PyTorch自动求导

作者：凯鲁嘎吉 - 博客园 http://www.cnblogs.com/kailugaji/ (http://www.cnblogs.com/kailugaji/)

所用版本：python 3.6.5，torch 1.6.0，torchvision 0.7.0

假设我们想对函数$y=2\mathbf{x}^{\top}\mathbf{x}$关于列向量$\mathbf{x}$求导。
首先，我们创建变量x并为其分配一个初始值。

```
In [1]: import torch
```

```
In [2]: x = torch.arange(4.0)
```

```
In [3]: x
Out[3]: tensor([0., 1., 2., 3.])
```

在计算y关于$\mathbf{x}$的梯度之前，需要一个地方来存储梯度。
重要的是，我们不会在每次对一个参数求导时都分配新的内存。因为我们经常会成千上万次地更新相同的参数，每次都分配新的内存可能很快就会将内存耗尽。注意，标量函数关于向量$\mathbf{x}$的梯度是向量，并且与$\mathbf{x}$具有相同的形状。

```
In [4]: x.requires_grad_(True)  # 等价于 `x = torch.arange(4.0, requires_grad=True)`
Out[4]: tensor([0., 1., 2., 3.], requires_grad=True)
```

In [5]:
```
x.grad  # 默认值是None
```

现在计算y

In [6]:
```
y = 2 * torch.dot(x, x)
```

In [7]:
```
y
```

Out[7]:
```
tensor(28., grad_fn=<MulBackward0>)
```

x是一个长度为4的向量，计算x和x的内积，得到了我们赋值给y的标量输出。接下来，可以通过调用反向传播函数来自动计算y关于x每个分量的梯度，并打印这些梯度。

In [8]:
```
y.backward()
```

In [9]:
```
x.grad
```

Out[9]:
```
tensor([ 0.,  4.,  8., 12.])
```

函数$y=2\mathbf{x}^{\top}\mathbf{x}$关于$\mathbf{x}$的梯度应为$4\mathbf{x}$。让我们快速验证我们想要的梯度是否正确计算。

In [10]:
```
x.grad == 4 * x
```

Out[10]:
```
tensor([True, True, True, True])
```

现在让我们计算x的另一个函数

In [11]:
```
# 在默认情况下，PyTorch会累积梯度，我们需要清除之前的值
```

In [12]:
```
x.grad.zero_()
```

Out[12]: tensor([0., 0., 0., 0.])

In [13]:
```
# y=sum(x)
```

In [14]:
```
y = x.sum()
```

In [15]:
```
y.backward()
```

In [16]:
```
x.grad
```

Out[16]: tensor([1., 1., 1., 1.])

In [17]:
```
# y=x的导数为1，因此计算出来是常数1
```

非标量变量的反向传播

In [18]:
```
# 对非标量调用`backward`需要传入一个`gradient`参数，该参数指定微分函数关于`self`的梯度。
# 在我们的例子中，我们只想求偏导数的和，所以传递一个1的梯度是合适的
```

In [19]:
```
x.grad.zero_()
```

Out[19]: tensor([0., 0., 0., 0.])

In [20]:
```
y = x * x
```

In [21]:
```
# 等价于y.backward(torch.ones(len(x)))
```

In [22]:
```
y.sum().backward()
```

In [23]:
```
x.grad
```

Out[23]: tensor([0., 2., 4., 6.])

In [24]:
```
# y=x^2的导数是2x
```

In [25]:
```
x.grad == 2 * x
```

Out[25]:
```
tensor([True, True, True, True])
```

分离计算

$z=x^3$，看成两部分，首先$y=x^2$，然后$z=yx$

In [26]:
```
x.grad.zero_()
```

Out[26]:
```
tensor([0., 0., 0., 0.])
```

In [27]:
```
y = x * x
```

在这里，我们可以分离y来返回一个新变量u，该变量与y具有相同的值，但丢弃计算图中如何计算y的任何信息。
换句话说，梯度不会向后流经u到x。因此，下面的反向传播函数计算z=u*x关于x的偏导数，同时将u作为常数处理，而不是z=x*x*x关于x的偏导数。

In [28]:
```
u = y.detach()
```

In [29]:
```
z = u * x
```

In [30]:
```
z.sum().backward()
```

In [31]:
```
x.grad == u
```

Out[31]:
```
tensor([True, True, True, True])
```

由于记录了y的计算结果，我们可以随后在y上调用反向传播，得到y=x*x关于的x的导数，这里是2*x。

In [32]:
```
x.grad.zero_()
```

Out[32]:
```
tensor([0., 0., 0., 0.])
```

In [33]:
```
y.sum().backward()
```

In [34]:
```
x.grad == 2 * x
```

Out[34]:
```
tensor([True, True, True, True])
```

Python控制流的梯度计算

In [35]:
```
def f(a):
    b = a * 2
    while b.norm() < 1000:
        b = b * 2
    if b.sum() > 0:
        c = b
    else:
        c = 100 * b
    return c
```

In [36]:
```
a = torch.randn(size=(), requires_grad=True)
```

In [37]:
```
d = f(a)
```

In [38]:
```
d.backward()
```

我们现在可以分析上面定义的f函数。请注意，它在其输入a中是分段线性的。换言之，对于任何a，存在某个常量标量k，使得f(a)=k*a，其中k的值取决于输入a。因此，d/a允许我们验证梯度是否正确。

In [39]:
```
a.grad == d / a
```

Out[39]:
```
tensor(True)
```

例子： 使$f(x)=\sin(x)$，绘制$f(x)$和$\frac{df(x)}{dx}$的图像，其中后者不使用$f'(x)=\cos(x)$。

In [40]:
```
import matplotlib.pyplot as plt
```

In [41]:
```
import torch
```

In [42]:
```
x = torch.arange(0.0,10,0.1)
```

In [43]:
```
x.requires_grad_(True)
```

Out[43]:
```
tensor([0.0000, 0.1000, 0.2000, 0.3000, 0.4000, 0.5000, 0.6000, 0.7000, 0.8000,
        0.9000, 1.0000, 1.1000, 1.2000, 1.3000, 1.4000, 1.5000, 1.6000, 1.7000,
        1.8000, 1.9000, 2.0000, 2.1000, 2.2000, 2.3000, 2.4000, 2.5000, 2.6000,
        2.7000, 2.8000, 2.9000, 3.0000, 3.1000, 3.2000, 3.3000, 3.4000, 3.5000,
        3.6000, 3.7000, 3.8000, 3.9000, 4.0000, 4.1000, 4.2000, 4.3000, 4.4000,
        4.5000, 4.6000, 4.7000, 4.8000, 4.9000, 5.0000, 5.1000, 5.2000, 5.3000,
        5.4000, 5.5000, 5.6000, 5.7000, 5.8000, 5.9000, 6.0000, 6.1000, 6.2000,
        6.3000, 6.4000, 6.5000, 6.6000, 6.7000, 6.8000, 6.9000, 7.0000, 7.1000,
        7.2000, 7.3000, 7.4000, 7.5000, 7.6000, 7.7000, 7.8000, 7.9000, 8.0000,
        8.1000, 8.2000, 8.3000, 8.4000, 8.5000, 8.6000, 8.7000, 8.8000, 8.9000,
        9.0000, 9.1000, 9.2000, 9.3000, 9.4000, 9.5000, 9.6000, 9.7000, 9.8000,
        9.9000], requires_grad=True)
```

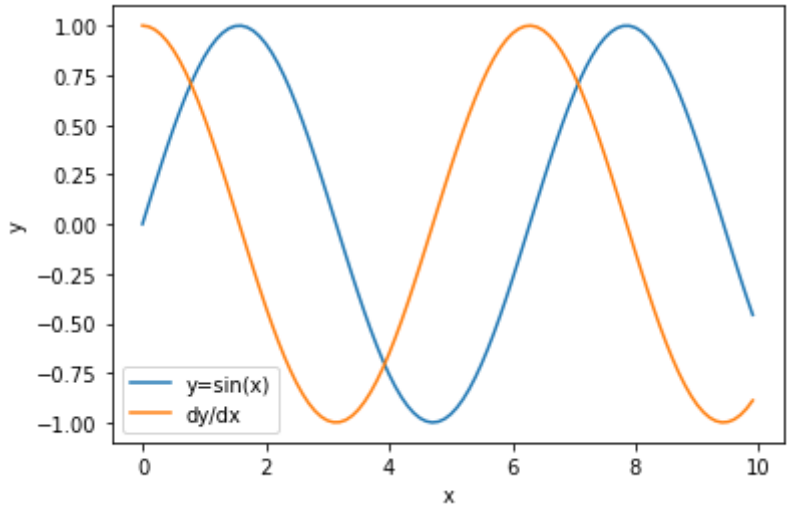
In [44]:
```
x1 = x.detach()
```

In [45]:
```
y1 = torch.sin(x1)
```

In [46]:
```
y2 = torch.sin(x)
```

In [47]:
```
y2.sum().backward()
```

In [48]:
```
plt.plot(x1, y1, label='y=sin(x)')
plt.plot(x1, x.grad, label='dy/dx')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



参考：《动手深度学习》 https://zh-v2.d2l.ai/chapter_preliminaries/autograd.html (https://zh-v2.d2l.ai/chapter_preliminaries/autograd.html)