AUTOMATIC DIFFERENTIATION

一、内容

当训练神经网络时，最常用的算法是反向传播。在这个算法中，根据损失函数对给定参数的梯度，调整参数（模型权重）的值。为了计算这些梯度，PyTorch有一个内置的微分引擎，叫做torch.autograd。它支持对任何计算图自动计算梯度。用来构建计算图的张量上应用的函数实际上是Function类的一个对象。这个对象知道如何在前向方向计算函数，也知道如何在反向传播步骤中计算其导数。对反向传播函数的引用存储在张量的grad_fn属性中。

二、代码

1、backward

> 可以在创建张量时设置requires_grad的值，或者稍后使用x.requires_grad_(True)方法。

> 为了计算这些导数，调用loss.backward()，然后从w.grad和b.grad中获取值。

```
import torch

x = torch.ones(5)  # input tensor
y = torch.zeros(3)  # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss = torch.nn.functional.binary_cross_entropy_with_logits(z, y)

loss.backward()
print(w.grad)
print(b.grad)
```

```
tensor([[0.3185, 0.1895, 0.3048],
        [0.3185, 0.1895, 0.3048],
        [0.3185, 0.1895, 0.3048],
        [0.3185, 0.1895, 0.3048],
        [0.3185, 0.1895, 0.3048]])
tensor([0.3185, 0.1895, 0.3048])
```

2、torch.no_grad() & x.detach()

默认情况下，所有设置了requires_grad=True的张量都会跟踪它们的计算历史并支持梯度计算。但是，有些情况下不需要这样做，例如，当已经训练好了模型，只想将其应用到一些输入数据上，即只想做网络的前向计算。可以通过用torch.no_grad()块包围我们的计算代码来停止跟踪计算：

```
z = torch.matmul(x, w)+b
print(z.requires_grad)

with torch.no_grad():
      z = torch.matmul(x, w)+b
print(z.requires_grad)
print(b.requires_grad)
print(w.requires_grad)
```

```
True
False
True
True
```

```
z = torch.matmul(x, w)+b
z_det = z.detach()
print(z_det.requires_grad)
```

> 达到同样效果的另一种方法是使用张量的detach()方法

好处：

- 标记神经网络中的一些参数为冻结参数（不希望这些参数发生改变）。
- 只进行前向传递的计算，因为不跟踪梯度的张量的计算会更高效。

3、retain_graph=True

$$c = \frac{\sum_i^4 a_i^2}{4}$$

$$d = \sum_i^4 a_i^2$$

$$\frac{\partial c}{\partial a_i} = \frac{a_i}{2}$$

$$\frac{\partial d}{\partial a_i} = 2a_i$$

```
a = torch.tensor([1.,2.,3.,4],requires_grad=True)
b = a ** 2
c = b.mean()
d = b.sum()

c.backward()
print(a.grad)
d.backward()
print(a.grad)
```

这样运行会报错，因为执行完c.backward()，计算图就释放了，d.backward()会报错。

```
a = torch.tensor([1.,2.,3.,4],requires_grad=True)
b = a ** 2
c = b.mean()
d = b.sum()

c.backward(retain_graph=True)
print(a.grad)
d.backward()
print(a.grad)
```

```
tensor([0.5000, 1.0000, 1.5000, 2.0000])
tensor([ 2.5000,  5.0000,  7.5000, 10.0000])
```

这里使用了retain_graph=True，即保留了计算图，并且d.backward()计算后的梯度值将与c.backward计算的梯度相加。

a=[1.,2.,3.,4]，c = a/2， d = a/2 + 2a.

4、In-place operation

> tensor类型不要进行原位操作，在进行backward时会报错。