③ if $n_e \notin$ tape

return n_i.sum_outgrads()

④ if getValue(n_e) $\notin R$

raise error: $f \notin R$

⑤ n_e.outgrads.append 1.

$\forall n \in$ tape

n.send_upstream

return n_i.sum_outgrads()

2. Differentiable D:

f, forward_pass $fp \rightarrow f_{new}$

where f_{new} is differentiable version of f.

1. $g: f(x_1, x_n), i \rightarrow f'(x_1, x_n)$

$:= \frac{df}{dx_i}(x_1, x_n)$

1.3 $f'(x_1, x_n)$

① top_tape = max(x_1.tape, x_n.tape)

$$\text{tape} = \begin{cases} \text{new list, priority}=1, & \text{if top tape} = \text{none} \\ \text{new list, pri} = \text{top\_tape.pri} + 1, & \text{otherwise.} \end{cases}$$

② start node n_i = Node(x_i, tape)

$x_1, x_i, x_n \leftarrow x_1, n_i, x_n$

end node $n_e = f(x_1, \cdots, x_n)$

1、$\forall x \in \{x_1, x_n\}$

$\Rightarrow \{1\}$ = parent ops = par_ops $\leq$ gfs (i, if)

⑥ return Node(y, tape, par_ops).

3. primitive P:

f, grad maker gm $\rightarrow f_{new}$

3.3 $f_p(x_1, x_n) \rightarrow y$, $gm(y, x_1, x_n)$

① $y \leftarrow f(x_1, x_n)$

② return y, $gm(y, x_1, x_n)$

3.4 P(f, gm)

① $f_p \leftarrow f$, gm. ② $f_{new} \leftarrow D(f, f_p)$

2.3 $f_{new}(x_1, x_n)$

① tape $\leftarrow$ max(x_1.tape, x_n.tape)

② if tape = none

return $y = f(x_1, x_n)$

③ val $\leftarrow$ { x.value, if $x \in$ tape
④ x, ow.

1、$\forall x \in \{x_1, x_n\} \Rightarrow \{val\} = vals$.

④ result y, grad fns gfs $\leftarrow f_p$(vals)

⑤ (grad fns gfs(i), x) (if) $x \in$ tape,

4.5 sum_outgrads sum_ogs :

① if |ogs| = 1 and ogs(1).value (getValue) ≠ Setter : return ogs(1)

② s ← zeros()

③ ∀ new ∈ ogs

s ← mutating_add(s, new)

return s.

4.6. get_item(idx) : take(n, idx)

5. getValue : { return getValue(x.value) if x is Node
return x, otherwise

4. Node n

4.2 value v, tape, par_ops, outgrads ogs

4.3 v, t, par.

value ← v, tape ← t, tape.append n.

par_ops ← par, ogs ← {}

4.4 send_upstream :

if ogs :

s ← sum_outgrads()

∀ (gradfun gf, par) ∈ par_ops

par.ogs.append(gf(s))

9. mutating_add_ma ← P(ma, λ ans y,
old, new : [λg:g, λg:g])

10. take(A, idx): return A(idx)

11. take ← P(take, λ y, A, idx:
[λ g: untake(g, idx)])

12. untake(x, idx): return Setter(idx, x)

13. untake ← P(untake, λ y, x, idx:
[λ g: take(g, idx)])

6. zeros_like(x)

return Node(x, cal tape(none)).zeros()

6.3 cal tape : prev_tape → new tape

① new tape = { new list, pri=1, if pt=none
new list, pri=pt.pri+1, ow.

7. Setter = tuple("Setter", ("idx", "val"))

8. mutating_add (old, new)

if new is Setter :
if old(new.idx)=0 : old(n.idx)←n.val
else old(n.idx)+=n.val.
else old+=new.
return old.