扩展阅读

`x` 张量中有两个元素，记作 x_1 与 x_2，`y` 张量中的两个元素记作 y_1 与 y_2，并且两者的关系是：

$$\mathbf{x} = [x_1, x_2]$$

$$\mathbf{y} = [y_1, y_2] = [3x_1 + 1, 3x_2 + 1]$$

此时，想直接求 $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{[3x_1 + 1, 3x_2 + 1]}{[x_1, x_2]}$$

在数学上是没有意义的，因此当然就报错了。实际上，当用户调用 `y.backward()` 时，其实想要的结果通常是：

$$[\frac{\partial y_1}{\partial x_1}, \frac{\partial y_2}{\partial x_2}]$$

当对 `y` 进行 `sum` 运算后：

$$y = y_1 + y_2 = 3x_1 + 3x_2 + 2$$

此时，调用 `backward()` 时，对 x_1 和 x_2 可求梯度：

$$\frac{\partial y}{\partial x_1} = \frac{\partial 3x_1 + 3x_2 + 2}{\partial x_1} = 3$$

$$\frac{\partial y}{\partial x_2} = \frac{\partial 3x_1 + 3x_2 + 2}{\partial x_2} = 3$$

除了使用 `sum` 之外，还可以使用更通用方法，即 **Vector Jacobian Product(VJP)** 完成非标量的根节点的梯度计算。依然用上文的例子，在反向传播过程中，OneFlow 会根据计算图生成雅可比矩阵：

$$J = \begin{pmatrix} \frac{\partial y1}{\partial x1} & \frac{\partial y1}{\partial x2} \\ \frac{\partial y2}{\partial x1} & \frac{\partial y2}{\partial x2} \end{pmatrix} = \begin{pmatrix} \frac{\partial y1}{\partial x1} & 0 \\ 0 & \frac{\partial y2}{\partial x2} \end{pmatrix}$$

只需提供一个与 $\mathbf{y}$ 大小一致的向量 $\mathbf{v}$，即可计算 VJP：

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \times \begin{pmatrix} \frac{\partial y1}{\partial x1} & 0 \\ 0 & \frac{\partial y2}{\partial x2} \end{pmatrix} = \begin{bmatrix} v_1 \frac{\partial y1}{\partial x1} \\ v_2 \frac{\partial y2}{\partial x2} \end{bmatrix}$$

若向量 $\mathbf{v}$ 是反向传播中上一层的梯度，VJP 的结果刚好是当前层要求的梯度。

`backward` 方法是可以接受一个张量做参数的，该参数就是 VJP 中的 $\mathbf{v}$，理解以上道理后，还可以使用以下的方式对张量求梯度：

```
x = flow.randn(1, 2, requires_grad=True)
y = 3*x + 1
y.backward(flow.ones_like(y))
print(x.grad)
```

输出：

```
tensor([[3., 3.]], dtype=oneflow.float32)
```

外部链接

- Automatic Differentiation