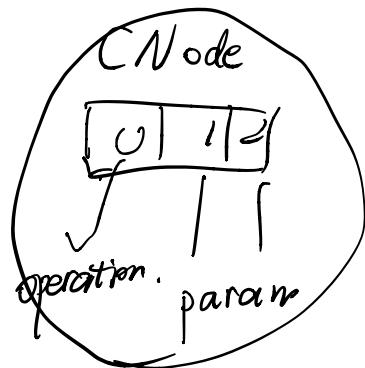


what?

how?

why?

What
where → IR 通用性.



as a param.

func as an operation

手写部分

数值部分

符号部分

U, V_1 placeholder

computational graph { placeholder

6行

operator

if $x_1 = 1$ (as a variant)

$x_2 = 0$ (as a C)

$$J_f = \begin{cases} & \\ & \end{cases}$$

$$y = 3 \sin x \quad \text{if } V_1.$$

$$y = \cos(\sin x)$$

$$V_0 = x = \pi$$

$$-\sin(\sin x) \cdot \cos x.$$

$$V_1 = \sin V_0 = \sin \pi = 0$$

$$y = \cos V_1$$

$$V_2 = \cos V_1 = 1$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial V_1} \cdot \frac{\partial V_1}{\partial x}$$

$$\downarrow y = V_2 = 1$$

$$V_0 = \overline{V_1} \frac{\partial V_1}{\partial V_0} =$$

前向传播求导，一个输入的全部输出

$$\overline{V_1} = \overline{V_2} \cdot \frac{\partial V_2}{\partial V_1} = -\sin V_1 = 0$$

反向传播求导，一个输出的全部输入

$$\overline{V_2} = \overline{y} = 1$$

$$J_f = \left[\begin{array}{c} \frac{\partial y}{\partial x_1} \cdots \frac{\partial y}{\partial x_n} \\ \vdots \\ \frac{\partial y}{\partial x_1} \cdots \frac{\partial y}{\partial x_n} \end{array} \right] \rightarrow \text{反向}$$

compiler.
deep learning
DL framework.

前向

drawback: 反向需要前向的一些值
故要跑2遍。

add \rightarrow operator \rightarrow node.

override
compute function

+ value
+ parents
+ children
- forward

def forward(self):
for node in self.parents
if node.value is None
node.compute()
node.forward

$$x_0 \xrightarrow{f} y \xrightarrow{g} \hat{o}.$$

$$y = f(x) \quad J_f \leftarrow \text{精度.}$$

$$\hat{o} = g(y) = g(f(x)) \quad J_g \leftarrow \text{精度.}$$

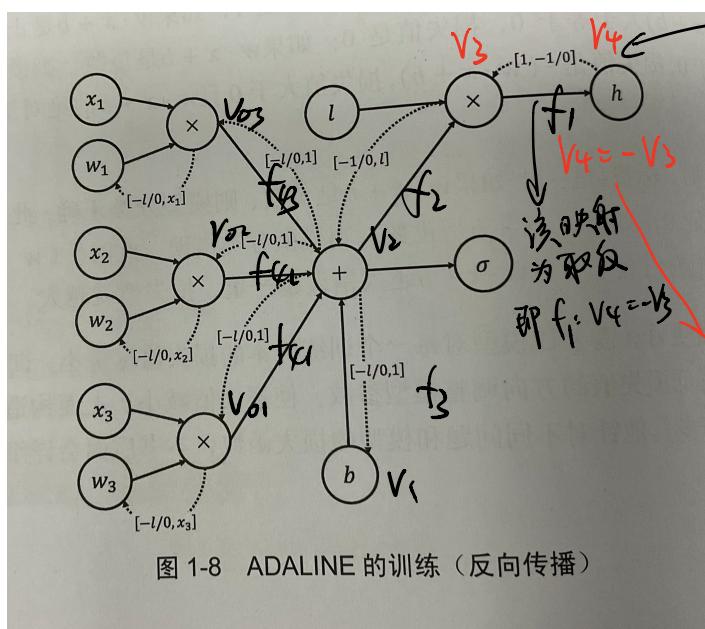


图 1-8 ADALINE 的训练 (反向传播)

为 h 对自身的梯度可梯矩阵

为 h 对 h 节点的梯度可梯矩阵

为 h 对 h 节点的系数为

即 $f_1: V_4 = -V_3$

$$h = \begin{cases} 1 & \text{if } (w \cdot x + b) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$V_4 = \begin{cases} -V_3 & , V_4 \leq 0 \\ 0 & , V_4 \geq 0 \end{cases}$$

节点映射 $J = \begin{cases} -1 & , V_4 \leq 0 \\ 0 & , V_4 \geq 0 \end{cases}$

$f_2: [\text{自身}, \text{自己对父}]$

子节点的 J 相乘即 $1^* -1/0 = -1/0$.

$$V_2 = w_2 x + b$$

$$V_3 = 1 (w_2 x + b)$$

自己对父 J = 1

多个子节点 J $f = \sum_s J_{rs} J_{sf}$ 的数学证明

该节点对结果节点的 Jacobi Matrix

def backward(self, result):

if self.jacobi is None: 出口. 为结果节点, Jacobi 为单行矩阵

if self is result:

self.jacobi = np.mat(np.eye(self.dimension))

else:

self.jacobi = np.mat(

np.zeros((result.dimension, self.dimension)))

for child in self.get_children():

if child.value is not None:

self.jacobi += child.backward(result) *

child.get_jacobi(self)

子节点对结果节点的 Jacobi
即自身的 Jacobi

子节点对父节点的 Jacobi

其中 get_jacobi() 需要重载

类名:
class Mul

return self.jacobi

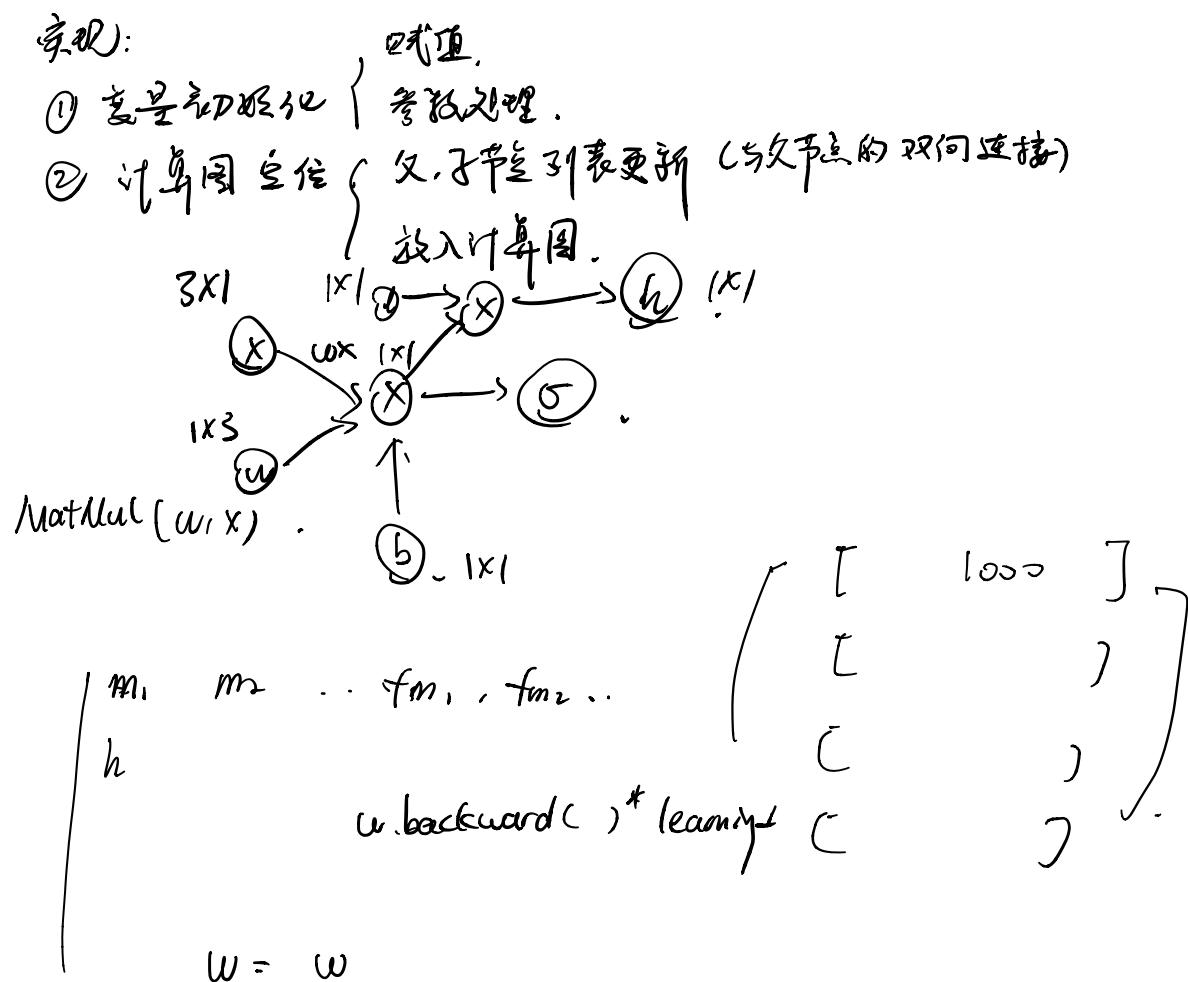
$\begin{matrix} x \\ w \end{matrix} \rightarrow \otimes$

```

def get_jacobi(self, parent):
    return

```

总结一下，计算图的变量节点被赋值或初始化后，在结果节点（比如损失值节点）上调用 `forward` 方法，递归计算路径上各个节点的值，信息沿着计算图向前传播，最终得到结果节点的值。之后，在需要更新的节点上调用 `backward` 方法，递归计算结果节点对路径上各个节点的雅可比矩阵，信息反向传播。如果有多个节点需要更新，比如权值向量节点和偏置节点，就在这些节点上分别调用 `backward` 方法。由于中间节点的雅可比矩阵（如果已经被计算）已经保存在了 `jacobi` 属性中，所以在多个节点上多次调用其 `backward` 方法时并不会增加额外的计算负担。这其实就是“反向传播”的精髓，它执行的无非就是复杂复合映射的求导链式法则，保存中间结果，从而以空间换时间。



ms. ops. MatMul (output, label)

BP, 只有 node node_0 init 时

创建一个 in node, 其 parents \rightarrow output to label, 该 graph 中

但并没有发生计算, perceptionloss (ms. ops. MatMul (output, label))

其中的 compute() to loss. forward() 通用, 也创建了一个节点, 该 compute()

也就是说, 该节点为



且只有 perceptionloss 的 compute()