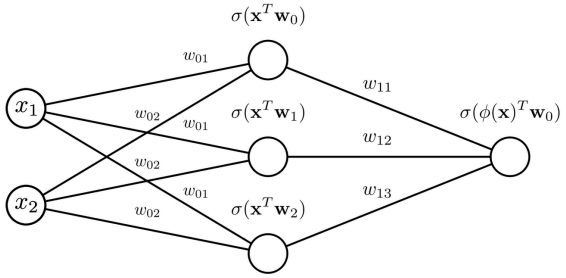
Automatic differentiation: Motivation



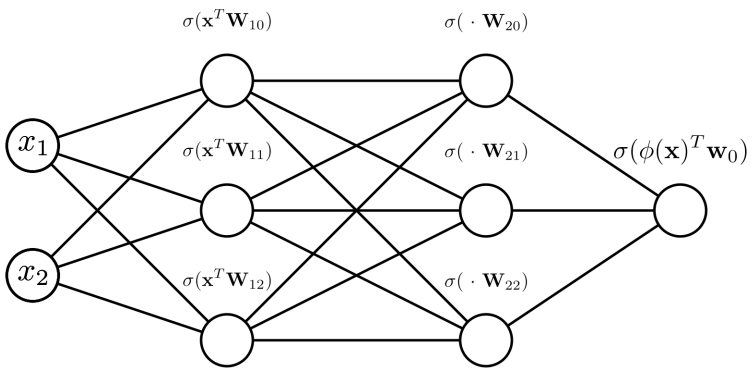
$$\mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{N} \log \sigma\big((2y_i - 1)\phi(\mathbf{x}_i)^T\mathbf{w}\big)$$

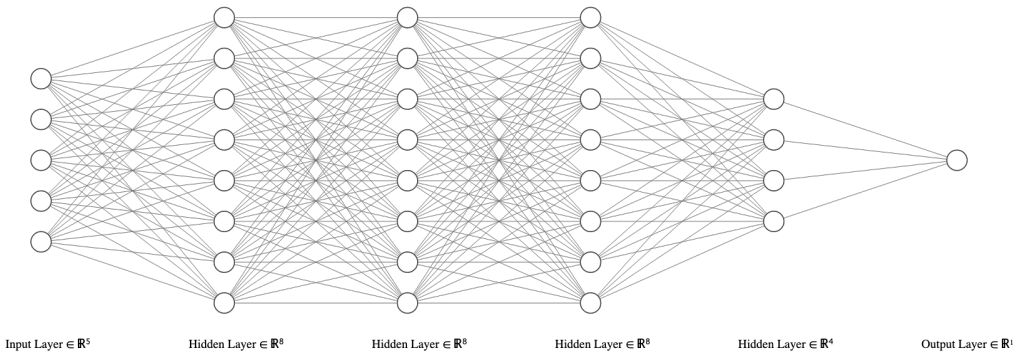
$$= -\sum_{i=1}^{N} \log \sigma\big((2y_i - 1)\sigma\big(w_{01} \cdot \sigma(x_1 W_{11} + x_2 W_{12}) + w_{02} \cdot \sigma(x_1 W_{21} + x_2 W_{22}) + w_{03} \cdot \sigma(x_1 W_{31} + x_2 W_{32})\big)\big)$$

$$W_{11}^{(i+1)} \leftarrow W_{11}^{(i)} - \alpha \frac{\partial NLL}{\partial W_{11}}$$

Automatic differentiation: Motivation

$$\mathbf{NLL}(\mathbf{w}_0, \mathbf{W}, \mathbf{X}, \mathbf{y}) = -\sum_{i=1}^{N} \log \sigma\big((2y_i - 1)\phi(\mathbf{x}_i)^T \mathbf{w}\big)$$





Derivatives revisited

Given a function $f: \mathbb{R} \to \mathbb{R}$, a derivative is the *instantaneous rate of change*

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Derivatives revisited

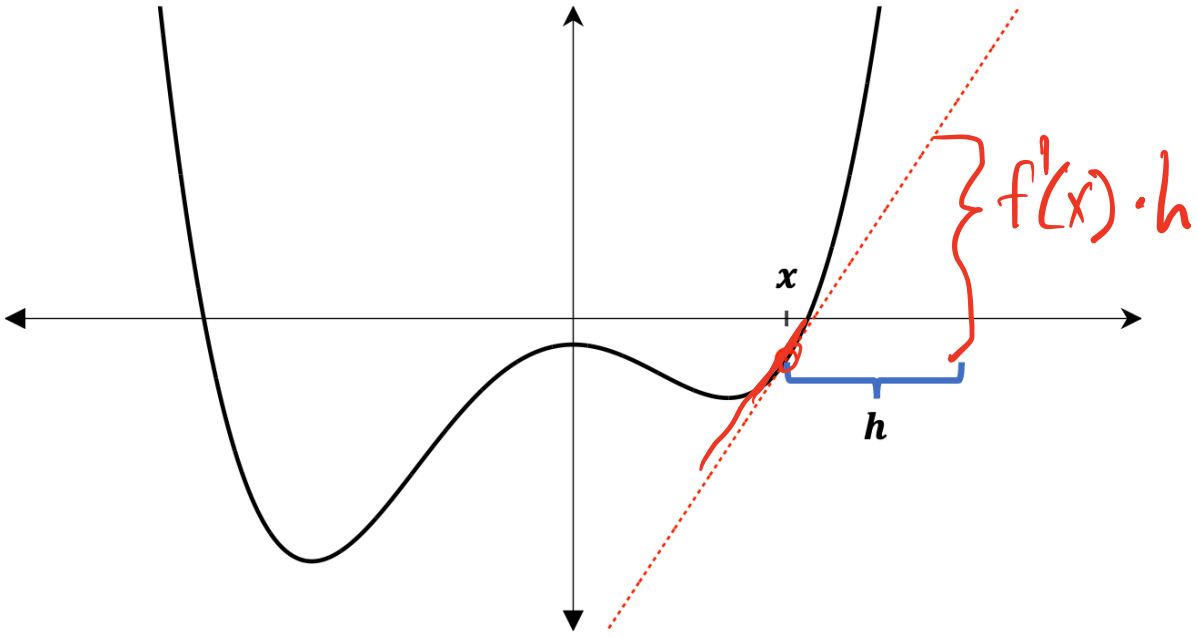
Given a function $f: \mathbb{R} \to \mathbb{R}$, a derivative is the *instantaneous rate of change*

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Equivalently, it's the slope of the *optimal linear approximation* of f near x.

Derivatives revisited

Optimal linear approximation



Consider the question:

"If I perturb x by h, how does the output change?"

The chain rule

Where does this appear?

Consider the chain rule

$$\frac{\partial}{\partial x} f\big(g(x)\big) \cdot 1 = f'\big(g(x)\big) \cdot g'(x) \cdot 1$$

If I perturb x by 1, by how much does $f\big(g(x)\big)$ change?

(under a first-order approx.)

The chain rule

Where does this appear?

Consider the chain rule

$$\frac{\partial}{\partial x} f\big(g(x)\big) \cdot 1 = f'\big(g(x)\big) \cdot g'(x) \cdot 1$$

If I perturb x by 1, by how much does $g(x)$ change?

The chain rule

Where does this appear?

Consider the chain rule

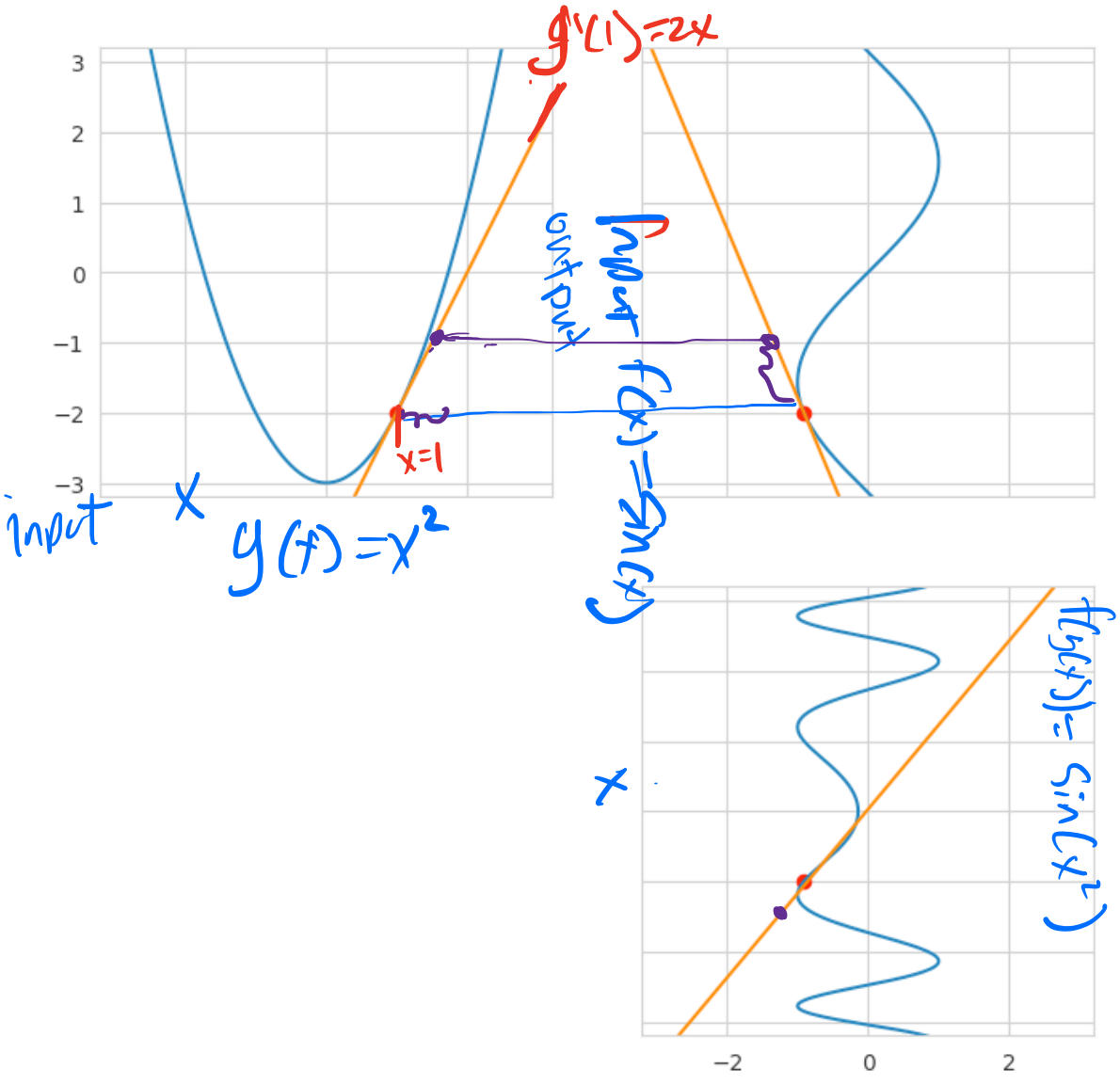
$$\frac{\partial}{\partial x} f\big(g(x)\big) \cdot 1 = f'\big(g(x)\big) \cdot h$$

If I perturb $g(x)$ by h, by how much does $f(g(x))$ change?

(to the first order)

To find the derivative of a complicated function, we iteratively map input perturbations to output perturbations for each sub-function

Chain rule illustrated



Useful notation

Function View

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$$

Value View

$$b = g(x)$$

$$a = f(b) = f(g(x))$$

$$\frac{da}{dx} = \frac{da}{db}\frac{db}{dx}$$

Corresponds to code

how do a change as we change x?

```
b = x ** 2
a = log(b)
```

Useful notation

```
b = x ** 2
a = log(b)
```

$$b = g(x)$$

$$a = f(b)$$

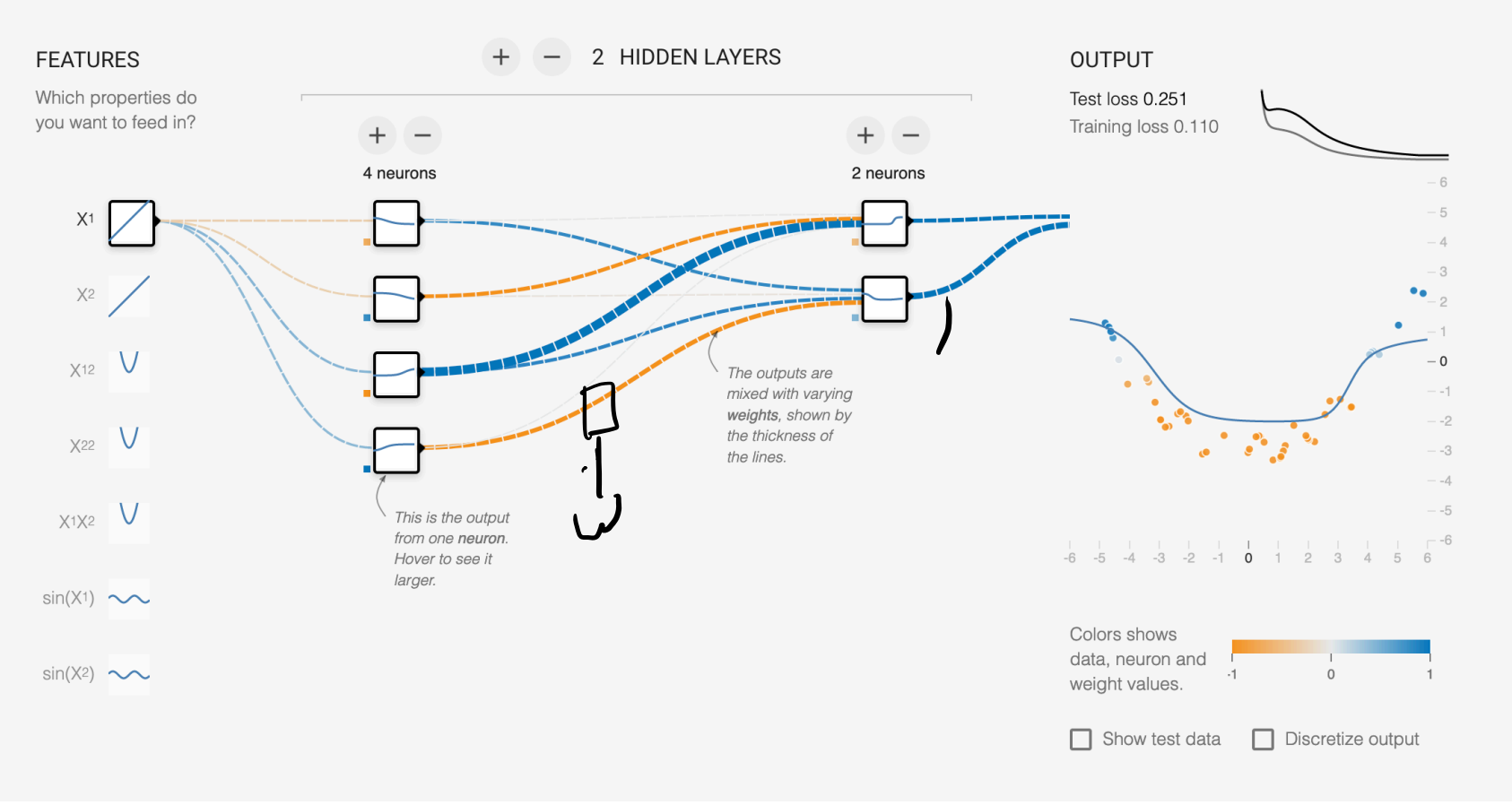
$$a = \log(b), \quad b = x^2$$

$$\frac{da}{db} = \frac{1}{b} \qquad \frac{db}{dx} = 2x$$

$$\frac{da}{dx} = \frac{da}{db}\frac{db}{dx}$$

$$\frac{da}{dx} = \frac{1}{b} \cdot 2x = \frac{2x}{x^2} = \frac{2}{x}$$

Neural networks



Total derivatives

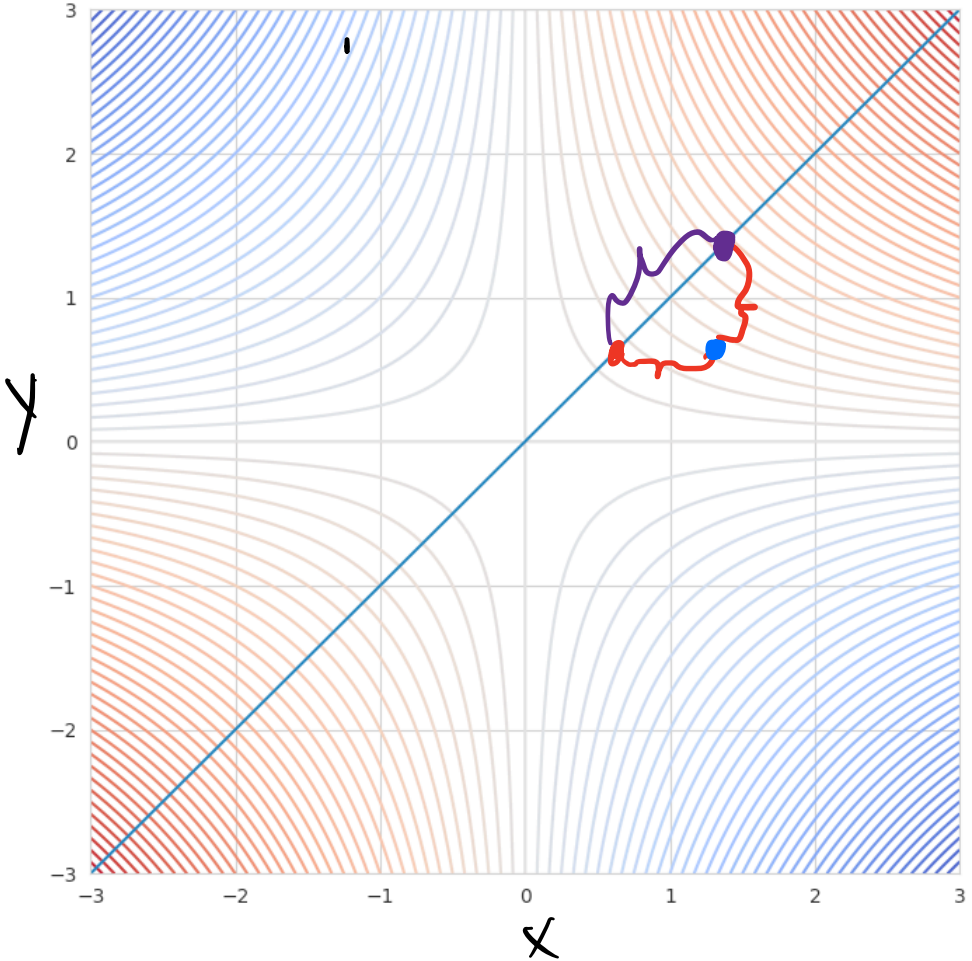
$$\frac{df(y(x), z(x))}{dx} = \frac{df}{dy}\frac{dy}{dx} + \frac{df}{dz}\frac{dz}{dx}$$

$f(y,z) = y \cdot z^2$

$y = 2x$

$z = \exp(x)$

Total derivatives



$$f(x, y) = xy$$

$$\frac{df}{dx} = y \qquad \frac{df}{dy} = x$$

$$\frac{d}{dx} f(x, x) = x + x = 2x$$

$$\frac{d}{dx} x^2 = 2x$$

Total derivatives

$$\frac{df(y(x), z(x))}{dx} = \frac{df}{dy}\frac{dy}{dx} + \frac{df}{dz}\frac{dz}{dx}$$

Total derivatives

$$\frac{df(y(x),x)}{dx} = \frac{df}{dy}\frac{dy}{dx} + \frac{df}{dx}$$

$$\frac{df}{dx} = \frac{\partial f}{\partial y}\cdot\frac{\partial y}{\partial x} + \frac{\partial f}{\partial x}$$

Partial derivatives

$$\frac{df(y(x),x)}{dx} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x} + \frac{\partial f}{\partial x}$$

Composing the chain rule

$$L = -\log \sigma(wx^2)$$

$$\frac{dL}{dx} = ?$$

$$L(g(L(b(a(x)))))$$

$$L'(g(\dots))\, g'(L(\dots))\, L'(b(\dots)) \dots$$

$$a = x^2$$

$$b = wa$$

$$c = \sigma(b)$$

$$g = \log c$$

$$L = -g$$

Composing the chain rule

$$L = -\log \sigma\left(wx^2\right)$$

$$a = x^2 \quad \frac{da}{dx} = 2x$$

$$b = wa \quad \frac{db}{da} = w$$

$$c = \sigma(b) \quad \frac{dc}{db} = \sigma(b)(1-\sigma(b))$$

$$g = \log c \quad \frac{dg}{dc} = \frac{1}{c}$$

$$L = -g \quad \frac{dL}{dg} = -1$$

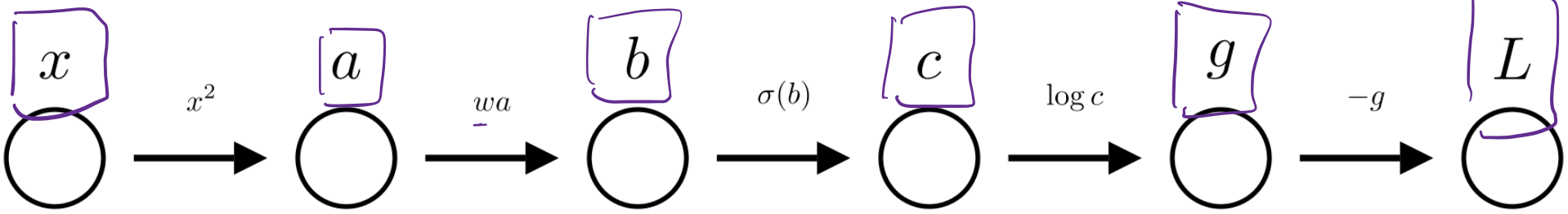
$$\frac{dL}{dx} = \frac{dL}{dg}\frac{dg}{dc}\frac{dc}{db}\frac{db}{da}\frac{da}{dx}$$

$$(-1)\left(\frac{1}{c}\right)(\sigma(\cdot))(w)(2x)$$

Computational graphs

$$\frac{dL}{dx}$$

$$L = -\log \sigma(wx^2)$$

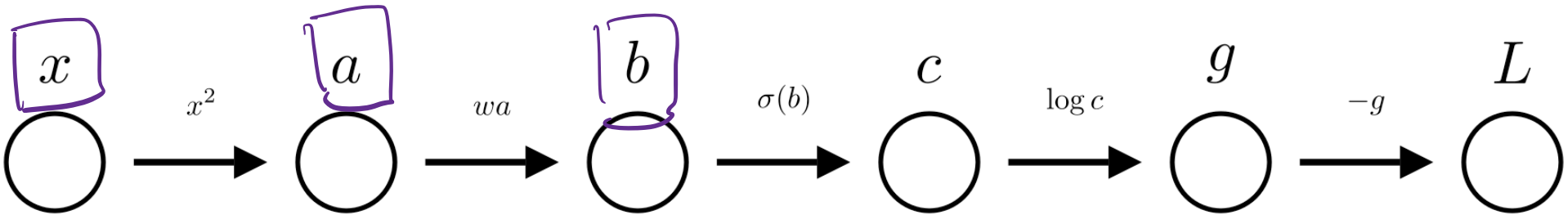


$\frac{da}{dx} = 2x$ $\quad \frac{db}{da} = w$ $\quad \frac{dc}{db} \sigma(b)(1-\sigma(b))$ $\quad \frac{dg}{dc} = \frac{1}{c}$ $\quad \frac{dL}{dg} = -1$

$$\frac{dL}{dx} = \frac{da}{dx} \cdot \frac{dy}{da} \cdot \ldots$$

Forward-mode automatic differentiation

$$L = -\log \sigma(wx^2)$$



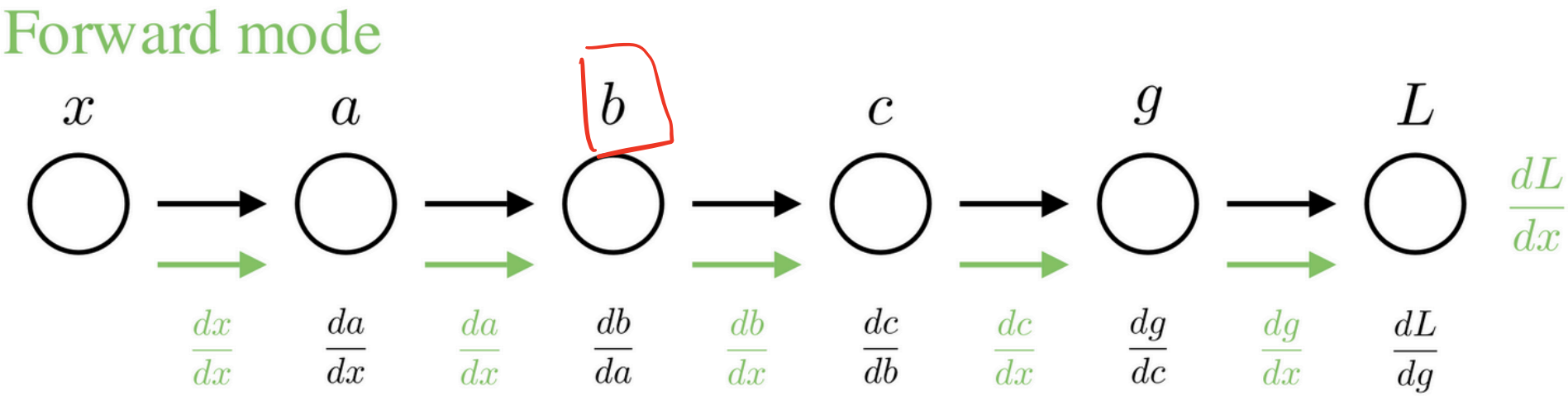
$\frac{da}{dx} = 2x$ $\quad \frac{db}{da} = w$ $\quad \frac{dc}{db} = \sigma(b)(1-\sigma(b))$ $\quad \frac{dg}{dc} =$ $\quad \frac{dL}{dg}$

$\frac{db}{dx} = 2x \cdot w \rightarrow \frac{dc}{dx} = \frac{dc}{db} \cdot \frac{db}{dx}$

$\hookrightarrow \frac{dg}{dx} = \frac{dg}{dc} \cdot \frac{dc}{dx}$

Forward-mode automatic differentiation

$$L = -\log \sigma\left(wx^2\right)$$



Automatic differentiation with multiple inputs

$$-\log \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3)$$

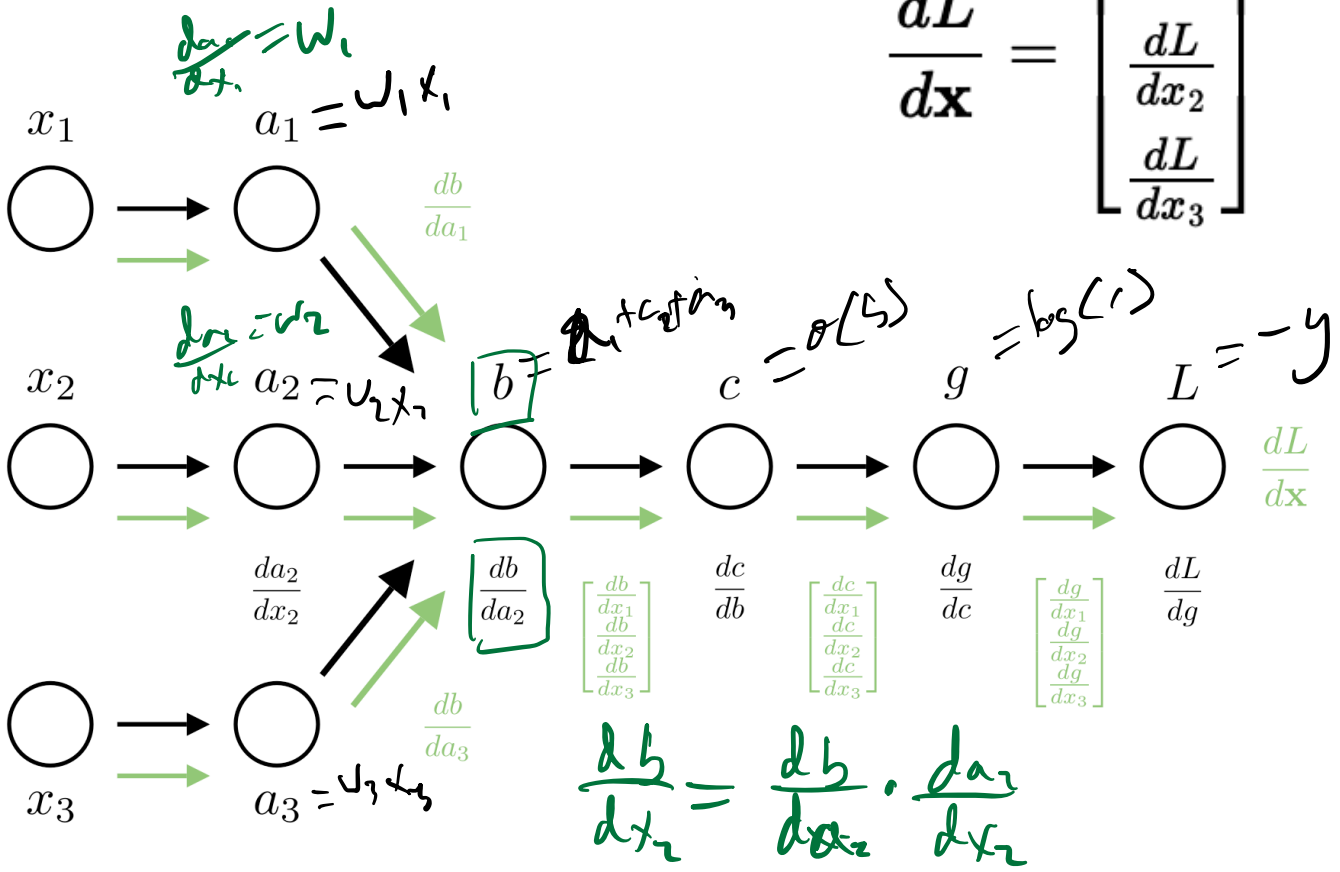
$$\frac{dL}{d\mathbf{x}} = \begin{bmatrix} \frac{dL}{dx_1} \\ \frac{dL}{dx_2} \\ \frac{dL}{dx_3} \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Automatic differentiation with multiple inputs

$$-\log \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3)$$

$$\frac{dL}{d\mathbf{x}} = \begin{bmatrix} \frac{dL}{dx_1} \\ \frac{dL}{dx_2} \\ \frac{dL}{dx_3} \end{bmatrix}$$



Reusing values

```
def loss(x):
    a = x ** 2
    b = 5 * a
    c = log(a)
    g = b * c
    L = -g
    return L
```

$$a = x^2$$

$$b = 5a$$

$$c = \log a$$

$$g = bc$$

$$L = -b$$

Reusing values

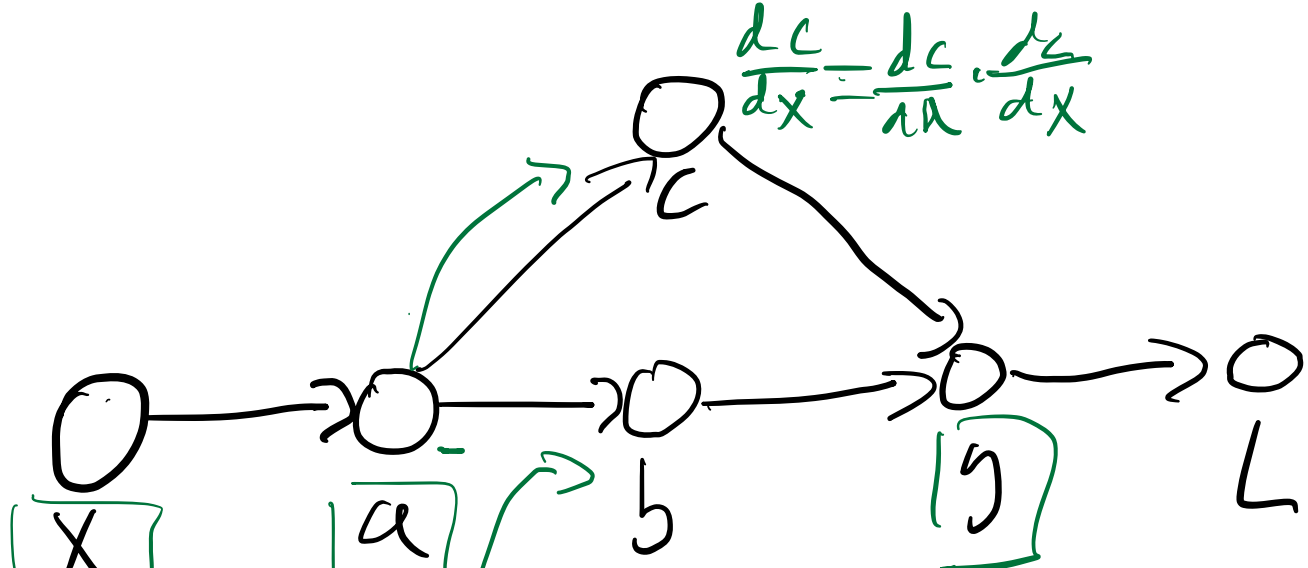
$$a = x^2$$

$$b = 5a$$

$$c = \log a$$

$$g = bc$$

$$L = g$$



$\frac{dc}{dx} = \frac{dc}{da} \cdot \frac{da}{dx}$

$\frac{da}{dx}$

$\frac{db}{dx} = \frac{db}{da} \cdot \frac{da}{dx}$

$\frac{dg}{dx} = \frac{dg}{db} \cdot \frac{db}{dx} + \frac{dg}{dc} \cdot \frac{dc}{dx}$

Total del. $= \frac{dg(b(x), c(x))}{dx} = \frac{dg}{db} \cdot \frac{db}{dx} + \frac{dg}{dc} \cdot \frac{dc}{dx}$

Reusing values

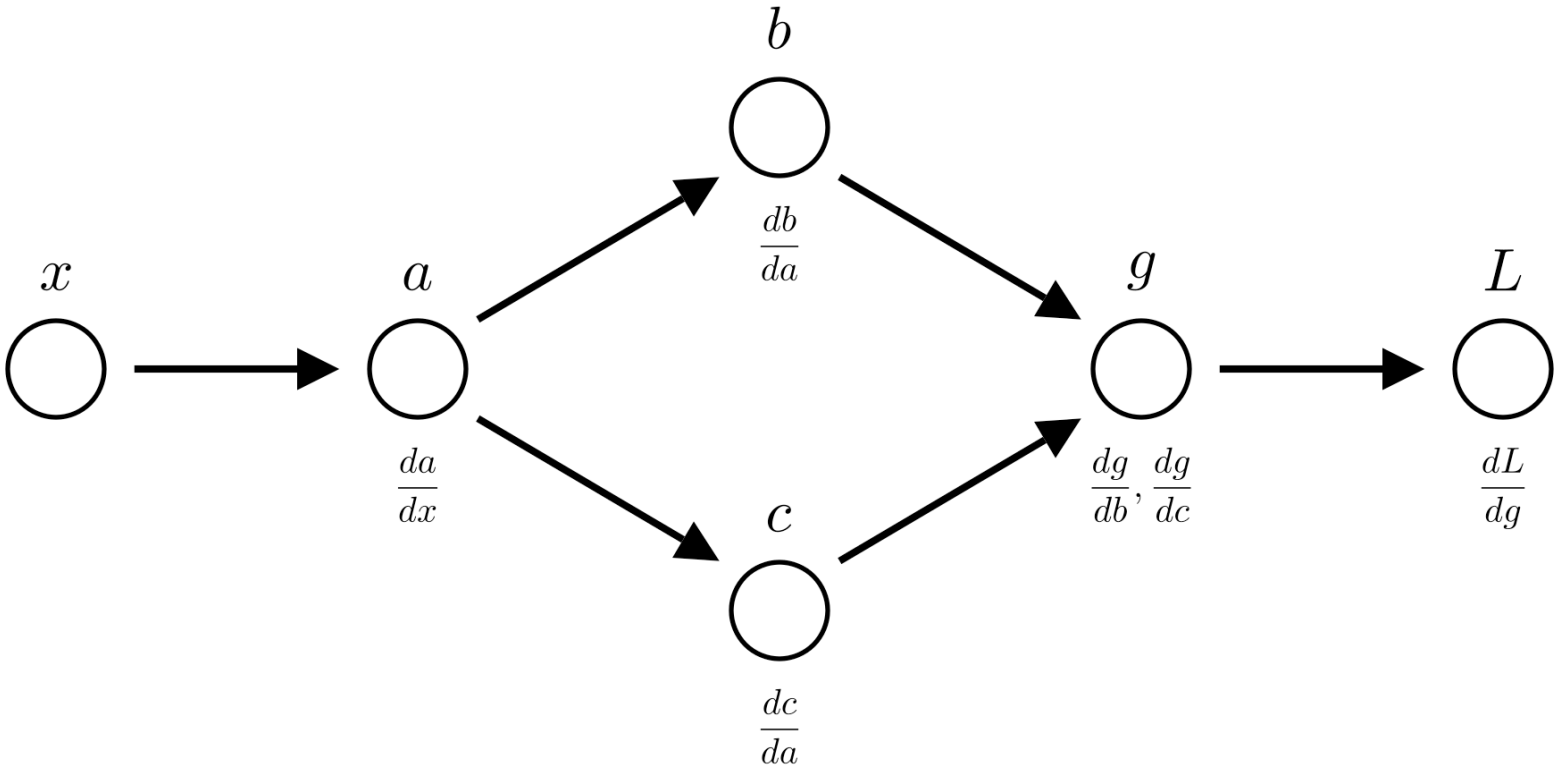
$$a = x^2$$

$$b = 5a$$

$$c = \log a$$

$$g = bc$$

$$L = -b$$



Partial derivatives revisited

$$a = x^2$$

$$b = 5a$$

$$c = ab$$

$$L = -c$$

Total

$$\frac{dc}{da}$$

$$1$$

$$10a$$

$$\neq$$

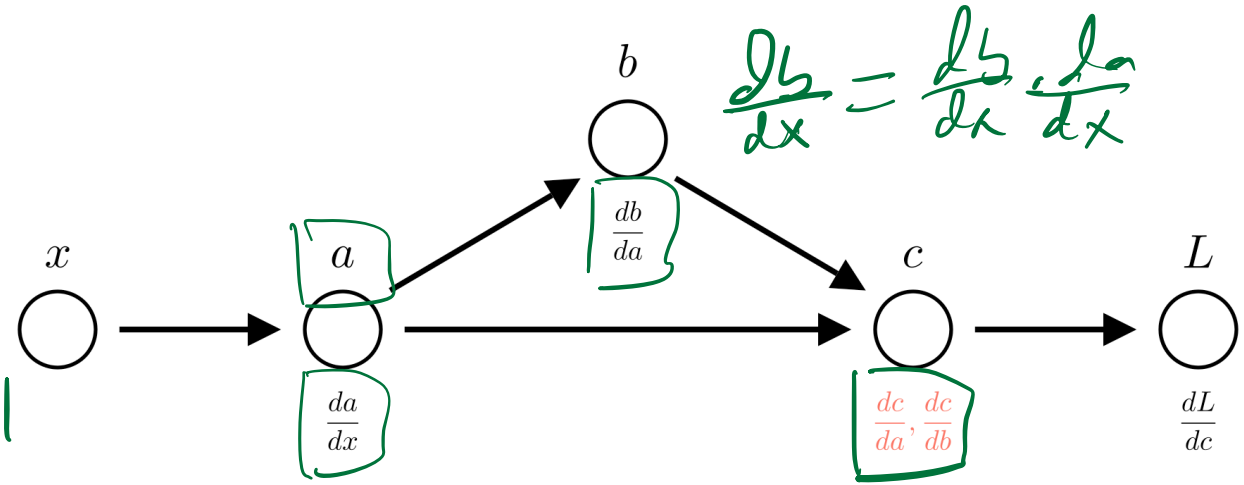
Partial

$$\frac{\partial c}{\partial a}$$

$$1$$

$$b = 5a$$

$$\frac{dc}{da} =$$



$$\frac{db}{dx} = \frac{db}{da} \cdot \frac{da}{dx}$$

$$\frac{dc}{dx} = \frac{dc}{da}\frac{da}{dx} + \frac{dc}{db}\frac{db}{dx}$$

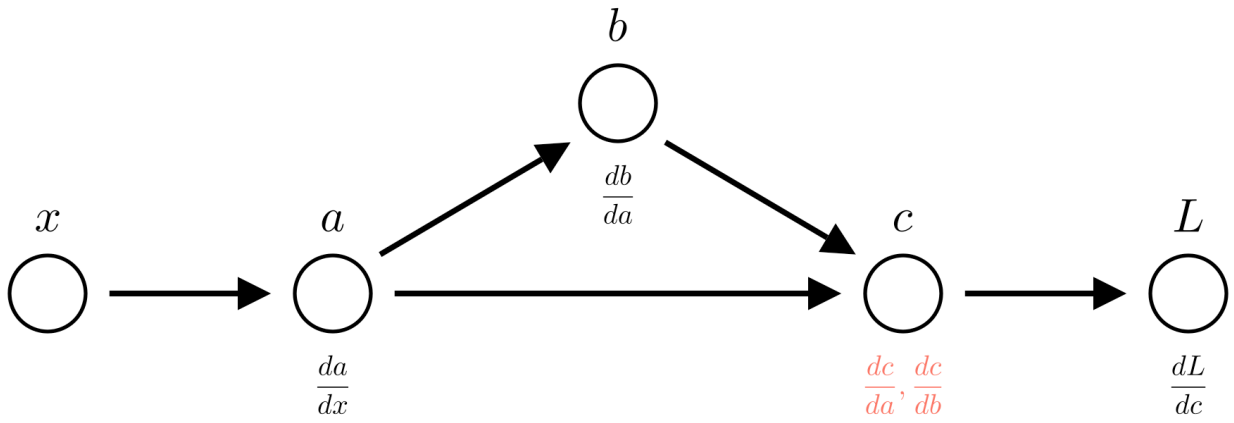
Partial derivatives revisited

$$a = x^2$$

$$b = 5a$$

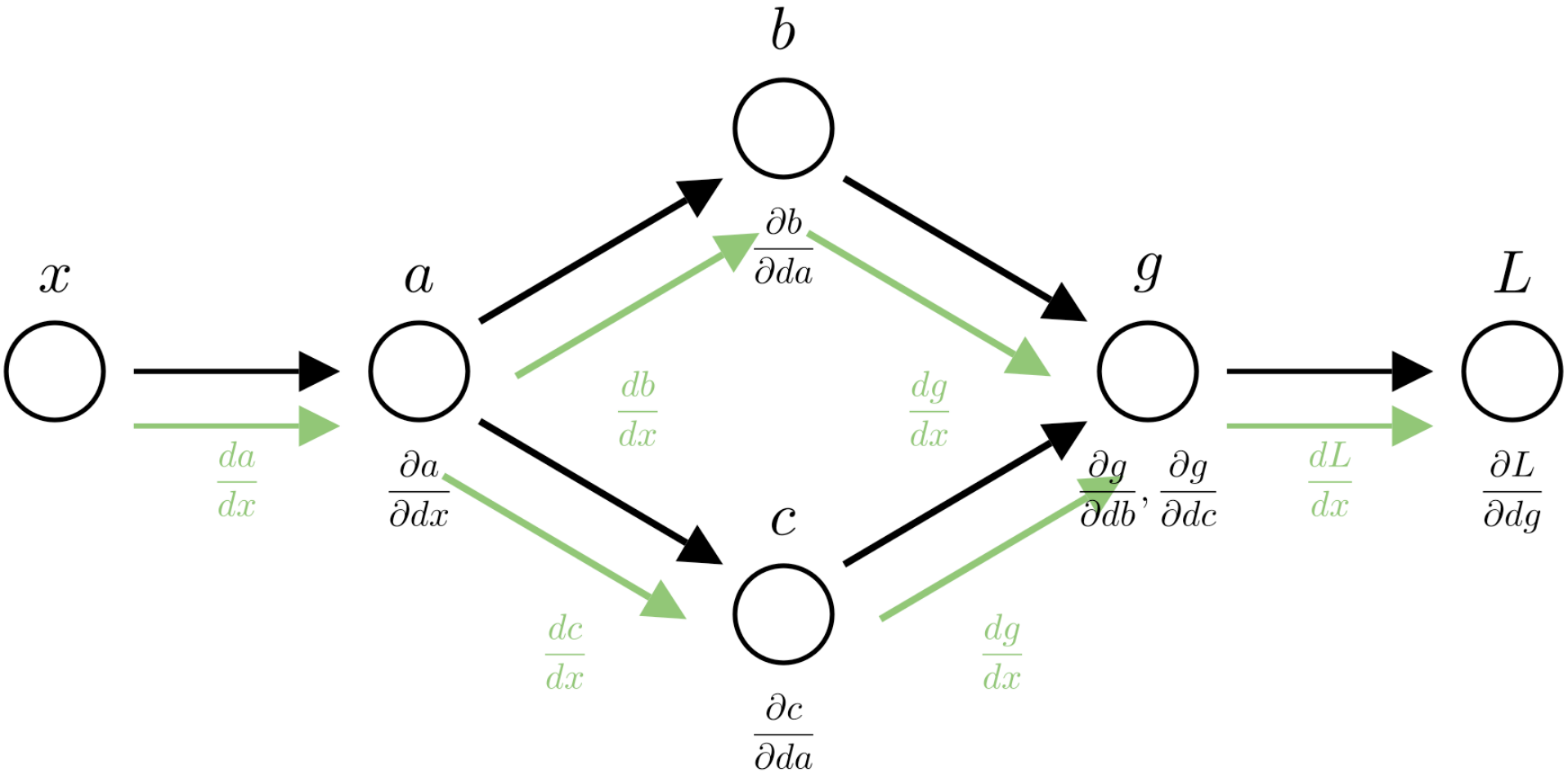
$$c = ab$$

$$L = -c$$

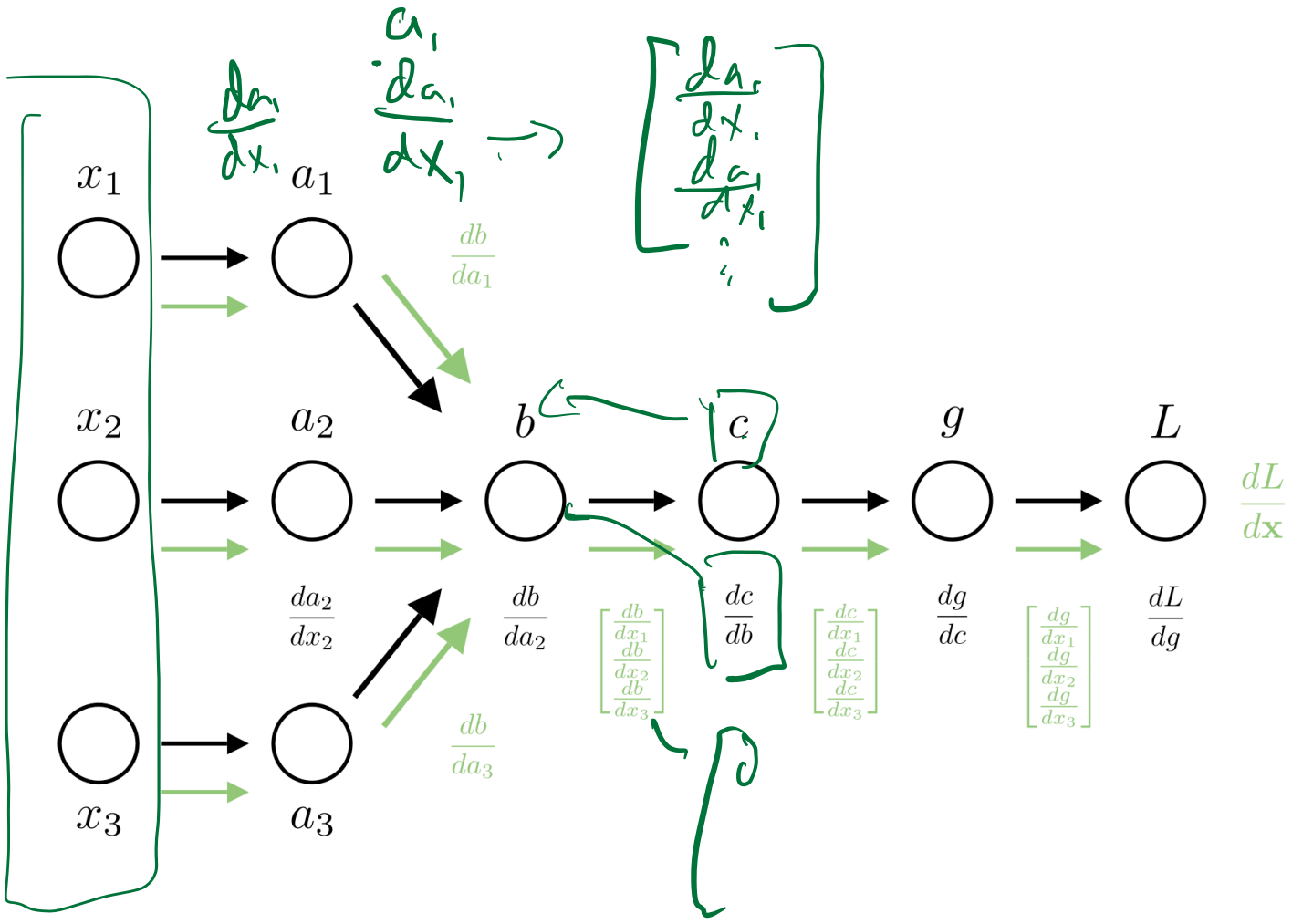


$$\frac{dc}{da} = \frac{\partial c}{\partial a} + \frac{\partial c}{\partial b}\frac{\partial b}{\partial a} = 5a + 5a = 10a$$

Partial derivatives revisited

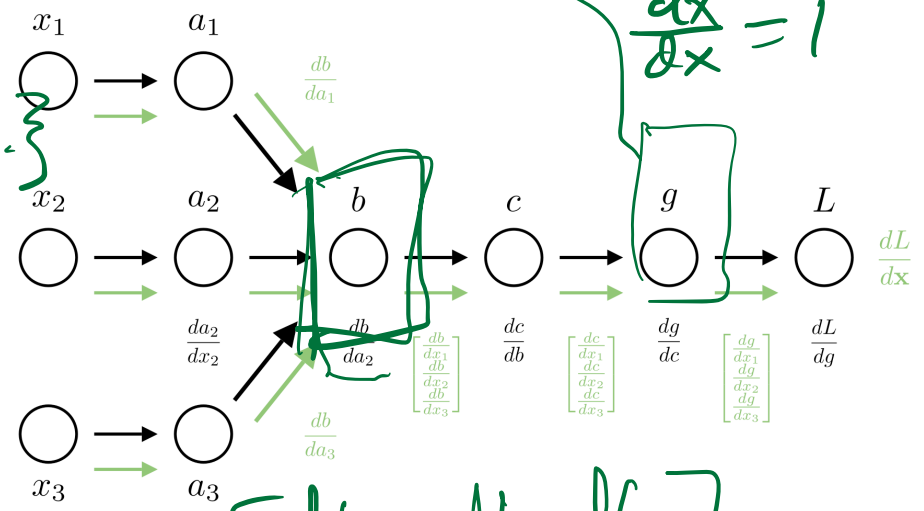


Implementing automatic differentiation



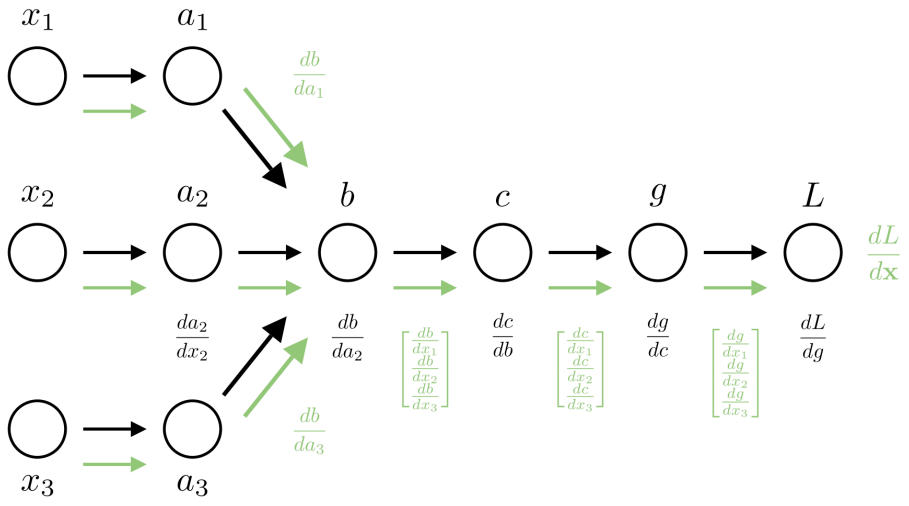
Implementing automatic differentiation

```python
class ForwardValue:
    '''
    Base class for automatic differentiation operations. Represents variable delcaration.
    Subclasses will overwrite func and grads to define new operations.

    Properties:
        parent_values    (list): A list of raw values of each input (as floats)
        value   (float): The value of the result of this operation
        forward_grads (dict): A dictionary mapping inputs to gradients
    '''

    def __init__(self, *args):
        self.parent_values = [arg.value if isinstance(arg, ForwardValue) else arg for arg in args]
        self.value = self.forward_pass(args)

        if len(self.forward_grads.keys()) == 0:
            self.forward_grads = {self: 1}


    def func(self, input):
        '''
        Compute the value of the operation given the inputs.
        For declaring a variable, this is just the identity function (return the input).

        Args:
            input (float): The input to the operation
        Returns:
            value (float): The result of the operation
        '''
        return input

    def grads(self, *args):
        '''
        Compute the derivative of the operation with respect to each input.
        In the base case the derivative of the identity function is just 1. (da/da = 1).

        Args:
            input (float): The input to the operation
        Returns:
            grads (tuple): The derivative of the operation with respect to each input
                           Here there is only a single input, so we return a length-1 tuple.
        '''
        return (1,)

    def forward_pass(self, args):
        # Calls func to compute the value of this operation
        self.forward_grads = {}
        return self.func(*self.parent_values)
```



Implementing automatic differentiation

```python
class _add(ForwardValue):
    # Addition operator (a + b)
    def func(self, a, b):
        return a + b

    def grads(self, a, b):
        return 1., 1.

class _neg(ForwardValue):
    # Negation operator (-a)
    def func(self, a):
        return -a

    def grads(self, a):
        return (-1.,)

class _sub(ForwardValue):
    # Subtraction operator (a - b)
    def func(self, a, b):
        # Your code here

    def grads(self, a, b):
        # Your code here
```



Calculus for vectors (Jacobian)

Now let's consider a vector-valued function $f: \mathbb{R}^n \to \mathbb{R}^m$

e.g. $f(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_m)$

We define the instantaneous rate of change via the **Jacobian matrix**, whose entries are **partial derivatives**

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

The Jacobian is also called the **derivative** of f. It is a $m \times n$ matrix

Each partial derivative fixes the other inputs and treats f as scalar-input scalar-output

Calculus for vectors (Jacobian)

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = A$$

$$\frac{\partial y_2}{\partial x_3} = A_{32}$$

$$\frac{\partial}{\partial x_3} y_2 = x_1 A_{12} + x_2 A_{22} + x_3 A_{32} + \cdots$$

$$A_{32}$$

Calculus for vectors (Jacobian)

$$\mathbf{y} = \mathbf{x}^2$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} =$$

Jacobians and gradients

$f: \mathbb{R}^n \rightarrow \mathbb{R}^m$

When $m = 1$, (scalar-valued output) the Jacobian is called the **gradient**

$$\frac{\partial f}{\partial x} = \left[\frac{\partial y}{\partial x_1} \quad \frac{\partial y}{\partial x_2} \quad \dots \quad \frac{\partial y}{\partial x_n}\right] = \nabla_{\vec{x}} f$$

Notes:

- The Jacobian of a scalar-valued function is a row vector.
- The gradient is often described as a column vector

Chain rule for Jacobians

An easy extension of the chain rule exists:

Given $f: \mathbb{R}^k \to \mathbb{R}^m$ and $g: \mathbb{R}^n \to \mathbb{R}^k$

$$\underbrace{\frac{\partial f\big(g(x)\big)}{\partial x}}_{m \times n} = \underbrace{\frac{\partial f(y)}{\partial y}}_{m \times k} \cdot \underbrace{\frac{\partial g(x)}{\partial x}}_{k \times n}$$

Where $y = g(x)$.

Forward-mode with vectors

$$\mathbf{a} = \mathbf{X}\mathbf{w}$$

$$\mathbf{b} = \mathbf{y} - \mathbf{a}$$

$$\mathbf{c} = \mathbf{b}^2$$

$$g = \sum_{i=1}^{N} c_i$$

$$L = \frac{1}{N} g$$

Forward-mode with vectors

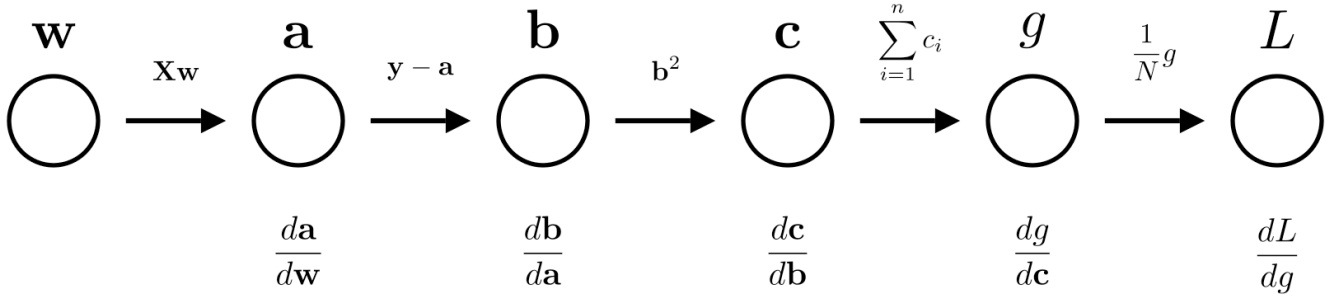
$$\mathbf{a} = \mathbf{X}\mathbf{w}$$

$$\mathbf{b} = \mathbf{y} - \mathbf{a}$$

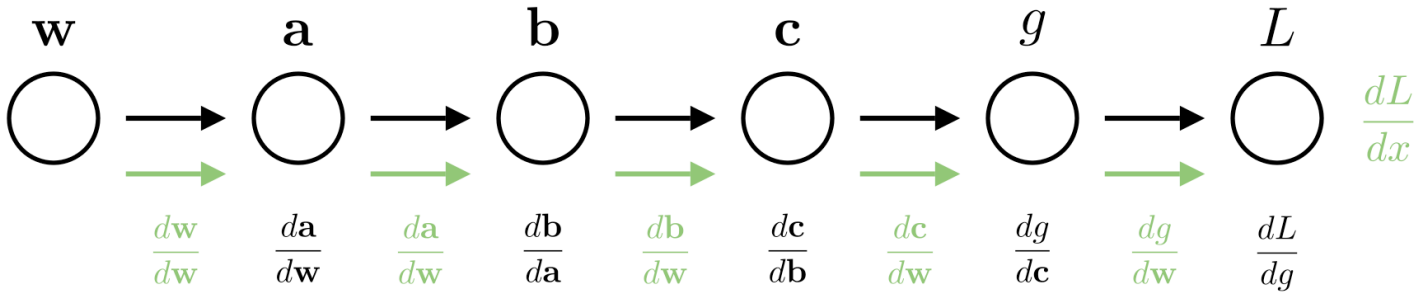
$$\mathbf{c} = \mathbf{b}^2$$

$$g = \sum_{i=1}^{N} c_i$$

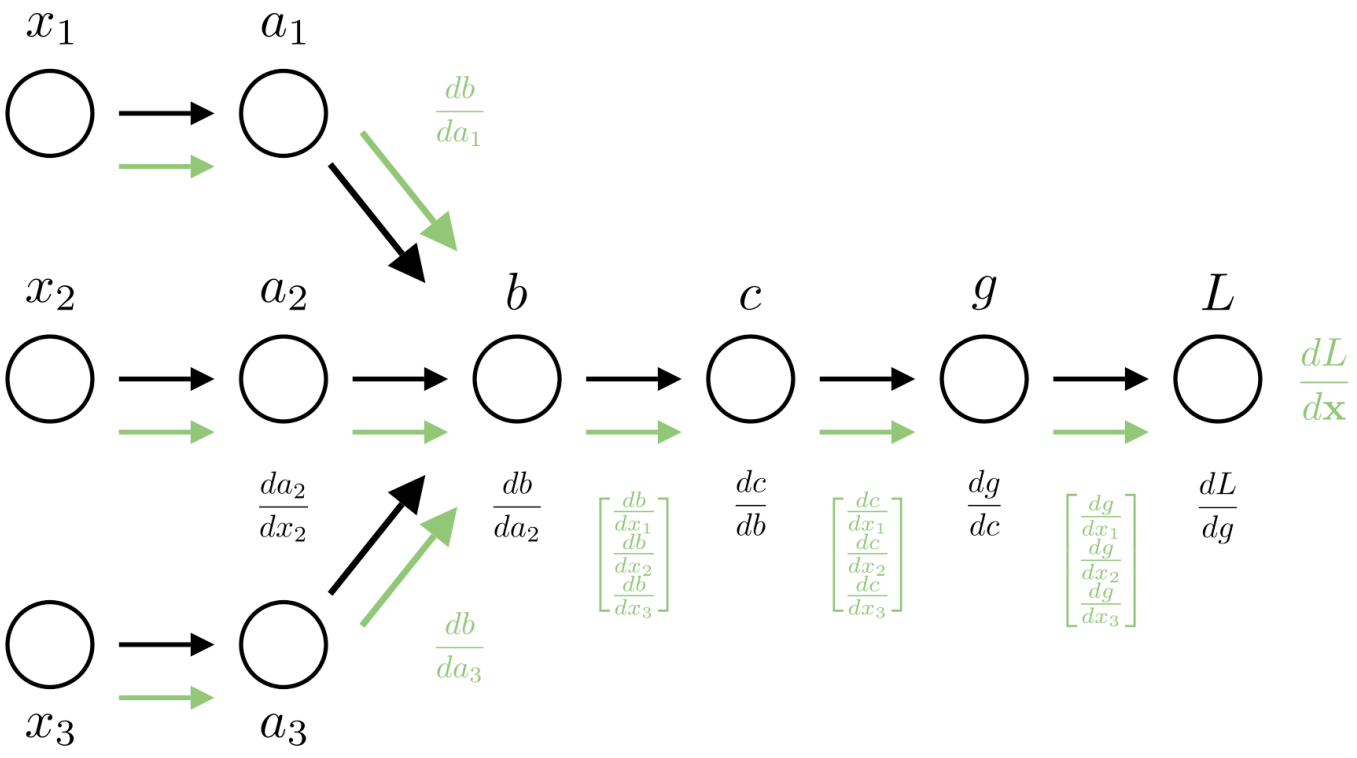
$$L = \frac{1}{N} g$$



Forward mode

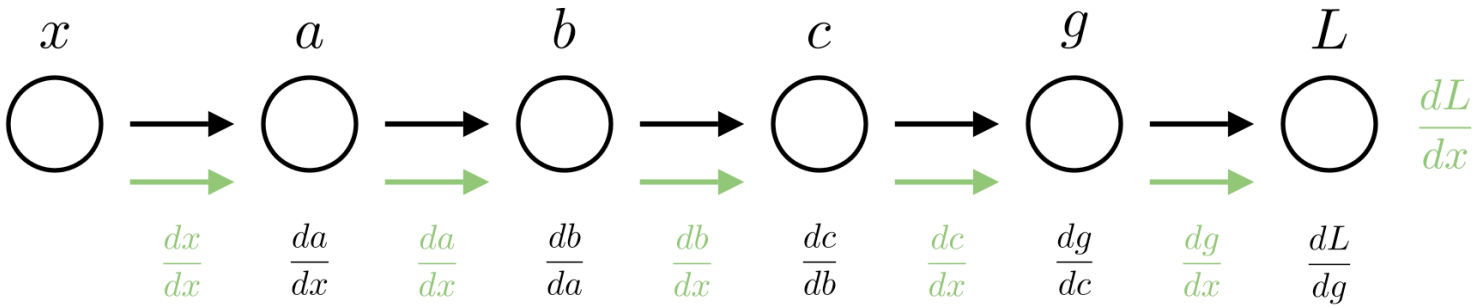


Forward-mode with vectors

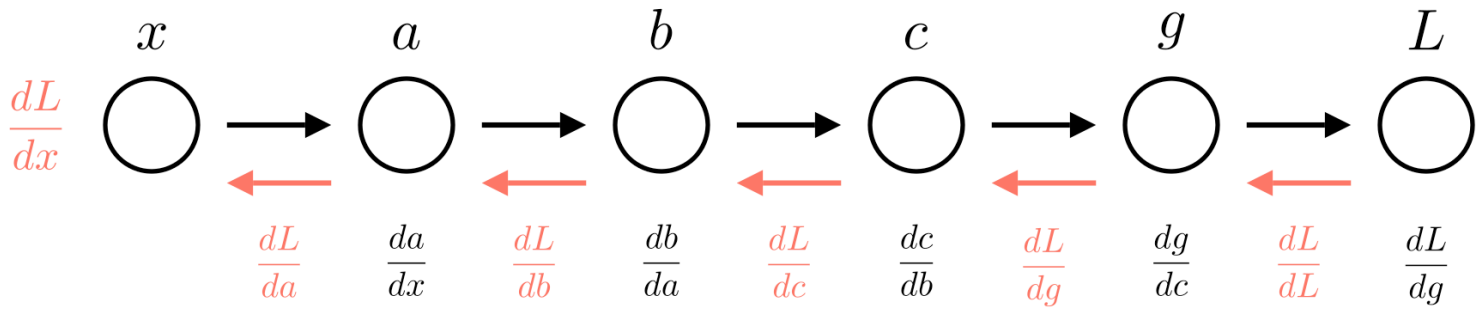


Reverse-mode AD

Forward mode



Reverse mode



Reverse-mode AD