Back propagation

Basic idea

反向传播（Back propagation）是一种高效的计算梯度的方法，特别适用于神经网络中的参数优化。在机器学习模型（尤其是神经网络）中，反向传播用于计算损失函数相对于模型参数的梯度，从而可以通过梯度下降法来更新参数。

假设有一个函数 y=2x，我们很容易计算出梯度：∂y / ∂x=2

但是在神经网络中，情况通常更复杂。神经网络由多个层组成，每一层都有自己的参数（例如权重和偏置）。当我们给网络输入数据并得到输出时，最终的输出（通常是损失）不仅依赖于输入，还依赖于网络的所有参数。

反向传播的作用是通过链式法则（链式求导）来计算损失函数对每个参数的梯度。这些梯度告诉我们如何调整每个参数，以减少最终的损失。

Chain rule

$$y = 2x \qquad L = (y-t)^2 \qquad \frac{\partial y}{\partial x} = 2$$

$$\frac{\partial L}{\partial y} = 2(y-t)$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x} = 2(y-t) \cdot 2 = 4(y-t)$$

Backpropagation Effects

在反向传播之后，模型会得到每个参数的梯度。这些梯度告诉我们，如果要减小损失，应该如何调整参数。在梯度下降法中，我们会通过这些梯度来更新参数，使模型的预测更接近目标值。

简而言之，反向传播是通过计算梯度来指导模型的参数调整，以使模型的输出更接近目标，从而提高模型的性能。

Code

```
import torch
x = torch.arange(4.0)
print('x:',x)

x.requires_grad_(True) # Equivalent to x = torch.arange(4.0,requires_grad=True)
print('x.grad_1:',x.grad) # x.grad是存梯度的地方，默认为None

y = 2 * torch.dot(x,x) # y = 2（x^2） = 2(0+1+4+9) = 28
print('y_2:',y)
y.backward() # After this line, y.grad = 2x, where x is the input tensor
print('x.grad_2:',x.grad) # Accessing the derivative, accessing the gradient,  y=2*
（x^2）, y'=4x，so,x.grad=4x

x.grad.zero_() # dereference the gradient
y = x.sum()
```

```
print('y_3:',y) # 这里的y是一个标量，0+1+2+3=6
y.backward()
print('x.grad_3:',x.grad) # Accessing the derivative, accessing the gradient,
y=x1+x2+x3......，y'=1，so，x.grad=1

y.backward()
x.grad.zero_()
y = x * x
print('y_4:',y) # 这里的y是一个向量，y=x^2，y'=2x，所以x.grad=2x
y.sum().backward()
print('x.grad_4:',x.grad) # y=x^2，y'=2x，so，x.grad=2x

x.grad.zero_()
y = x * x
print('y_5:',y)
u = y.detach() # detach函数将y当作一个常数，而不是关于x的一个函数
print('y.detach():',y.detach())
print('u:',u)
z = u * x
print('z:',z)
z.sum().backward()
print(x.grad == u) # 这里的x.grad是关于u的函数，所以x.grad(u) = u，即x.grad(u) = u(x) =
u(2x) = 4x，所以x.grad(u) = 4x

def f(a):
    b = a * 2
    while b.norm() < 1000: # norm函数计算向量的模
        b = b * 2
    if b.sum() > 0:
        c = b
    else:
        c = 100 * b
    return c

a = torch.randn(size=(),requires_grad=True)
print('a:',a)
d = f(a)
d.backward()
print('a.grad:',a.grad)
print('d/a:',d/a)
print(a.grad == d/a) # d是a的线性函数，所以导数就是斜率d/a

'''
为什么d是关于a的线性函数呢？因为b是关于a的线性函数，c是关于b的线性函数。
比如说b要进行n次乘2模才能大于1000，则b = 2 * a *n
如果坐标相加大于0则d = c = 2 * a *n，反之d = 200 * a *n
所以d是关于a的线性函数
'''
```