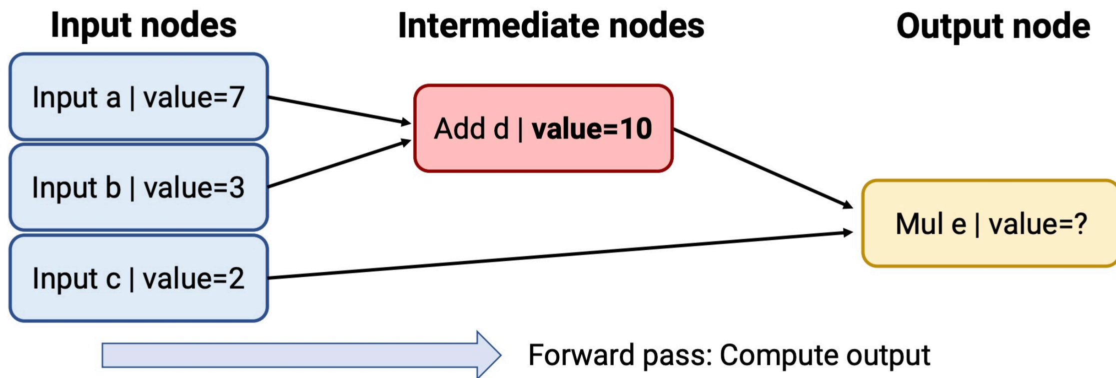
Computation Graph

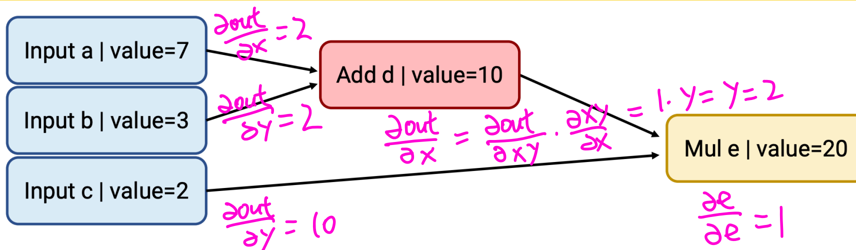
Computation graph for ***(a + b) * c*** when *a=7, b=3, c=2*



Gradient checking

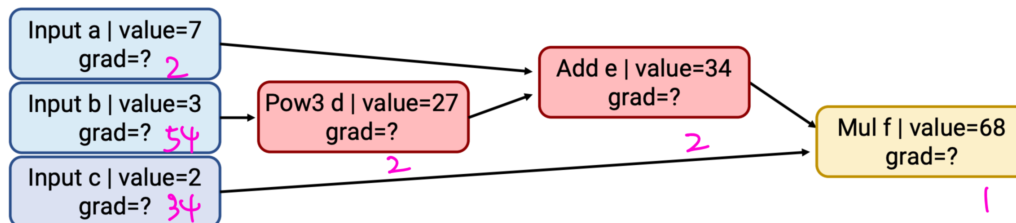
- numerical gradients
- Easy to implement
- Slow - O(#inputs)

Backpropagation on trees



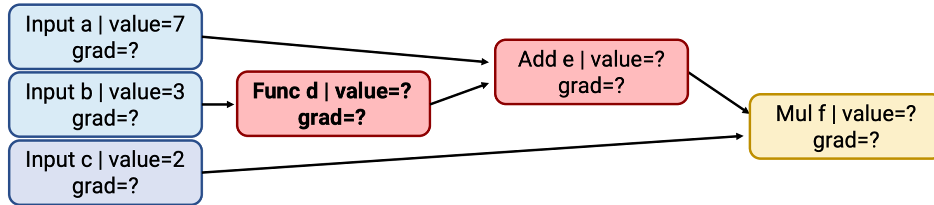
- For now: assume that the computation graph is a tree
  - Each node is only used in a single computation
  - Root of tree is output
  - Leaves of tree are inputs
- Idea: Recursively compute ***∂(output)/∂(node)*** **for each node**, starting at output

PowerNode



New function: *(a + b^3) * c* where *a=7, b=3, c=2*

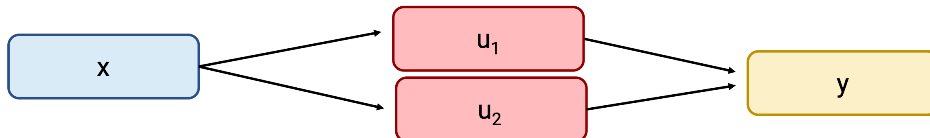
Generic Unary Function



New function: *(a + Func(b)) * c* where *a=7, b=3, c=2*

- Steps 1-3 are the same
- Step 4: Func (generic function)
  - child.grad = parent.grad * *∂(Func(child))/∂child*

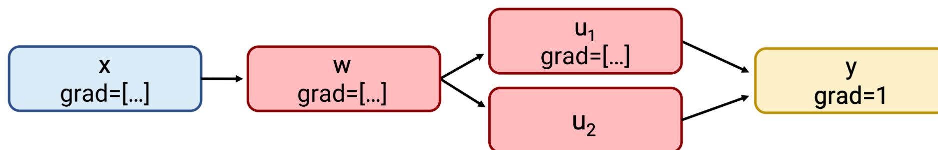
Multivariate chain rule



- Minimal example
  - u_1 and u_2 depend on variable x (i.e., x has two parents)
  - y depends on both u_1 and u_2
- By Multivariate Chain Rule: $\partial y/\partial x = \partial y/\partial u_1 * \partial u_1/\partial x + \partial y/\partial u_2 * \partial u_2/\partial x$
  - Changing x by epsilon changes each parent a bit; Effects add for small epsilon
  - Generalization of multiplying derivatives is matrix multiplication of Jacobians

Dot product of $[\partial y/\partial u_1, \partial y/\partial u_2]$ & $[\partial u_1/\partial x, \partial u_2/\partial x]$

What order of traversal?



Better code:

- topo_order = [x, w, u_1, u_2, y]
- Iterate in reverse order:
  - y.backward()
  - u2.backward()
  - u1.backward()
  - w.backward()
  - x.backward()

- Going recursively double-counts
  - First call to w.backward() makes final x.grad too large
- Solution: **Topological sort the nodes**
  - Iterate in reverse order, starting from output
  - Ensures that we process each node after all of its parents