深度学习之自动微分

荣耀学院

2022 年 07 月

第一版序言

自动微分是深度学习的基石之一。自动微分实现了深度学习的梯度计算的自动化，节省了大量推导时间。

本书致力于让所有人都能完全掌握自动微分。

本书最佳使用方式：拿出纸和笔，亲自推导所有公式，运行所有的代码。为了便于理解和学习，本书所有的推导给出了所有步骤，不做任何省略。

目 录

第一章　起源

一个神经网络，无论有多大，无论有多复杂，只是表达一个函数。

一个简单的三层前馈神网络如此。一个十几亿参数的深度学习模型也是如此。

这个函数的参数是权重 (Weight) 和偏差 (Bias)。神经网络越复杂，权重和偏差越多。用 w 表示权重，b 表示偏差，这个函数可以表示为：

$$f_{w_0,w_1,...,x_i,...,b_0,b_1,...,b_j,...}(\cdot)$$

用 $\boldsymbol{w}$ 表示所有的 w_i，用 $\boldsymbol{b}$ 表示所有的 b_j，上式又可以写成：

$$f_{\boldsymbol{w},\boldsymbol{b}}(\cdot)$$

这个函数，有一个或者多个输入，一个或者多个输出。

用 x 表示输入，y 表示输出，这个函数可以表示为：

$$y_0, y_1, ..., y_k, ... = f_{\boldsymbol{w},\boldsymbol{b}}(x_0, x_1, ..., x_l, ...)$$

用 $\boldsymbol{y}$ 表示所有的 y_k，用 $\boldsymbol{x}$ 表示所有的 x_l，上式又可以写成：

$$\boldsymbol{y} = f_{\boldsymbol{w},\boldsymbol{b}}(\boldsymbol{x})$$

神经网络的训练，本质上就是“调整”这个函数的 $\boldsymbol{w}$ 和 $\boldsymbol{b}$，使得输入是 $\boldsymbol{x}$ 的时候，输出非常接近 $\boldsymbol{y}$。

“调整”的方式，就是微分求极值，也就需要推导目标函数对 $\boldsymbol{w}$ 和 $\boldsymbol{b}$ 的一阶偏导。

如果这个函数比较简单，可以手工推导微分。如果函数很复杂，手工推导也就太麻烦了，需要一种自动的方式。深度学习模型很复杂，需要自动微分。

第二章　前向推导微分的例子

先用最简单的例子研究微分问题。

第一个例子:

$$y = x_0 + x_1$$

$$\begin{aligned}
\frac{\partial y}{\partial x_0} &= \frac{\partial(x_0 + x_1)}{\partial x_0} \\
&= \frac{\partial x_0}{\partial x_0} + \frac{\partial x_1}{\partial x_0} \\
&= 1 + 0 \\
&= 1
\end{aligned}$$

$$\begin{aligned}
\frac{\partial y}{\partial x_1} &= \frac{\partial(x_0 + x_1)}{\partial x_1} \\
&= \frac{\partial x_0}{\partial x_1} + \frac{\partial x_1}{\partial x_1} \\
&= 0 + 1 \\
&= 1
\end{aligned}$$

第二个例子:

$$y = 2x_0 + 3x_1$$

$$\begin{aligned}\frac{\partial y}{\partial x_0} &= \frac{\partial(2x_0+3x_1)}{\partial x_0}\\ &= \frac{\partial(2x_0)}{\partial x_0}+\frac{\partial(3x_1)}{\partial x_0}\\ &= 2+0\\ &= 2\end{aligned}$$

$$\begin{aligned}\frac{\partial y}{\partial x_1} &= \frac{\partial(2x_0+3x_1)}{\partial x_1}\\ &= \frac{\partial(2x_0)}{\partial x_1}+\frac{\partial(3x_1)}{\partial x_1}\\ &= 0+3\\ &= 3\end{aligned}$$

在这个例子，计算 y 需要做两次乘法和一次加法。一个函数的复杂度是不定的，如果要做自动微分，肯定不能一次性地对复杂度不定的函数做微分。要拆分，拆成每次做一个运算，然后再组合。因此，可以把这个函数调整成：

$$v_0 = 2x_0$$

$$v_1 = 3x_1$$

$$y = v_0 + v_1$$

假如要计算 y 对 x_0 的一阶偏导，根据上面三个公式的次序，可以依次按照如下步骤计算：

$$\dot{v_0}|_{x_0} = \frac{\partial v_0}{\partial x_0} = 2$$

$$\dot{v_1}|_{x_0} = \frac{\partial v_1}{\partial x_0} = 0$$

$$\begin{aligned}
\dot{y}|_{x_0} &= \frac{\partial(v_0+v_1)}{\partial x_0} \\
&= \frac{\partial v_0}{\partial x_0}+\frac{\partial v_1}{\partial x_0} \\
&= \dot{v_0}|_{x_0}+\dot{v_1}|_{x_0} \\
&= 2+0 \\
&= 2
\end{aligned}$$

假如要计算 y 对 x_1 的一阶偏导，可以依次按照如下步骤计算：

$$\dot{v_0}|_{x_1}=\frac{\partial v_0}{\partial x_1}=0$$

$$\dot{v_1}|_{x_1}=\frac{\partial v_1}{\partial x_1}=3$$

$$\begin{aligned}
\dot{y}|_{x_1} &= \frac{\partial(v_0+v_1)}{\partial x_1} \\
&= \frac{\partial v_0}{\partial x_1}+\frac{\partial v_1}{\partial x_1} \\
&= \dot{v_0}|_{x_1}+\dot{v_1}|_{x_1} \\
&= 0+3 \\
&= 3
\end{aligned}$$

第三个例子：

调整一下 y 的值：

$$v_0=2x_0$$

$$v_1=3x_1$$

$$y=\frac{v_0}{v_1}$$

假如要计算 y 对 x_0 的一阶偏导，根据上面三个公式的次序，可以依次按照如下步骤计算：

$$\dot{v_0}|_{x_0} = \frac{\partial v_0}{\partial x_0} = 2$$

$$\dot{v_1}|_{x_0} = \frac{\partial v_1}{\partial x_0} = 0$$

$$\begin{aligned}
\dot{y}|_{x_0} &= \frac{\partial(\frac{v_0}{v_1})}{\partial x_0} \\
&= \frac{\partial v_0}{\partial x_0}\frac{1}{v_1} + v_0(-1)\frac{1}{v_1^2}\frac{\partial v_1}{\partial x_0} \\
&= \dot{v_0}|_{x_0}\frac{1}{v_1} - \frac{v_0}{v_1^2}\frac{\partial v_1}{\partial x_0} \\
&= \dot{v_0}|_{x_0}\frac{1}{v_1} - \frac{v_0}{v_1^2}\dot{v_1}|_{x_0} \\
&= \frac{2}{3x_1} - \frac{v_0}{v_1^2}\cdot 0 \\
&= \frac{2}{3x_1}
\end{aligned}$$

假如要计算 y 对 x_1 的一阶偏导，可以依次按照如下步骤计算：

$$\dot{v_0}|_{x_1} = \frac{\partial v_0}{\partial x_1} = 0$$

$$\dot{v_1}|_{x_1} = \frac{\partial v_1}{\partial x_1} = 3$$

$$\begin{aligned}
\dot{y}|_{x_1} &= \frac{\partial(\frac{v_0}{v_1})}{\partial x_1} \\
&= \frac{\partial v_0}{\partial x_1}\frac{1}{v_1} + v_0(-1)\frac{1}{v_1^2}\frac{\partial v_1}{\partial x_1} \\
&= \dot{v_0}|_{x_1}\frac{1}{v_1} - \frac{v_0}{v_1^2}\frac{\partial v_1}{\partial x_1} \\
&= \dot{v_0}|_{x_1}\frac{1}{v_1} - \frac{v_0}{v_1^2}\dot{v_1}|_{x_0} \\
&= 0 - \frac{v_0}{v_1^2}\cdot 3 \\
&= -\frac{2x_0}{9x_1^2}\cdot 3 \\
&= -\frac{2x_0}{3x_1^2}
\end{aligned}$$

用前向方式推导微分，有啥问题？重复！x_0 推一遍，x_1 推一遍。如果有几亿个 x_i，要推导几亿次。

能省下来吗?

可以。

怎么做?

反向推导微分。

第三章　反向推导微分的例子

第一个例子：

$$v_0 = 2x_0$$

$$v_1 = 3x_1$$

$$y = v_0 + v_1$$

反向推导，对上面三个公式，按照从下向上推导。先推导 y 对 v_0 和 v_1 的一阶偏导。再推导 v_1 和 v_0 分别对 x_1 和 x_0 的一阶偏导。按照这个次序：

$$\begin{aligned}
\dot{y}|_{v_0} &= \frac{\partial y}{\partial v_0} \\
&= \frac{\partial (v_0 + v_1)}{\partial v_0} \\
&= \frac{\partial v_0}{\partial v_0} + \frac{\partial v_1}{\partial v_0} \\
&= 1 + 0 \\
&= 1
\end{aligned}$$

$$\begin{aligned}
\dot{y}|_{v_1} &= \frac{\partial y}{\partial v_1} \\
&= \frac{\partial (v_0 + v_1)}{\partial v_1} \\
&= \frac{\partial v_0}{\partial v_1} + \frac{\partial v_1}{\partial v_1} \\
&= 0 + 1 \\
&= 1
\end{aligned}$$

$$
\begin{aligned}
\dot{y}|_{x_0} &= \frac{\partial y}{\partial v_0}\frac{\partial v_0}{\partial x_0} + \frac{\partial y}{\partial v_1}\frac{\partial v_1}{\partial x_0} \\
&= \dot{y}|_{v_0}\frac{\partial v_0}{\partial x_0} + \dot{y}|_{v_1}\frac{\partial v_1}{\partial x_0} \\
&= 1 \cdot \frac{\partial v_0}{\partial x_0} + 1 \cdot \frac{\partial v_1}{\partial x_0} \\
&= \frac{\partial v_0}{\partial x_0} + \frac{\partial v_1}{\partial x_0} \\
&= 2 + 0 \\
&= 2
\end{aligned}
$$

$$
\begin{aligned}
\dot{y}|_{x_1} &= \frac{\partial y}{\partial v_0}\frac{\partial v_0}{\partial x_1} + \frac{\partial y}{\partial v_1}\frac{\partial v_1}{\partial x_1} \\
&= \dot{y}|_{v_0}\frac{\partial v_0}{\partial x_1} + \dot{y}|_{v_1}\frac{\partial v_1}{\partial x_1} \\
&= 1 \cdot \frac{\partial v_0}{\partial x_1} + 1 \cdot \frac{\partial v_1}{\partial x_1} \\
&= \frac{\partial v_0}{\partial x_1} + \frac{\partial v_1}{\partial x_1} \\
&= 0 + 3 \\
&= 3
\end{aligned}
$$

看，总推导次数变少了。上一章推导 $\dot{y}|_{x_0}$ 和 $\dot{y}|_{x_1}$ 一共需要做 6 次推导，现在是 4 次。

第二个例子：

$$v_0 = 2x_0$$

$$v_1 = 3x_1$$

$$y = \frac{v_0}{v_1}$$

反向推导，按照这个次序：

$$\begin{aligned}
\dot{y}|_{v_0} &= \frac{\partial y}{\partial v_0} \\
&= \frac{\partial(\frac{v_0}{v_1})}{\partial v_0} \\
&= \frac{1}{v_1}
\end{aligned}$$

$$\begin{aligned}
\dot{y}|_{v_1} &= \frac{\partial y}{\partial v_1} \\
&= \frac{\partial(\frac{v_0}{v_1})}{\partial v_1} \\
&= v_0(-1)\frac{1}{v_1^2} \\
&= -\frac{v_0}{v_1^2}
\end{aligned}$$

$$\begin{aligned}
\dot{y}|_{x_0} &= \frac{\partial y}{\partial v_0}\frac{\partial v_0}{\partial x_0} + \frac{\partial y}{\partial v_1}\frac{\partial v_1}{\partial x_0} \\
&= \dot{y}|_{v_0}\frac{\partial v_0}{\partial x_0} + \dot{y}|_{v_1}\frac{\partial v_1}{\partial x_0} \\
&= \frac{1}{v_1} \cdot 2 - \frac{v_0}{v_1^2} \cdot 0 \\
&= \frac{2}{3x_1}
\end{aligned}$$

$$\begin{aligned}
\dot{y}|_{x_1} &= \frac{\partial y}{\partial v_0}\frac{\partial v_0}{\partial x_1} + \frac{\partial y}{\partial v_1}\frac{\partial v_1}{\partial x_1} \\
&= \dot{y}|_{v_0}\frac{\partial v_0}{\partial x_1} + \dot{y}|_{v_1}\frac{\partial v_1}{\partial x_1} \\
&= \frac{1}{v_1} \cdot 0 - \frac{v_0}{v_1^2} \cdot 3 \\
&= -\frac{2x_0}{3x_1^2}
\end{aligned}$$

第四章　计算图

计算图，没有任何神奇，只是把函数的计算过程用图形表示出来，更直观了。

比如：

$$v_0 = x_0$$

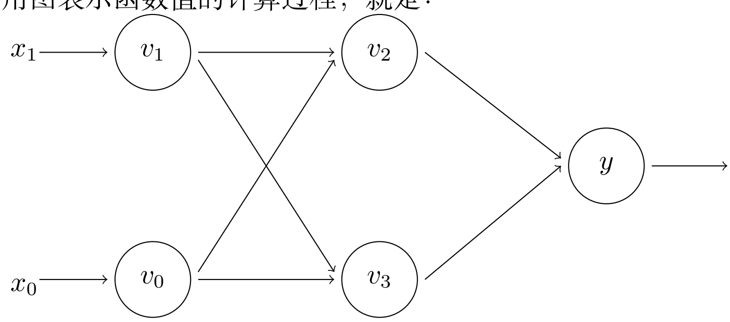
$$v_1 = x_1$$

$$v_2 = v_0 v_1$$

$$v_3 = \frac{v_1}{v_0}$$

$$y = v_2 + v_3$$

用图表示函数值的计算过程，就是：



设 $x_0 = 3$，$x_1 = 5$，依次求值结果：

$$v_0 = x_0 = 3$$

$$v_1 = x_1 = 5$$

$$v_2 = v_0 v_1 = 3 \cdot 5 = 15$$

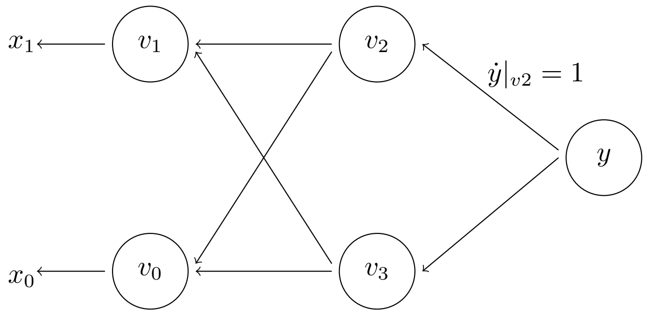
$$v_3 = \frac{v_1}{v_0} = \frac{5}{3}$$

$$y = v_2 + v_3 = 16\frac{2}{3}$$

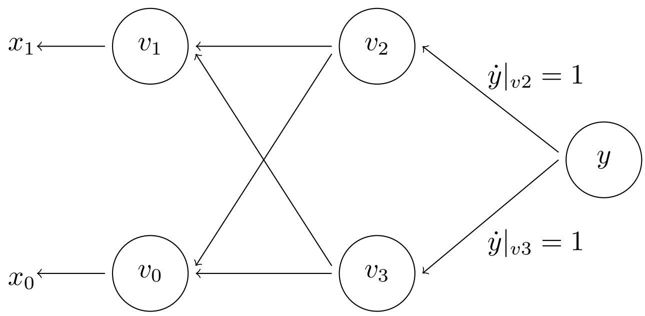
此时，$x_0 = 3$，$x_1 = 5$，求 y 对 x_0 和 x_1 的一阶偏导的值。

用反向推导在图上求解：

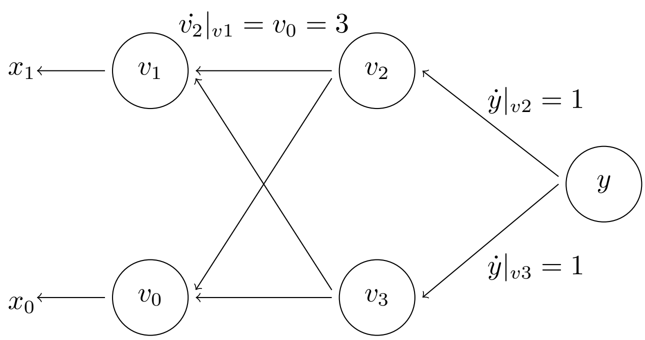
1) 求 y 对 v_2 的一阶偏导：$\dot{y}|_{v2} = 1$，在图上标出来：



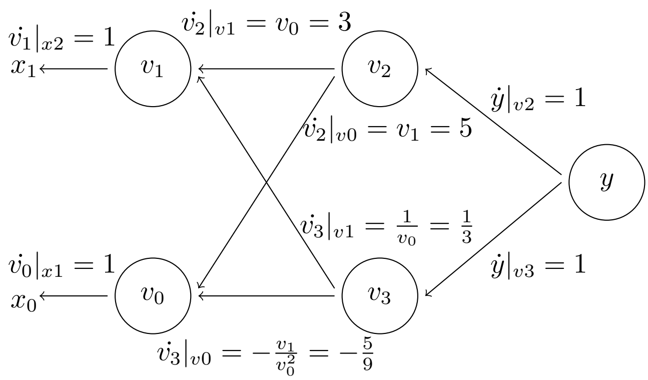
2) 求 y 对 v_3 的一阶偏导：$\dot{y}|_{v2} = 1$，在图上标出来：



3) 求 v_2 对 $_1$ 的一阶偏导：$\dot{v_2}|_{v1} = v_0 = 3$，在图上标出来：



4) 依次求出后续所有偏导，都在图上标出来，结果如下：



在图上，假如给 v_2 节点增加一个变量 $grad$ 记录它的梯度，那么可以表示为 $v_2.grad = \dot{y}|_{v_2} = 1$。

同理，对 v_3 也进行这个操作，$v_3.grad = \dot{y}|_{v_3} = 1$

注意，到关键点了，$v_1.grad$ 是什么?

$v_1.grad$ 的本质是:

$$v_1.grad = \dot{y}|_{v_1} = \frac{\partial y}{\partial v_1}$$

因为

$$\begin{aligned}
\dot{y}|_{v_1} &= \frac{\partial y}{\partial v_1} \\
&= \frac{\partial y}{\partial v_2}\frac{\partial v_2}{\partial v_1} + \frac{\partial y}{\partial v_3}\frac{\partial v_3}{\partial v_1} \\
&= \dot{y}|_{v_2} \cdot \dot{v_2}|_{v_1} + \dot{y}|_{v_3} \cdot \dot{v_3}|_{v_1}
\end{aligned}$$

所以，v_1 的梯度在图上，就表现为从 y 到 v_1 的两条路径的累加。一条路是 y，$v2$，v_1，另一条路是 y，$v3$，v_1。因此它的值计算起来是非常清晰的:

$$\begin{aligned}
\dot{y}|_{v_1} &= \dot{y}|_{v_2} \cdot \dot{v_2}|_{v_1} + \dot{y}|_{v_3} \cdot \dot{v_3}|_{v_1} \\
&= v_2.grad \cdot \dot{v_2}|_{v_1} + v_3.grad \cdot \dot{v_3}|_{v_1} \\
&= 1 \cdot 3 + 1 \cdot \frac{1}{3} \\
&= 3\frac{1}{3}
\end{aligned}$$

因此，$v_1.grad = \dot{y}|_{v_1} = 3\frac{1}{3}$。

有了上面的经验，$v_0.grad$ 的计算可以直接从图上算出来，不再推导了：

$$\begin{aligned}\dot{y}|_{v_0} &= \dot{y}|_{v_2}\cdot \dot{v_2}|_{v_0} + \dot{y}|_{v_3}\cdot \dot{v_3}|_{v_0}\\ &= v_2.grad\cdot \dot{v_2}|_{v_0} + v_3.grad\cdot \dot{v_3}|_{v_0}\\ &= 1\cdot 5 + 1\cdot(-\frac{5}{9})\\ &= 4\frac{4}{9}\end{aligned}$$

因此，$v_0.grad = \dot{y}|_{v_0} = 4\frac{4}{9}$。

在图上按照同样的规则，可以计算出 $\dot{y}|_{x_1}$：

$$\begin{aligned}\dot{y}|_{x_1} &= \dot{y}|_{v_1}\cdot \dot{v_1}|_{x_1}\\ &= v_1.grad\cdot 1\\ &= 3\frac{1}{3}\cdot 1\\ &= 3\frac{1}{3}\end{aligned}$$

也在图计算出 $\dot{y}|_{x_0}$：

$$\begin{aligned}\dot{y}|_{x_0} &= \dot{y}|_{v_0}\cdot \dot{v_1}|_{x_0}\\ &= v_0.grad\cdot 1\\ &= 4\frac{4}{9}\cdot 1\\ &= 4\frac{4}{9}\end{aligned}$$

至此，全部计算就完成了，都在图上进行。

注意到，图上的四个节点 v_0，v_1，v_2，v_3，都有 $grad$ 变量，但 y 节点没有，为了保持“形式”上的一致性，也给 y 节点添加 $grad$ 变量，令 $y.grad = 1$，1 乘上任何数都不改变结果，因此设 $y.grad = 1$ 是最合适的。

下一步，用代码实现计算图。

第五章　自动微分的 Python 实现

这一章，用最简单最原始的方式实现自动微分，不做任何优化，以便于理解。

继续使用上一章的例子：

$$v_0 = x_0$$

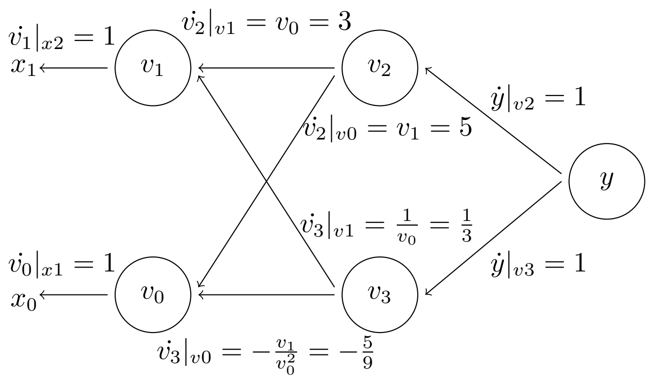
$$v_1 = x_1$$

$$v_2 = v_0 v_1$$

$$v_3 = \frac{v_1}{v_0}$$

$$y = v_2 + v_3$$

把图再贴一次，根据图设计代码：



从节点 y 开始分析。

先用一个类表示节点：

```
class Node(object):
    pass
```

每个节点是 Node 类的实例。每个节点都有自己的梯度，也就是 $grad$，初始值为 0。y 节点的 $grad$ 在运行会设置为 1。

```
class Node(object):

    def __init__(self):
        self.grad = 0
```

每个节点有且只有一个运算 Operation，执行节点的计算功能，实例变量是 op，初始值设为 None。

```
class Node(object):

    def __init__(self):
        self.grad = 0
        self.op = None
```

有运算，就有数值。一个运算可以对 0 个、1 个、多个数值进行运算，实例变量是 inputs，因为数量不定，因此数据类性是列表，初始值设为 []，运算结果存放在变量 value，初始值为 0。

```
class Node(object):

    def __init__(self):
        self.grad = 0
        self.op = None
        self.inputs = []
        self.value = 0
```

每个节点需要计算节点值，增加 evaluate 函数：

```
class Node(object):

    def __init__(self):
        self.grad = 0
        self.op = None
        self.inputs = []
        self.value = 0

    def evaluate(self):
        self.op.compute(self.inputs)
```

每个节点需要计算节点的梯度，增加 gradient 函数，具体的计算，由节点的 op 运算变量执行，又因为后续节点的梯度跟当前节点的梯度是相关的，因此要把当前节点的 $grad$ 值传入。

```python
class Node(object):

    def __init__(self):
        self.grad = 0
        self.op = None
        self.inputs = []
        self.value = 0

    def evaluate(self):
        self.op.compute(self.inputs)

    def gradient(self):
        self.op.gradient(self.inputs, self.grad)
```

定义运算基类 Operation：

```python
class Operation(object):

    def compute(self, inputs):
        pass

    def gradient(self, inputs, current_grad):
        pass
```

实现具体的运算。

第一个是 AddOp。这个运算符，值计算很简单。梯度计算，把当前节点的 grad 累加到后续节点的 grad 上就行了。

```python
class AddOp(Operation):

    def compute(self, inputs):
        return inputs[0].value + inputs[1].value

    def gradient(self, inputs, current_grad):
        inputs[0].grad += current_grad
        inputs[1].grad += current_grad
```

第二个是 MulOp。梯度计算，把当前节点的 grad 乘以另一个输入值，然后累加到后续节点的 grad 上就行了。

```
class MulOp(Operation):

    def compute(self, inputs):
        return inputs[0].value * inputs[1].value

    def gradient(self, inputs, current_grad):
        inputs[0].grad += current_grad*inputs[1].value
        inputs[1].grad += current_grad*inputs[0].value
```

第三个是 DivOp。

```
class DivOp(Operation):

    def compute(self, inputs):
        return inputs[0].value / inputs[1].value

    def gradient(self, inputs, current_grad):
        inputs[0].grad += current_grad/inputs[1].value
        inputs[1].grad += -current_grad*inputs[0].value/pow(inputs[1].value,2)
```

需要把实数封装成 Node。

```
class RealNode(Node):

    def __init__(self, v):
        self.grad = 0
        self.op = None
        self.inputs = None
        self.value = v

    def evaluate(self):
        return

    def gradient(self):
        return
```

执行计算，要从图的左侧向右侧执行，计算微分，要从右侧向左侧执行，因此要对节点进行排序：

```
def get_nodes(root, node_list):
    if root.inputs == None:
        node_list.append(root)
        return
    for i in root.inputs:
        get_nodes(i, node_list)
        if not(i in node_list):
            node_list.append(i)
    node_list.append(root)
```

正向计算输出值，反向计算微分:

```
def run(root):
    node_list = []
    #把所有节点排序，从叶节点排到root，不重复
    get_nodes(root, node_list)
    #计算y值
    for i in node_list:
        i.evaluate()
    #反向微分
    node_list.reverse()
    for i in node_list:
        i.gradient()
```

这些代码是初步设计结果，经过调整后，得到正式代码:

```python
class Node(object):

    def __init__(self, inputs, op):
        self.grad = 0
        self.op = op
        self.inputs = inputs
        self.value = 0

    def evaluate(self):
        self.op.compute(self, self.inputs)

    def gradient(self):
        self.op.gradient(self.inputs, self.grad)


class Operation(object):

    def __call__(self, nodes):
        return Node(nodes, self)

    def compute(self, current_node, inputs):
        pass

    def gradient(self, inputs, current_grad):
        pass


class AddOp(Operation):

    def compute(self, current_node, inputs):
        current_node.value = inputs[0].value + inputs[1].value

    def gradient(self, inputs, current_grad):
        inputs[0].grad += current_grad
        inputs[1].grad += current_grad


class MulOp(Operation):

    def compute(self, current_node, inputs):
        current_node.value = inputs[0].value * inputs[1].value

    def gradient(self, inputs, current_grad):
        inputs[0].grad += current_grad*inputs[1].value
        inputs[1].grad += current_grad*inputs[0].value
```

```
class DivOp(Operation):

    def compute(self, current_node, inputs):
        current_node.value = inputs[0].value / inputs[1].value

    def gradient(self, inputs, current_grad):
        inputs[0].grad += current_grad/inputs[1].value
        inputs[1].grad += -current_grad*inputs[0].value/pow(inputs[1].value,2)


class RealNode(Node):

    def __init__(self, v):
        self.grad = 0
        self.op = None
        self.inputs = None
        self.value = v

    def evaluate(self):
        return

    def gradient(self):
        return


def get_nodes(root, node_list):
    if root.inputs == None:
        node_list.append(root)
        return
    for i in root.inputs:
        get_nodes(i, node_list)
        if not(i in node_list):
            node_list.append(i)
    node_list.append(root)


def run(root):
    node_list = []
    #把所有节点排序，从叶节点排到root，不重复
    get_nodes(root, node_list)
    #计算y值
    for i in node_list:
        i.evaluate()
    #反向微分
    node_list.reverse()
    for i in node_list:
        i.gradient()
```

```
x0 = RealNode(3)
x1 = RealNode(5)
v0 = x0
v1 = x1
v2 = MulOp()([v0, v1])
v3 = DivOp()([v1, v0])
y = AddOp()([v2, v3])
y.grad = 1

run(y)

print('y.value = ', y.value)
print('x0.grad = ', x0.grad)
print('x1.grad = ', x1.grad)
```

代码运行输出结果：

```
y.value =  16.666666666666668
x0.grad =  4.444444444444445
x1.grad =  3.3333333333333335
```

这些值，跟此前计算出来的，$y = 16\frac{2}{3}$，$\dot{y}|_{x_0} = 4\frac{4}{9}$，$\dot{y}|_{x_1} = 3\frac{1}{3}$ 是一致的，符合预期。

这个例子，只实现了加、乘、除三种运算。根据本例，不难写出更多的运算符，以及自定义运算符。

根据本例，也不难写出神经网络的计算图，本质上是一样的。