Module 1.3 - Backprop

Functions

- Function $f(x) = x \times 5$

```python
class TimesFive(ScalarFunction):
    @staticmethod
    def forward(ctx, x: float) -> float:
        return x * 5
```

Multi-arg Functions

- Function $f(x, y) = x \times y$

```python
class Mul(ScalarFunction):
    @staticmethod
    def forward(ctx, x: float, y: float) -> float:
        return x * y
```

Loading [MathJax]/extensions/Safe.js

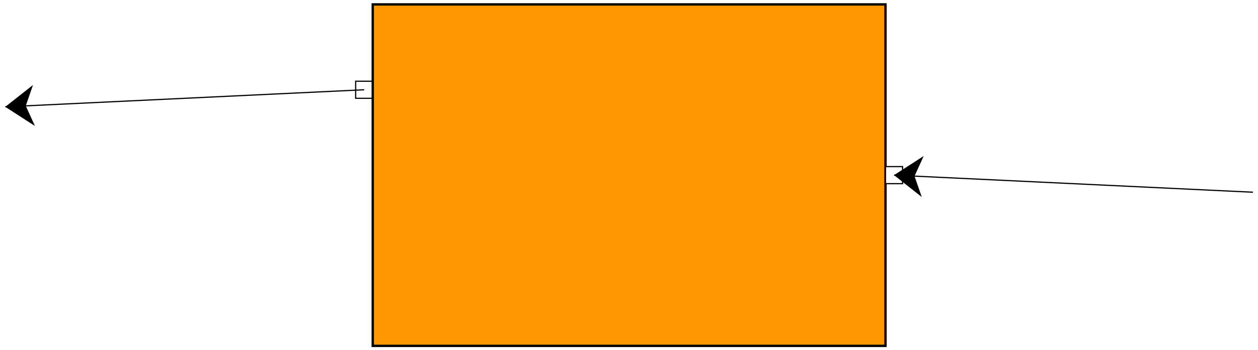
Context

$$f(x) = x^2$$

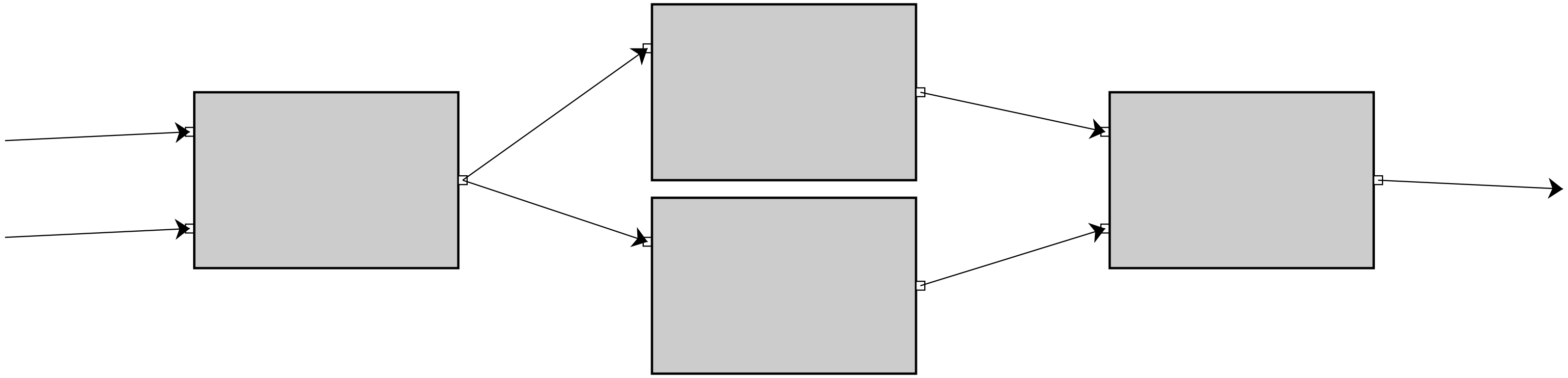
$$f'(x) = 2 \times x$$

```
class Square(ScalarFunction):
    @staticmethod
    def forward(ctx: Context, x: float) -> float:
        ctx.save_for_backward(x)
        return x * x

    @staticmethod
    def backward(ctx: Context, d: float) -> Tuple[float, float]:
        (x,) = ctx.saved_values
        f_prime = 2 * x
        return f_prime * d
```

Box for Function



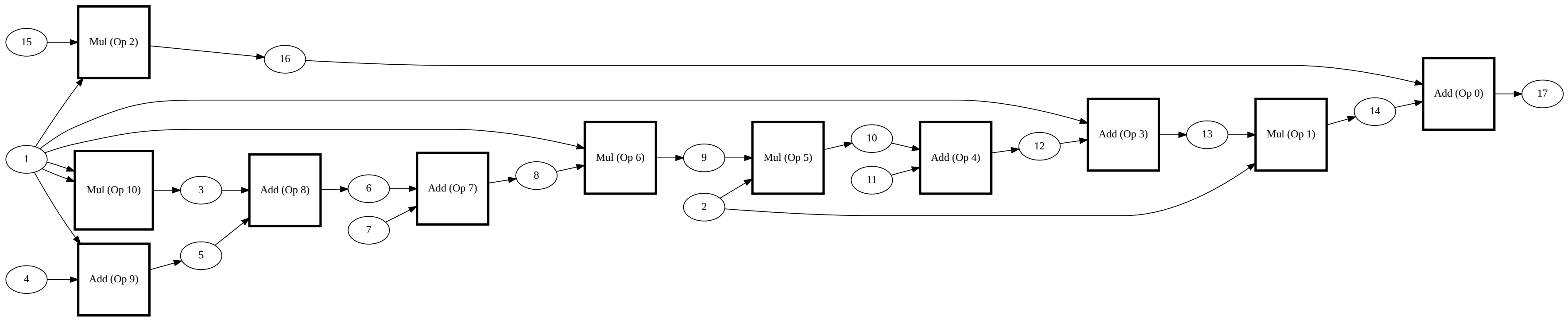
Computational Graph



Loading [MathJax]/extensions/Safe.js

Forward Graph

```
def expression():
    x = Scalar(1.0)
    y = Scalar(1.0)
    z = (sum([1, x, x * x, 65]) * x * y + 6 + x) * y + 10.0 * x

    return z
```



Loading [MathJax]/extensions/Safe.js

Lecture Quiz

Outline

- Chain Rule
- Backpropagation

Chain Rule

Graph Structure

```
x = Scalar(2.0)
x_2 = Square.apply(x)
print(x_2.history)
```

```
ScalarHistory(last_fn=<class '__main__.Square'>, ctx=Context(no_grad=Fal
se, saved_values=(2.0,)), inputs=[Scalar(2.0)])
```

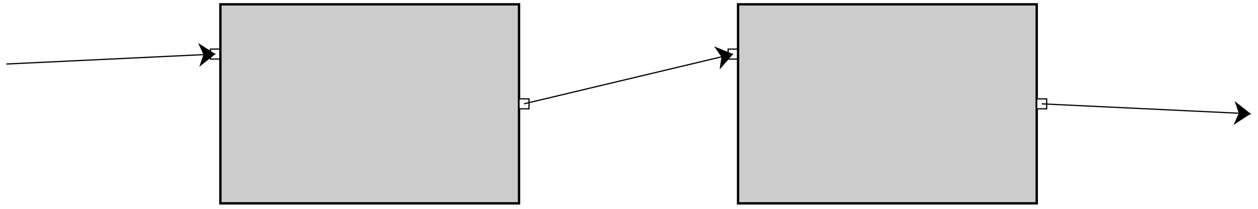
```
print(x_2.history.inputs[0].history)
```

```
ScalarHistory(last_fn=None, ctx=None, inputs=())
```

Derivative

```
x = Scalar(2.0)
x_2 = Square.apply(x)
x_3 = Square.apply(x_2)
x_3.backward()
print(x.derivative)
```

```
32.0
```

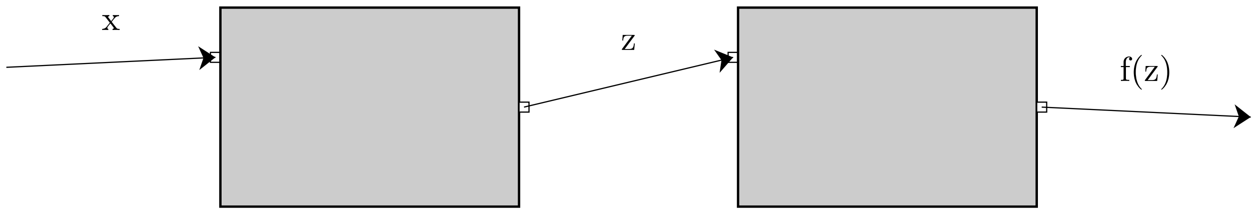


Loading [MathJax]/extensions/Safe.js

Chain Rule

Compute derivative from chain

$$f(g(x)) = f(z)$$



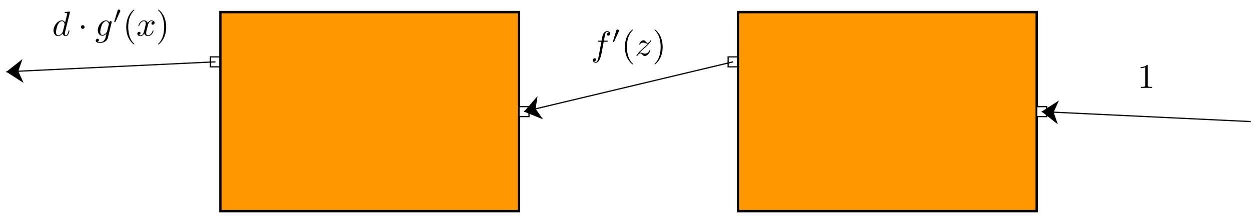
Chain Rule

Compute derivative from chain

$$f'_x(g(x)) = g'(x) \times f'_{g(x)}(g(x))$$

Chain Rule

$$z = g(x)$$
$$d = f'(z)$$
$$f'_x(g(x)) = g'(x) \times d$$



Example: Chain Rule

$$log(x)^2$$

$$\begin{aligned} f(z) \quad &= z^2 \\ g(x) &= \log(x) \end{aligned}$$

Loading [MathJax]/extensions/Safe.js

Example: Chain Rule

$$
\begin{aligned}
f'(z) &= 2z \times 1 \\
g'(x) &= 1/x
\end{aligned}
$$

What is the combination?

$$f'_x(g(x))$$

Example: Chain Rule

$$((x)^2)^2$$

$$\begin{aligned} f(z) &= z^2 \\ g(x) &= x^2 \end{aligned}$$

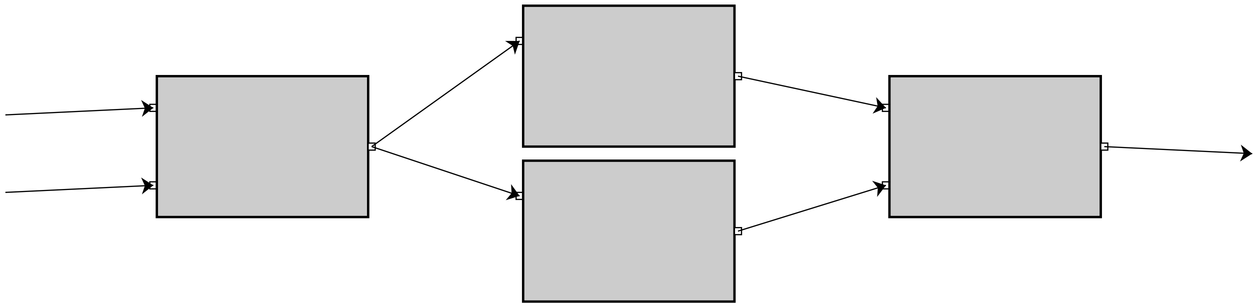
$$\begin{aligned} f'(z) &= 2 \times z \\ g'(x) &= 2 \times x \end{aligned}$$

Example: Chain Rule

$$f'_x(g(x)) = 2 \times x \times 2 \times x^2 = 4x^3$$

Two Arguments: Chain

$$f(g(x, y))$$



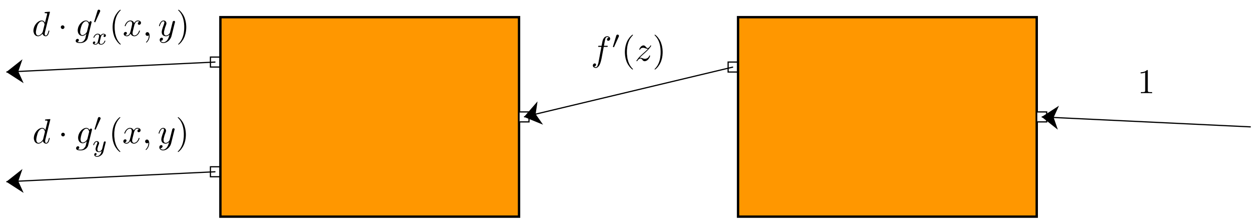
Two Arguments: Chain

$$f'_x(g(x,y)) = g'_x(x,y) \times f'_{g(x,y)}(g(x,y))$$

$$f'_y(g(x,y)) = g'_y(x,y) \times f'_{g(x,y)}(g(x,y))$$

Two Arguments: Chain

$$z = g(x, y)$$
$$d = f'(z)$$
$$f'_x(g(x, y)) = g'_x(x, y) \times d_{out}$$
$$f'_y(g(x, y)) = g'_y(x, y) \times d_{out}$$



Example: Chain Rule

$$(x \times y)^2$$

$$\begin{aligned} f(z) &= z^2 \\ g(x, y) &= (x \times y) \end{aligned}$$

Example: Chain Rule

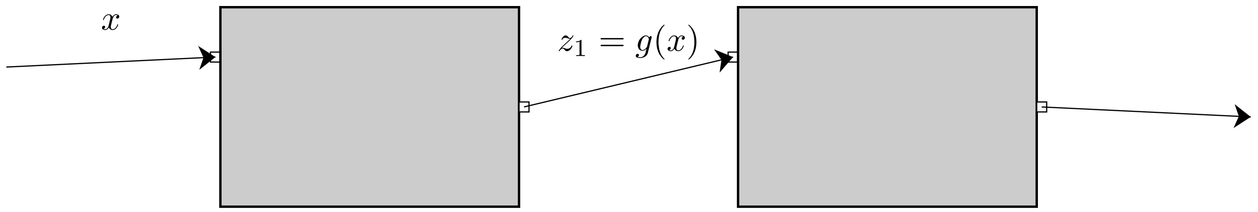
$$f'(z) = 2z \times 1$$
$$g'_x(x, y) = y$$
$$g'_y(x, y) = x$$

What is the combination?

$$f'_x(g(x, y)) = 2zy$$
$$f'_y(g(x, y)) = 2zx$$

Multivariable Chain

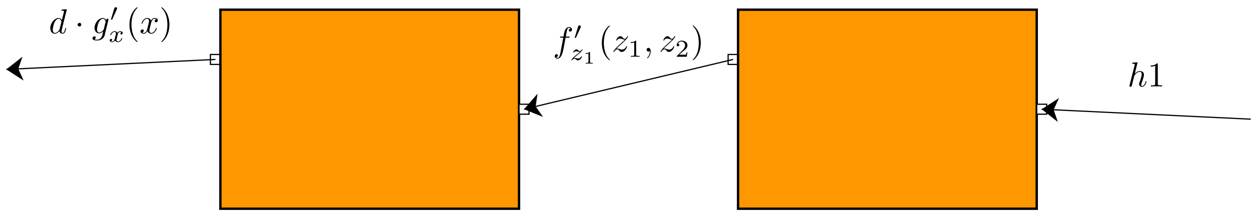
$$f(g(x), g(x))$$



Loading [MathJax]/extensions/Safe.js

Multivariable Chain

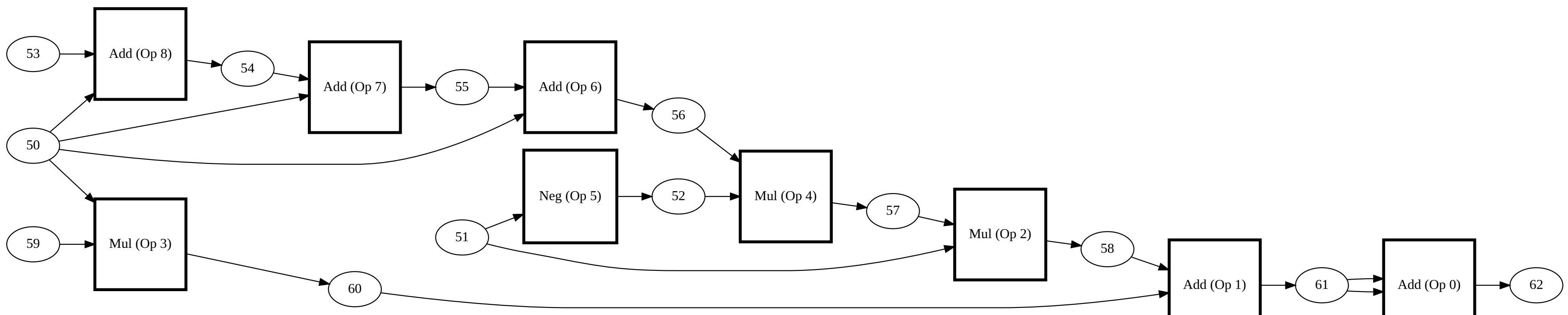
$$d = 1 \times f'_{z_1}(z_1, z_2) + 1 \times f'_{z_2}(z_1, z_2)$$
$$h'_x(x) = d \times g'_x(x)$$



Backpropagation

Complex Graphs

```python
def expression():
    x = Scalar(1.0, name="x")
    y = Scalar(1.0, name="y")
    z = -y * sum([x, x, x]) * y + 10.0 * x
    return z + z
```

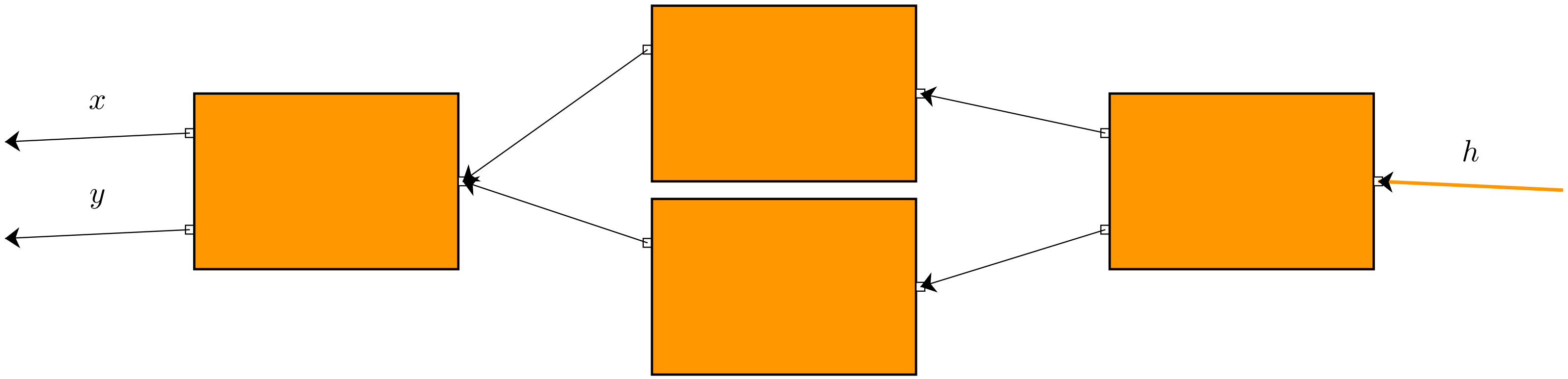


Goal

- Efficient implementation of chain-rule
- Assume access to the graph.
- Goal: Call backward once per variable

Full Graph

$$z = x \times y$$
$$h(x, y) = \log(z) + \exp(z)$$



Loading [MathJax]/extensions/Safe.js

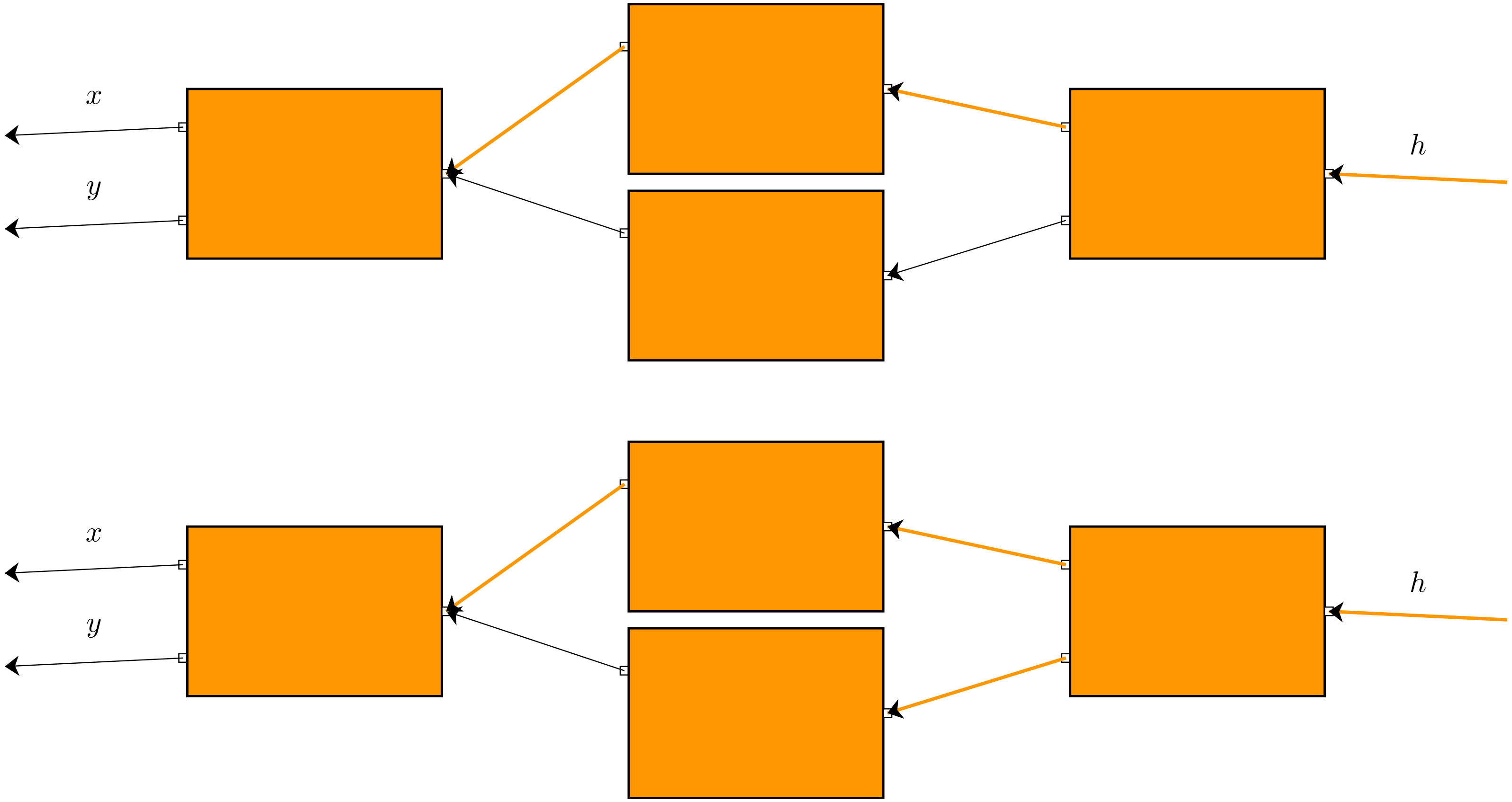
Tool

If we have:

- the derivative with respect to a scalar
- the function last called on the scalar

We can apply the chain rule through that function.

Step



Issue

Order matters!

- If we proceed without finishing a variable, we may need to apply chain rule multiple times

Desired property: all derivatives for a variable before backward.

Ordering Step

- Do not process any Variable until all downstream Variables are done.
- Collect a list of the Variables first.

Topological Sorting

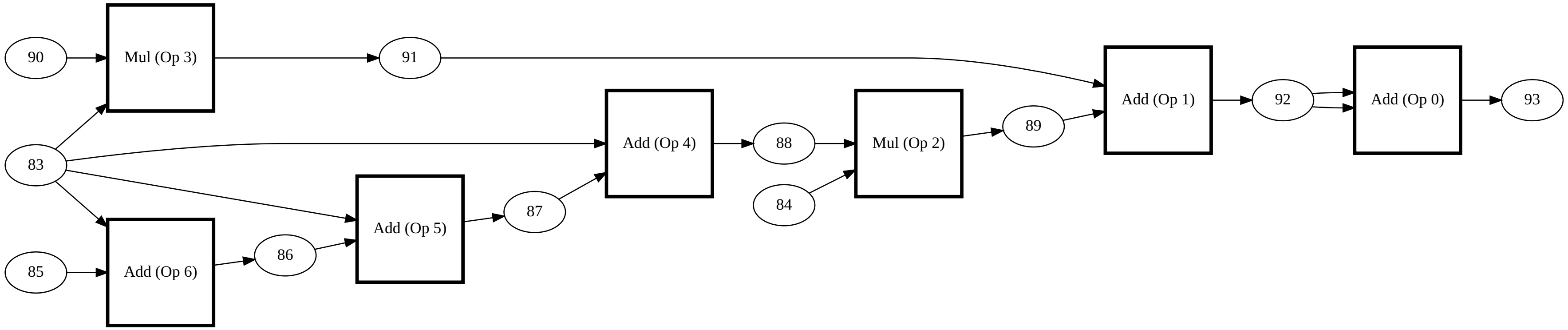
- Topological Sorting
- High-level -> Run depth first search and mark nodes.

Topological Sorting

```
visit(last)

function visit(node n)
  if n has a mark then return

  for each node m with an edge from n to m do
      visit(m)

  mark n with a permanent mark
  add n to list
```

Topological Sorting

```
def expression():
    x = Scalar(1.0, name="x")
    y = Scalar(1.0, name="y")
    z = sum([x, x, x]) * y + 10.0 * x
    return z + z
```



Backpropagation

- Graph propagation
- Ensure flow to original Variables.

Terminology

- Leaf: Variable created from scratch
- Non-Leaf: Variable created with a Function
- Constant: Term passed in that is not a variable

Algorithm: Outer Loop

0. Call topological sort

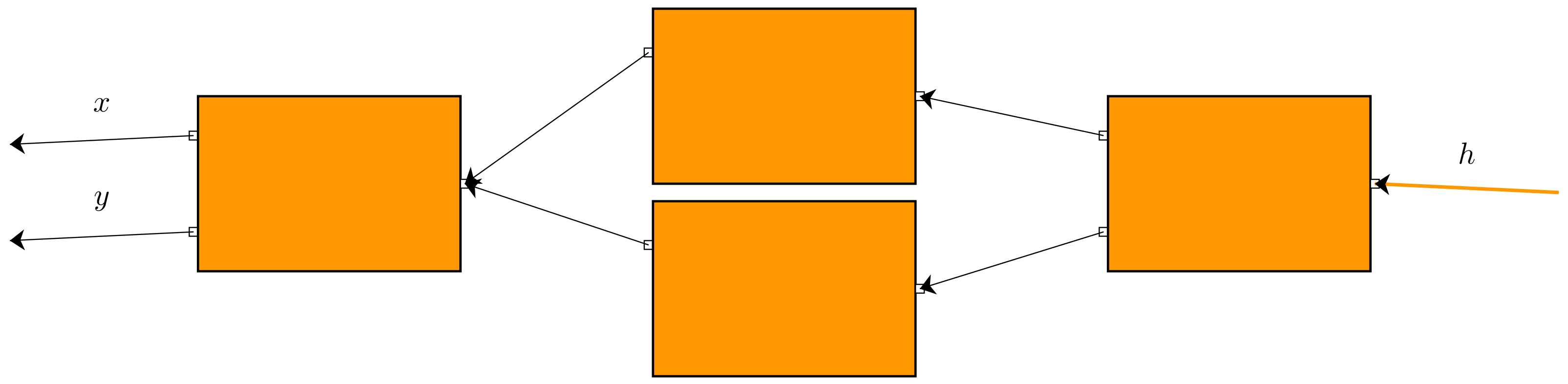
1. Create dict of Variables and derivatives

2. For each node in backward order:

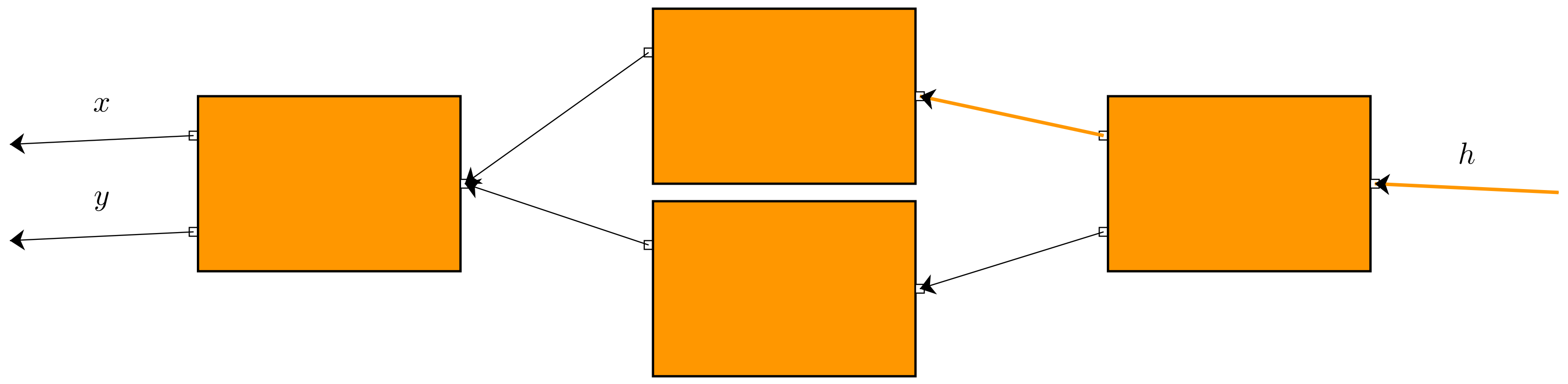
Algorithm: Inner Loop

```
1. if Variable is leaf, add its final derivative
2. if the Variable is not a leaf,
  A. call backward with $d$
  B. loop through all the Variables+derivative
  C. accumulate derivatives for the Variable
```

Example

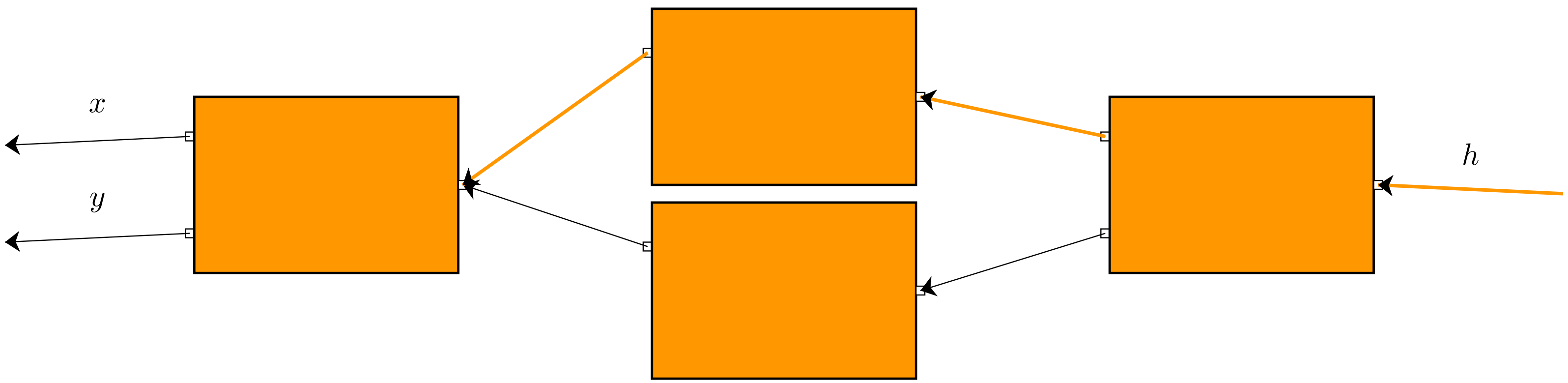


Example

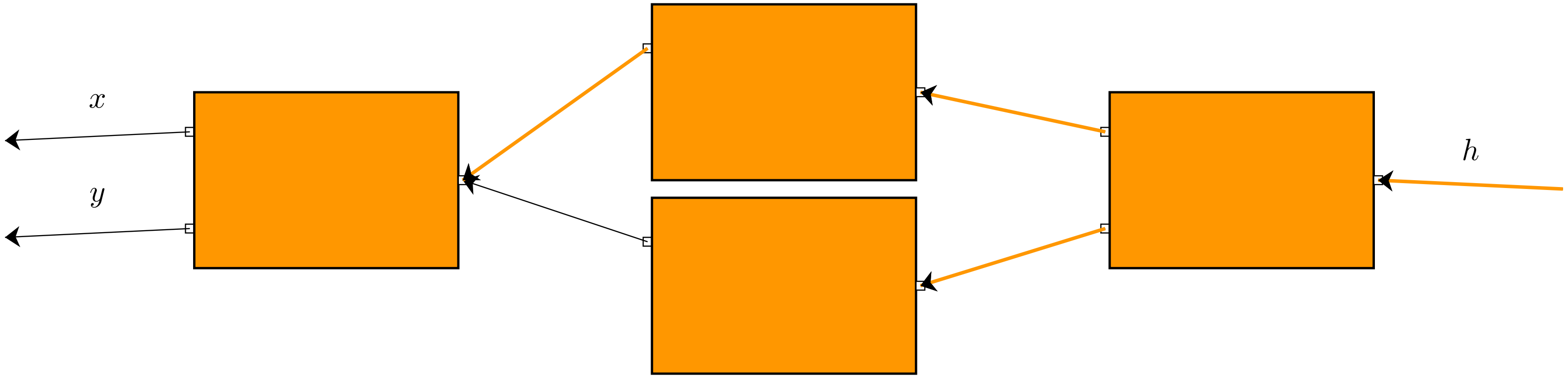


Loading [MathJax]/extensions/Safe.js

Example

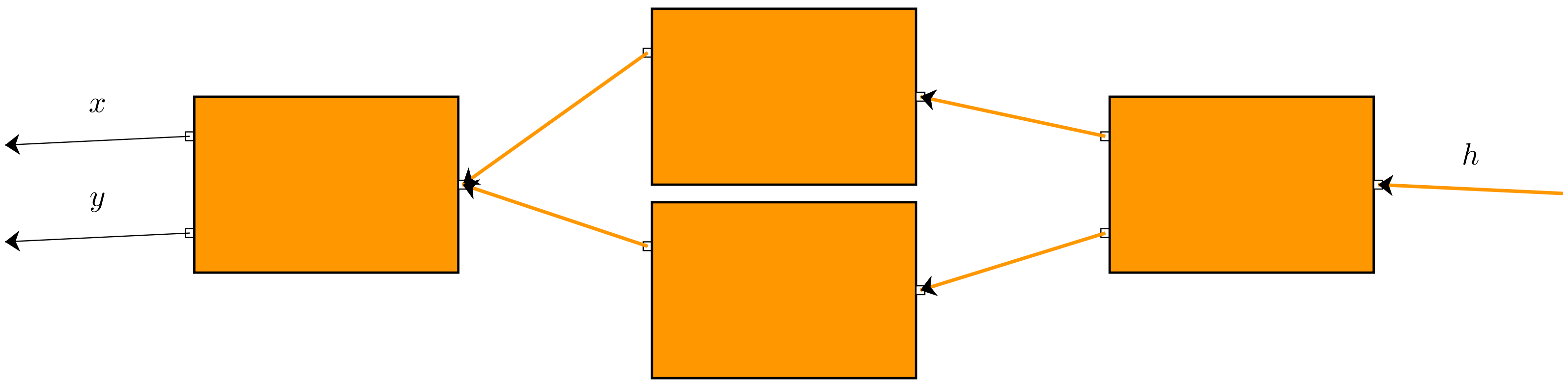


Example

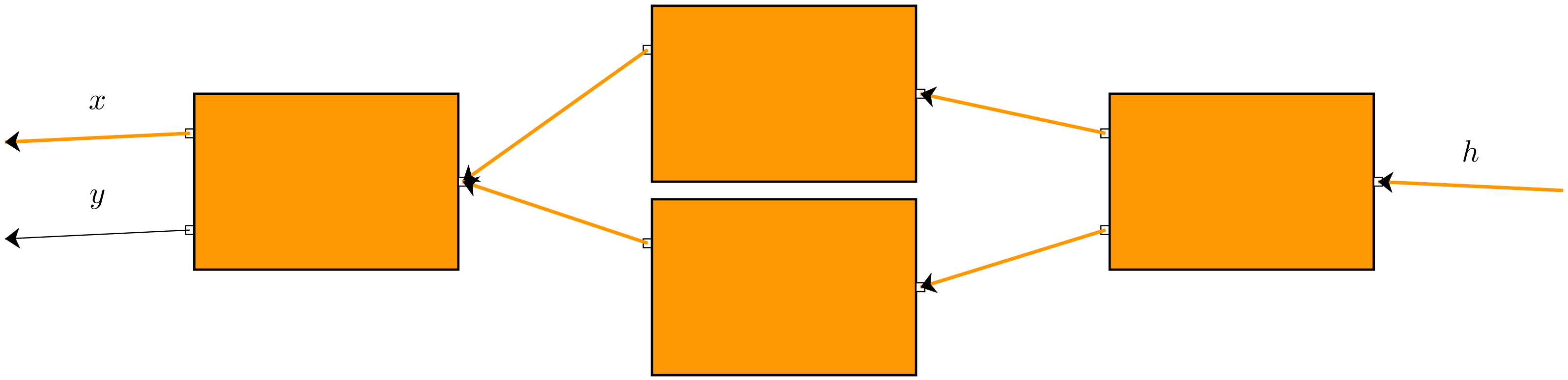


Loading [MathJax]/extensions/Safe.js

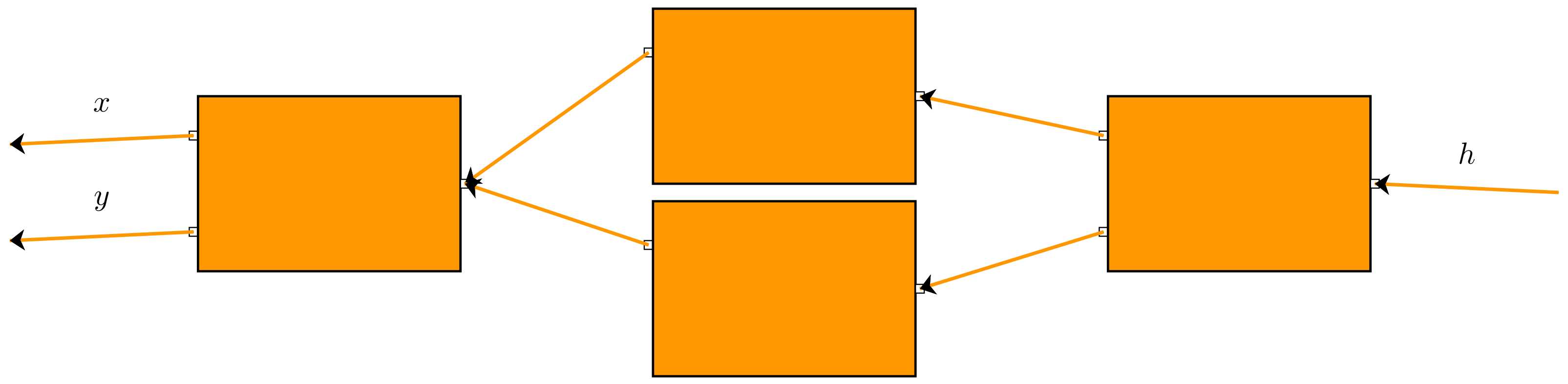
Example



Example



Example



QA