有张量的 requires_grad 属性，如果未指定的话，默认输入的 flag 是 alse。

```
a = torch.randn(2, 2)
a = (a * 3) / (a - 1)
```

```
print(a.requires_grad)

a.requires_grad_(True)
print(a.requires_grad)

b = (a * a).sum()
print(b.grad_fn)
```

```
False
True
<SumBackward0 object at 0x7fb1bde05c70>
```

梯度

反向传播

因为 out 是一个纯量（scalar），out.backward() 等于 out.backward(torch.tensor(1))。

```
out.backward()
```

输出梯度 d(out)/dx：

```
print(x.grad)
```

```
tensor([[4.5000, 4.5000],
        [4.5000, 4.5000]])
```

得到矩阵 4.5。将 out 叫做 Tensor "o"。

得到 o=\frac{1}{4}\sum_i{z_i}，z_i=3(x_i+2)^2 和 z_i|_{x_i=1}=27。

因此,

\frac{\partial o}{\partial x_i} = \frac{3}{2}(x_i+2)，则

\frac{\partial o}{\partial x_i}\bigr\rvert_{x_i=1} = \frac{9}{2} = 4.5.

在数学上，如果我们有向量值函数 \vec{y} = f(\vec{x}) ，且 \vec{y} 关于 \vec{x} 的梯度是一个雅可矩阵(Jacobian matrix)：

J = \begin{pmatrix} \frac{\partial y_{1}}{\partial x_{1}} & \cdots & \frac{\partial y_{1}}{\partial x_n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_{m}}{\partial x_{1}} & \cdots & \frac{\partial y_{m}}{\partial x_{n}} \end{pmatrix}

一般来说，torch.autograd 就是用来计算 vector-Jacobian product 的工具。也就是说，给定任一量 v=(v_{1}\;v_{2}\;\cdots\;v_{m})^{T} ，计算 v^{T}\cdot J ，如果 v 恰好是标量函数 l=g(\vec{y}) 梯度，也就是说 v=(\frac{\partial l}{\partial y_{1}}\;\cdots\;\frac{\partial l}{\partial y_{m}})^{T} 那么根据链式法则，vector-Jacobian product 是 l 关于 \vec{x} 的梯度：

J^{T}\cdot v = \begin{pmatrix} \frac{\partial y_{1}}{\partial x_{1}} & \cdots & \frac{\partial y_{m}}{\partial x_{1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_{1}}{\partial x_{n}} & \cdots & \frac{\partial y_{m}}{\partial x_{n}} \end{pmatrix} \begin{pmatrix} \frac{\partial l}{\partial y_{1}}\\ \vots \\ \frac{\partial l}{\partial y_{m}} \end{pmatrix} = \begin{pmatrix} \frac{\partial l}{\partial x_1}}\\ \vdots \\ \frac{\partial l}{\partial x_{n}} \end{pmatrix}

（注意，v^{T}\cdot J 给出了一个行向量，可以通过 J^{T}\cdot v 将其视为列向量）

vector-Jacobian product 这种特性使得将外部梯度返回到具有非标量输出的模型变得非常方便。

现在让我们来看一个vector-Jacobian product的例子

```
x = torch.randn(3, requires_grad=True)

y = x * 2
while y.data.norm() < 1000:
    y = y * 2

print(y)
```

```
tensor([1810.9065,  344.1436,  -78.7548], grad_fn=<MulBackward0>)
```

在这个情形中，y 不再是个标量。torch.autograd 无法直接计算出完整的雅可比行列，但是如果我们想要 vector-Jacobian product，只需将向量作为参数传入 backward：

```
gradients = torch.tensor([0.1, 1.0, 0.0001], dtype=torch.float)
y.backward(gradients)

print(x.grad)
```

```
tensor([2.0480e+02, 2.0480e+03, 2.0480e-01])
```

如果 .requires_grad=True 但是你又不希望进行autograd的计算，

那么可以将变量包裹在 with torch.no_grad() 中:

```
print(x.requires_grad)
print((x ** 2).requires_grad)

with torch.no_grad():
    print((x ** 2).requires_grad)
```

```
True
True
False
```