Automatic Differentiation (1)

Slides Prepared By:

Atılım Güneş Baydin
gunes@robots.ox.ac.uk

Outline

This lecture:

- Derivatives in machine learning
- Review of essential concepts (what is a derivative, Jacobian, etc.)
- How do we compute derivatives
- Automatic differentiation

Next lecture:

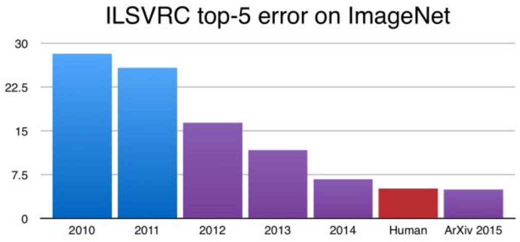
- Current landscape of tools
- Implementation techniques
- Advanced concepts (higher-order API, checkpointing, etc.)

Derivatives and machine learning

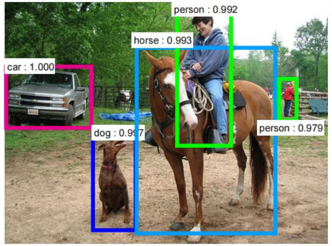
Derivatives in machine learning

"Backprop" and gradient descent are at the core of all recent advances

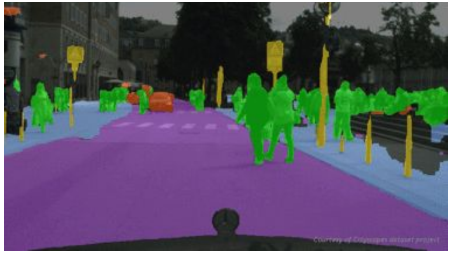
Computer vision



Top-5 error rate for ImageNet (NVIDIA devblog)

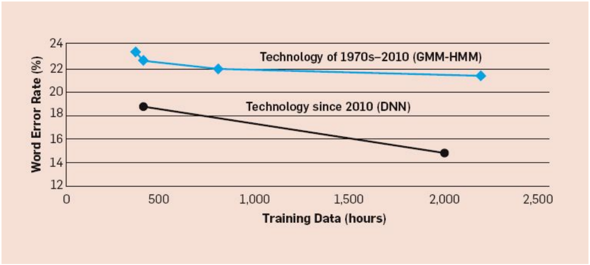


Faster R-CNN (Ren et al. 2015)



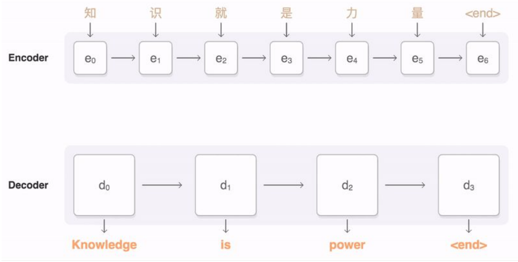
NVIDIA DRIVE PX 2 segmentation

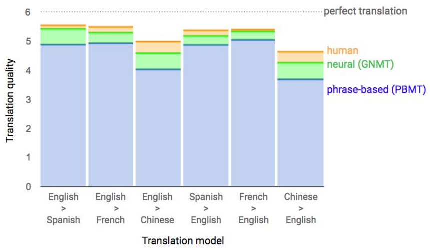
Speech recognition/synthesis



Word error rates (Huang et al., 2014)

Machine translation





Google Neural Machine Translation System (GNMT)

Derivatives in machine learning

"Backprop" and gradient descent are at the core of all recent advances

Probabilistic programming (and modeling)

Pyro (2017)

ProbTorch (2017)

Edward (2016)

TensorFlow Probability (2018)

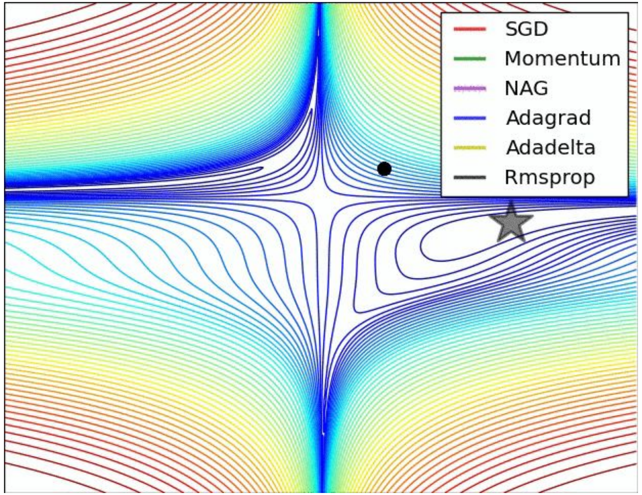
- Variational inference
- "Neural" density estimation
    - Transformed distributions via bijectors
    - Normalizing flows (Rezende & Mohamed, 2015)
    - Masked autoregressive flows (Papamakarios et al., 2017)

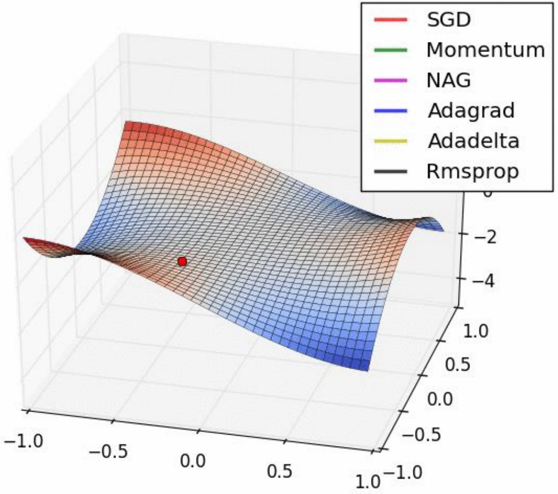
Derivatives in machine learning

At the core of all: **differentiable functions (programs)** whose parameters are tuned by **gradient-based optimization**

$$Q(\boldsymbol{w}) = \sum_{i=1}^{N} Q_i(\boldsymbol{w})$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \sum_{i=1}^{d} \nabla_{\boldsymbol{w}} Q_i(\boldsymbol{w})$$





(Ruder, 2017) http://ruder.io/optimizing-gradient-descent/

Automatic differentiation

Execute **differentiable functions (programs)** via **automatic differentiation**

A word on naming:

- Differentiable programming, a generalization of deep learning (Olah, LeCun)
  "Neural networks are just a class of differentiable functions"
- Automatic differentiation
- Algorithmic differentiation
- AD
- Autodiff
- Algodiff
- Autograd

Also remember:

- Backprop
- Backpropagation (backward propagation of errors)

Essential concepts refresher

Derivative

Function of a real variable $f : \mathbb{R} \to \mathbb{R}$

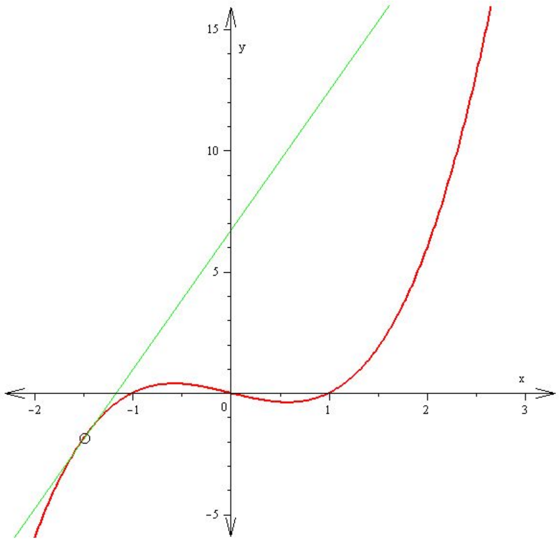
Sensitivity of function value w.r.t.
a change in its argument
(the instantaneous rate of change)

Dependent Independent

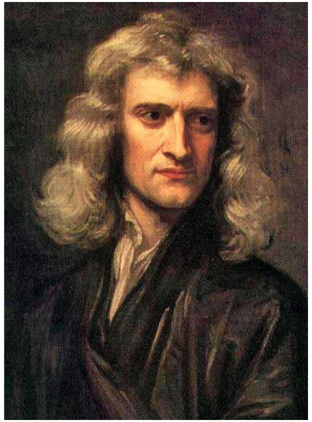
$$y = f(x)$$

$$\frac{dy}{dx} = f'(x) = \dot{y}$$

Leibniz Lagrange Newton



$$\lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$



Newton, c. 1665

Leibniz, c. 1675

Derivative

Function of a real variable $f : \mathbb{R} \to \mathbb{R}$

General Formulas

1. $$\frac{d}{dx}c = 0$$

2. $$\frac{d}{dx}[f(x) \mp g(x)] = f'(x) \mp g'(x)$$

3. $$\frac{d}{dx}[f(x)g(x)] = f'(x)g(x) + g'(x) + f(x)$$

4. $$\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{g(x)f'(x) - g'(x)f(x)}{(g(x))^2}$$

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

6. $$\frac{d}{dx}x^n = nx^{n-1}$$

Exponential and Logarithmic Functions

7. $$\frac{d}{dx}e^{f(x)} = f'(x)e^{f(x)}$$

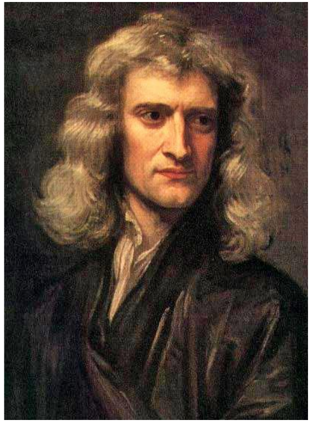
8. $$\frac{d}{dx}a^x = a^x \ln(a)$$

9. $$\frac{d}{dx}\ln(C|f(x)|) = \frac{d}{dx}[\ln(C) + \ln(f(x))] = \frac{f'(x)}{f(x)}$$

...

around 15 such rules

Note: the derivative is a linear operator, a.k.a. a **higher-order function** in programming languages $(\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$

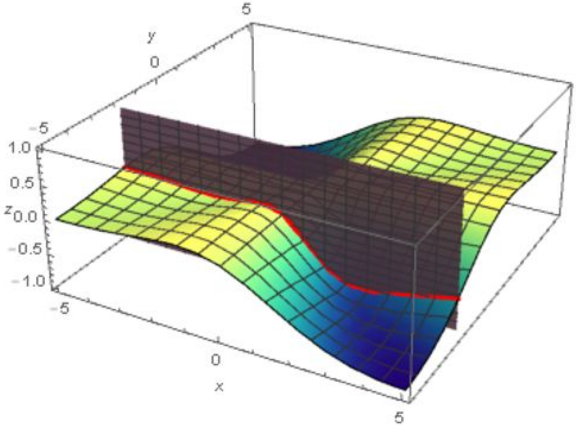


Newton, c. 1665

Leibniz, c. 1675

Partial derivative

Function of several real variables $f : \mathbb{R}^n \to \mathbb{R}$



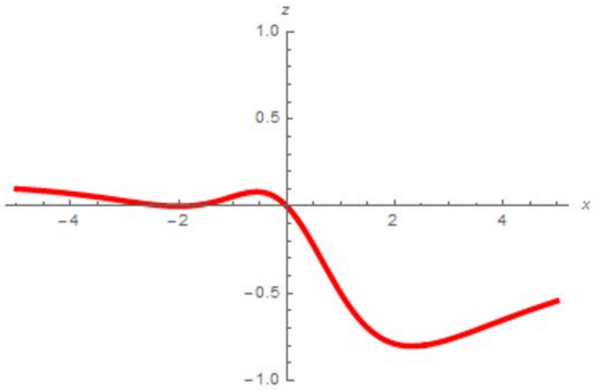
A derivative w.r.t. one independent variable,
with others held constant

$$z = f(x, y) = x^2 + xy + y^2$$

“del”

$$\frac{\partial z}{\partial x} = 2x + y$$

$$\frac{\partial z}{\partial y} = 2y + x$$



Partial derivative

Function of several real variables $f : \mathbb{R}^n \to \mathbb{R}$

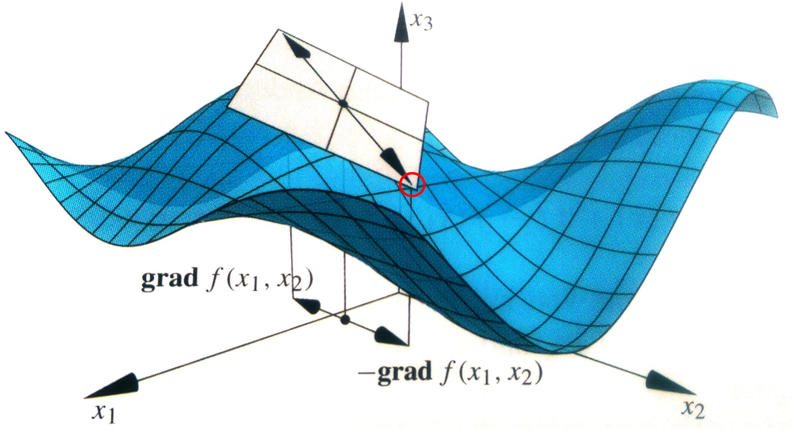
The gradient, given

$$f(\mathbf{x}),\ \mathbf{x} \in \mathbf{R}^n$$

is the vector of all partial derivatives

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

"nabla"
or "del"



$\nabla f(\mathbf{x})$ points to the direction with the largest rate of change

Nabla is the higher-order function: $(\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}^n)$

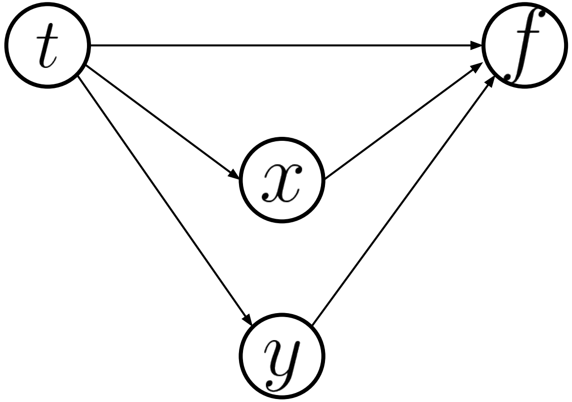
Total derivative

Function of several real variables $f : \mathbb{R}^n \to \mathbb{R}$

The derivative **w.r.t. all variables** (independent & dependent)

$$f(t, x(t), y(t))$$

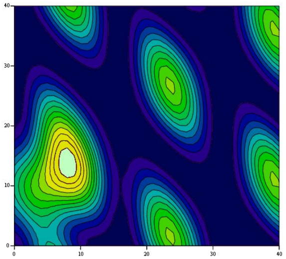
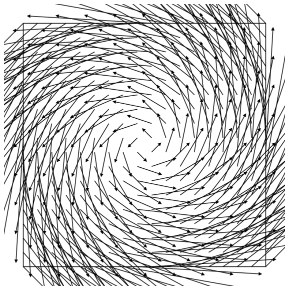
$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$$



Consider all partial derivatives simultaneously and **accumulate all direct and indirect contributions** (Important: will be useful later)

Matrix calculus and machine learning

Extension to multivariable functions

	Scalar output	Vector output
Scalar input	$f : \mathbb{R} \to \mathbb{R}$	$\mathbf{f} : \mathbb{R} \to \mathbb{R}^m$
Vector input	$f : \mathbb{R}^n \to \mathbb{R}$  scalar field	$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$  vector field

In machine learning, we construct (deep) **compositions** of

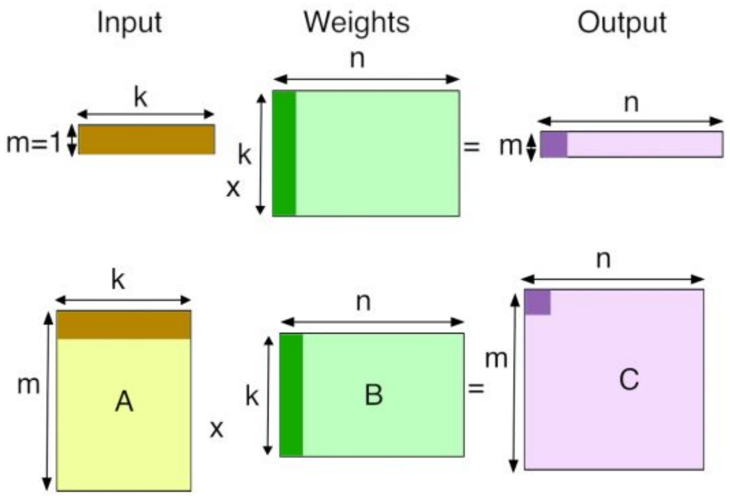
- $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$, e.g., a neural network
- $f : \mathbb{R}^n \to \mathbb{R}$, e.g., a loss function, KL divergence, or log joint probability

Matrix calculus and machine learning

Differential identities: matrix [1][5]

Condition	Expression	Result (numerator layout)
A is not a function of **X**	$d(\mathbf{A}) =$	0
a is not a function of **X**	$d(a\mathbf{X}) =$	$a\,d\mathbf{X}$
	$d(\mathbf{X}+\mathbf{Y}) =$	$d\mathbf{X} + d\mathbf{Y}$
	$d(\mathbf{X}\mathbf{Y}) =$	$(d\mathbf{X})\mathbf{Y} + \mathbf{X}(d\mathbf{Y})$
(Kronecker product)	$d(\mathbf{X} \otimes \mathbf{Y}) =$	$(d\mathbf{X}) \otimes \mathbf{Y} + \mathbf{X} \otimes (d\mathbf{Y})$
(Hadamard product)	$d(\mathbf{X} \circ \mathbf{Y}) =$	$(d\mathbf{X}) \circ \mathbf{Y} + \mathbf{X} \circ (d\mathbf{Y})$
	$d(\mathbf{X}^\top) =$	$(d\mathbf{X})^\top$
	$d(\mathbf{X}^{-1}) =$	$-\mathbf{X}^{-1}(d\mathbf{X})\mathbf{X}^{-1}$
(conjugate transpose)	$d(\mathbf{X}^{\mathrm{H}}) =$	$(d\mathbf{X})^{\mathrm{H}}$

And many, many more rules



Generalization to **tensors (multi-dimensional arrays)** for efficient batching, handling of sequences, channels in convolutions, etc.

Matrix calculus and machine learning

Finally, two constructs relevant to machine learning: Jacobian and Hessian

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix}$$

$$(\mathbb{R}^n \to \mathbb{R}^m) \to (\mathbb{R}^n \to \mathbb{R}^{m \times n})$$

$$\mathbf{H}_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

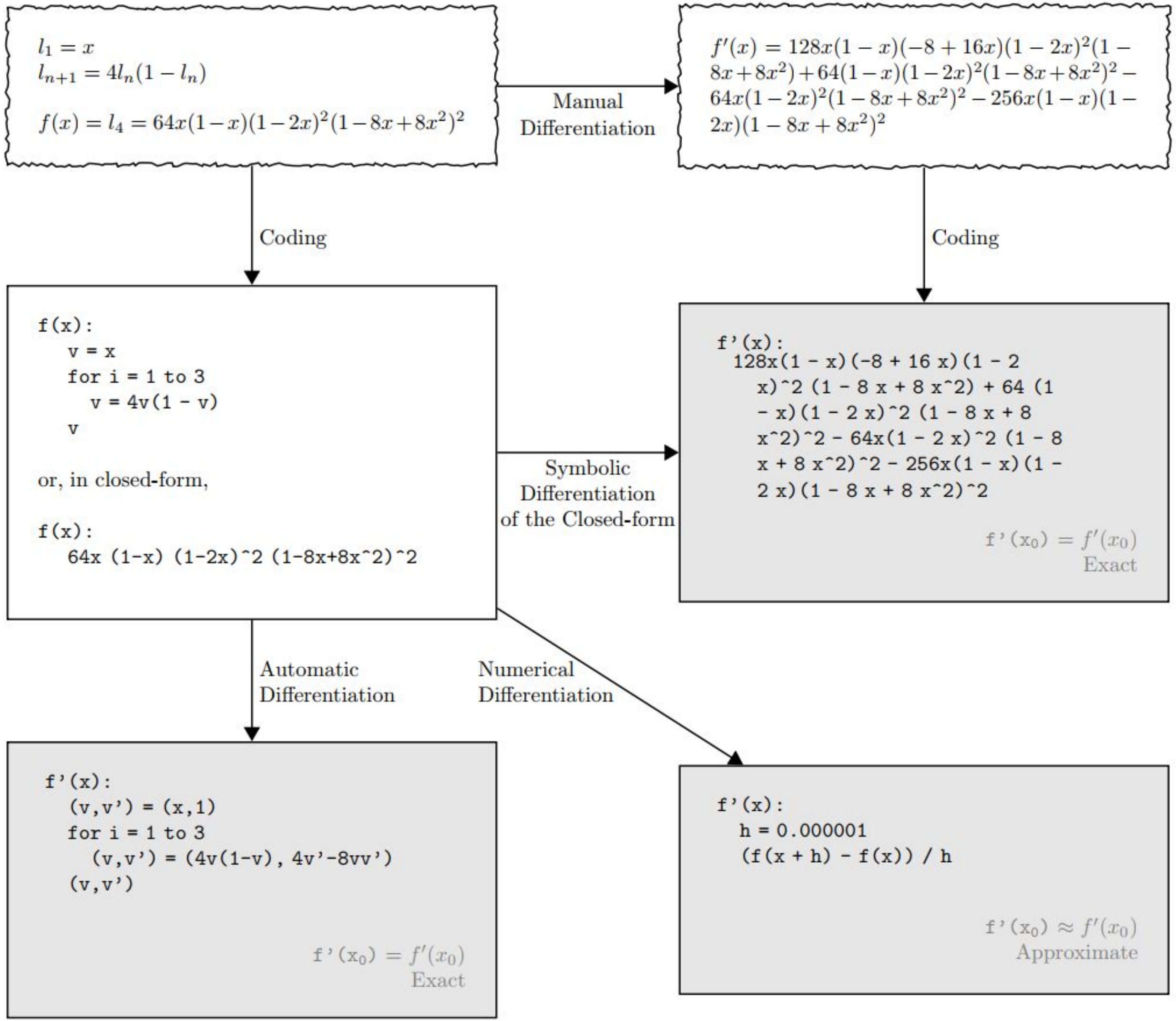
$$(\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R}^{n \times n})$$

How to compute derivatives

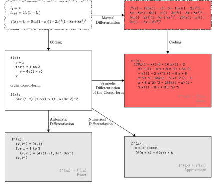
Derivatives as code

We can compute the derivatives **not just of mathematical functions, but of general programs** (with control flow)

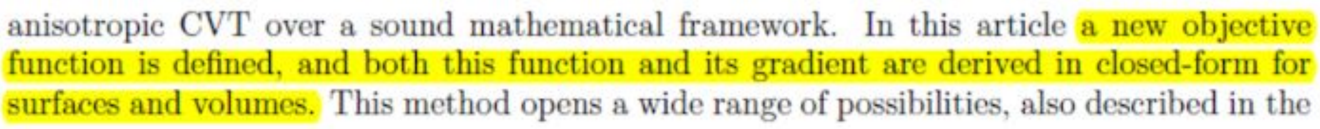
Derivatives as code

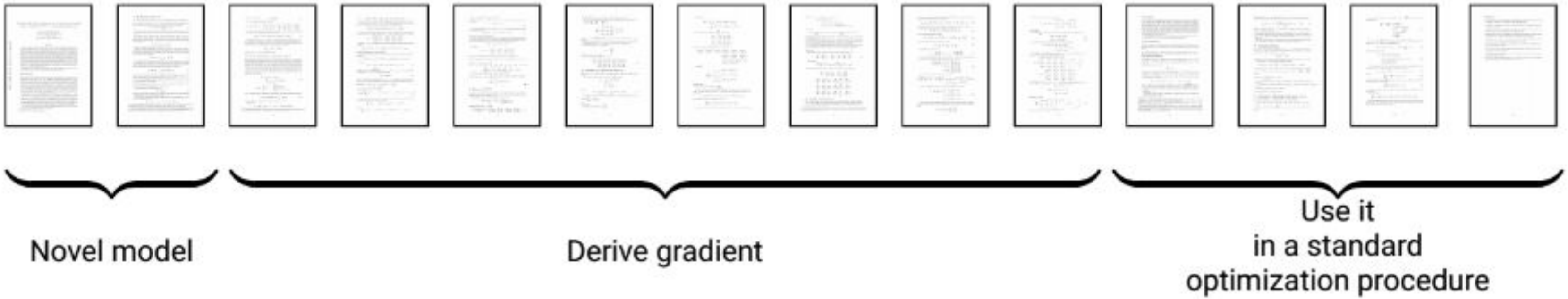


Manual



You can see papers like this:


anisotropic CVT over a sound mathematical framework. In this article a new objective function is defined, and both this function and its gradient are derived in closed-form for surfaces and volumes. This method opens a wide range of possibilities, also described in the

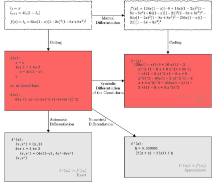


Analytic derivatives are needed for **theoretical insight**

- analytic solutions, proofs
- mathematical analysis, e.g., stability of fixed points

Unnecessary when we just need derivative evaluations for optimization

Symbolic differentiation

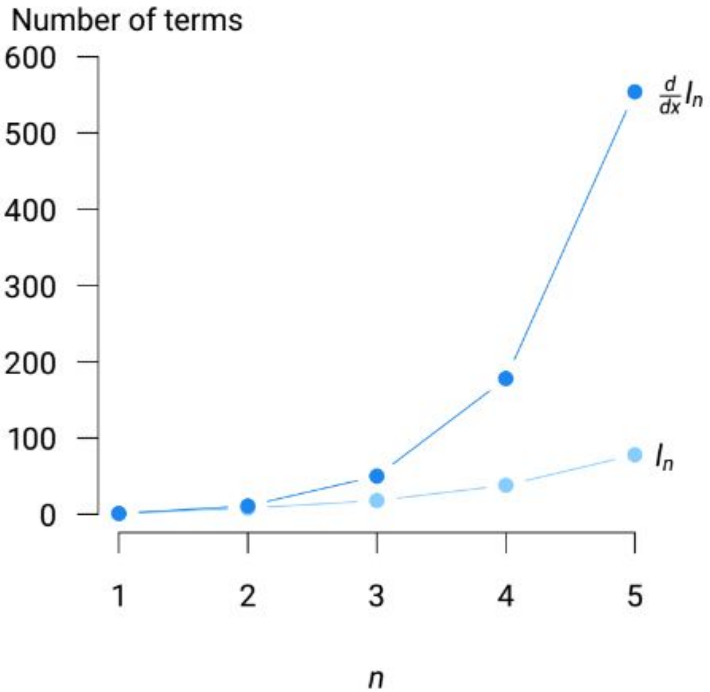


Symbolic computation with Mathematica, Maple, Maxima, and deep learning frameworks such as Theano

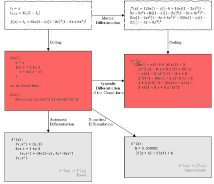
Problem: expression swell

Logistic map $l_{n+1} = 4l_n(1 - l_n)$, $l_1 = x$

n	l_n	$\frac{d}{dx} l_n$
1	x	1
2	$4x(1-x)$	$4(1-x) - 4x$
3	$16x(1-x)(1-2x)^2$	$16(1-x)(1-2x)^2 - 16x(1-2x)^2 - 64x(1-x)(1-2x)$
4	$64x(1-x)(1-2x)^2 (1-8x+8x^2)^2$	$128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$



Symbolic differentiation



Symbolic computation with Mathematica, Maple, Maxima,
and deep learning frameworks such as Theano

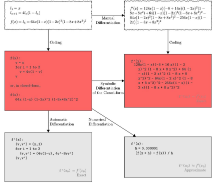
Problem: expression swell

Graph optimization
(e.g., in Theano)

Logistic map $l_{n+1} = 4l_n(1 - l_n)$, $l_1 = x$

n	l_n	$\frac{d}{dx} l_n$	$\frac{d}{dx} l_n$ (Simplified form)
1	x	1	1
2	$4x(1-x)$	$4(1-x) - 4x$	$4 - 8x$
3	$16x(1-x)(1-2x)^2$	$16(1-x)(1-2x)^2 - 16x(1-2x)^2 - 64x(1-x)(1-2x)$	$16(1 - 10x + 24x^2 - 16x^3)$
4	$64x(1-x)(1-2x)^2 (1-8x+8x^2)^2$	$128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$	$64(1 - 42x + 504x^2 - 2640x^3 + 7040x^4 - 9984x^5 + 7168x^6 - 2048x^7)$

Symbolic differentiation



Problem: only applicable to **closed-form mathematical functions**

You can find the derivative of

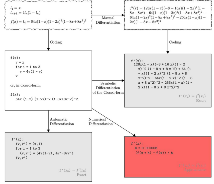
```
In [1]: def f(x):
            return 64 *(1-x) *(1-2*x)^2 *(1-8*x+8*x*x)^2
```

but not of

```
In [2]: def f(x,n):
            if n == 1:
                return x
            else:
                v = x
                for i in range(1,n):
                    v = 4*v*(1-v)
                return v
```

Symbolic graph builders such as Theano and TensorFlow
have limited, unintuitive control flow, loops, recursion

Numerical differentiation

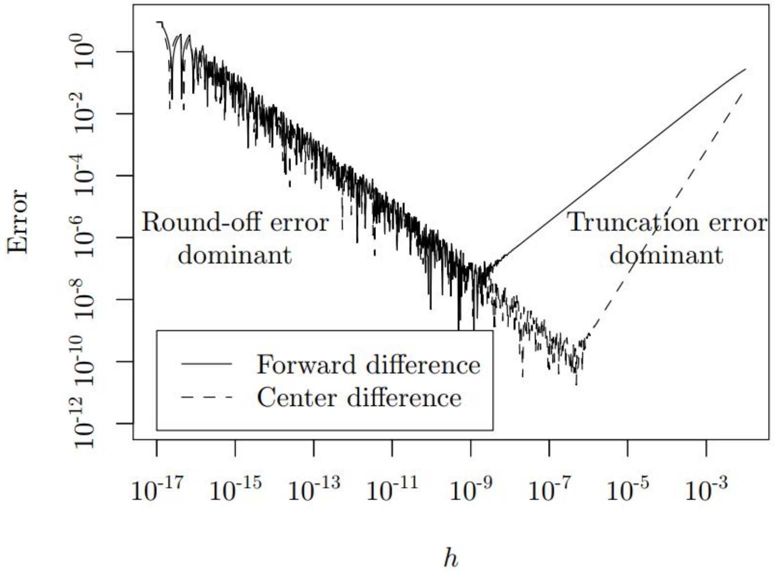


Finite difference approximation of $\nabla f, \; f : \mathbb{R}^n \to \mathbb{R}$

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}, \qquad 0 < h \ll 1$$

Problem: needs to be evaluated n times, once with each standard basis vector $\mathbf{e}_i \in \mathbb{R}^n$

Problem: we must select h and we face **approximation errors**

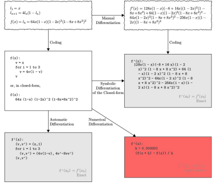


$$E(h, x^*) = \left| \frac{f(x^* + h) - f(x^*)}{h} - \frac{d}{dx} f(x) \Big|_{x^*} \right|$$

$$f(x) = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

$$x^* = 0.2$$

Numerical differentiation



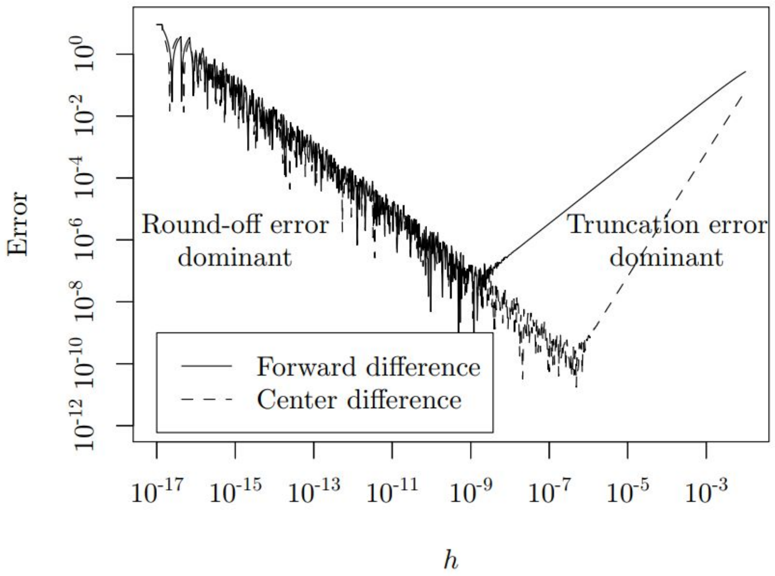
Finite difference approximation of $\nabla f, \ f : \mathbb{R}^n \to \mathbb{R}$

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}, \quad 0 < h \ll 1$$

Better approximations exist:

- Higher-order finite differences
  e.g., center difference:
  $$\frac{\partial f(\mathbf{x})}{\partial x_i} = \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h} + O(h^2)$$
- Richardson extrapolation
- Differential quadrature

These increase rapidly in complexity
and **never completely eliminate the error**



$$E(h, x^*) = \left| \frac{f(x^* + h) - f(x^*)}{h} - \frac{d}{dx} f(x)\big|_{x^*} \right|$$

$$f(x) = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

$$x^* = 0.2$$

Numerical differentiation

Finite difference approximation of $\nabla f, \; f : \mathbb{R}^n \to \mathbb{R}$

Still extremely useful as a **quick check of our gradient implementations**

Good to learn:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} - h\mathbf{e}_i)}{2h} + O(h^2)$$

Bet

-

-

- Differential quadrature

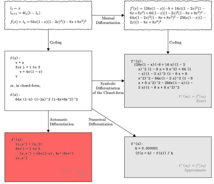
These increase rapidly in complexity and **never completely eliminate the error**

$$E(h, x^*) = \left| \frac{f(x^* + h) - f(x^*)}{h} - \frac{d}{dx} f(x)|_{x^*} \right|$$

$$f(x) = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

$$x^* = 0.2$$

Automatic differentiation



If we don't need analytic derivative expressions, we can **evaluate a gradient exactly** with only one forward and one reverse execution

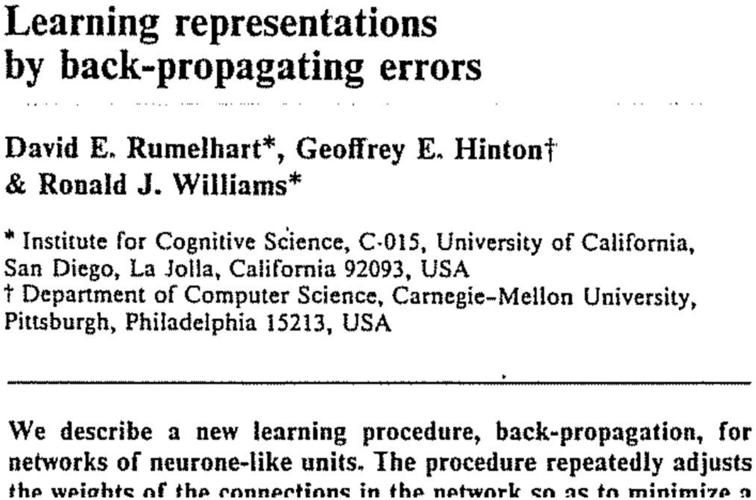
$$f : \mathbb{R}^n \to \mathbb{R} \qquad \nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

In machine learning, this is known as **backpropagation** or "backprop"

- Automatic differentiation is more than backprop
- Or, backprop is a specialized *reverse mode* automatic differentiation
- We will come back to this shortly

nature
International journal of science

Nature 323, 533–536 (9 October 1986)



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a

Backprob or automatic differentiation?

1960s	1970s	1980s
Precursors	**Linnainmaa, 1970, 1976** ***Backpropagation***	**Speelpenning, 1980** ***Automatic reverse mode***
Kelley, 1960 Bryson, 1961 Pontryagin et al., 1961 Dreyfus, 1962	Dreyfus, 1973 *Control parameters*	Werbos, 1982 *First NN-specific backprop*
Wengert, 1964 ***Forward mode***	Werbos, 1974 *Reverse mode*	Parker, 1985
		LeCun, 1985
		Rumelhart, Hinton, Williams, 1986 ***Revived backprop***
		Griewank, 1989 ***Revived reverse mode***

1960s | 1970s | 1980s

Precursors

Linnainmaa, 1970, 1976

Speelpenning, 1980

Recommended reading:

Griewank**, A., 2012. **Who Invented the Reverse Mode of Differentiation?
Documenta Mathematica, Extra Volume ISMP, pp.389-400.

Schmidhuber**, J., 2015. **Who Invented Backpropagation?
[*http://people.idsia.ch/~juergen/who-invented-backpropagation.html*](http://people.idsia.ch/~juergen/who-invented-backpropagation.html)

Rumelhart, Hinton, Williams, 1986
Revived backprop

Griewank, 1989
Revived reverse mode

Automatic differentiation

Automatic differentiation

All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives

- Called a **trace** or a **Wengert list** (Wengert,1964)
- Alternatively represented as a **computational graph** showing dependencies

Automatic differentiation

All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives

- Called a **trace** or a **Wengert list** (Wengert,1964)
- Alternatively represented as a **computational graph** showing dependencies

$$f(a,b) = \log(ab)$$

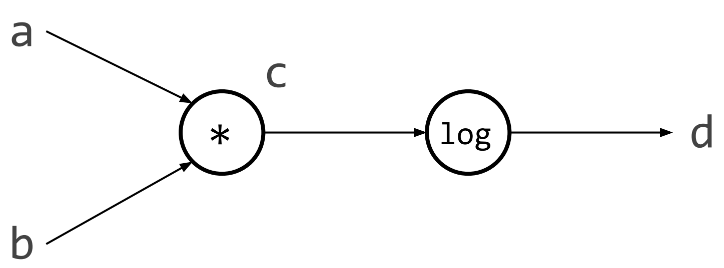
$$\nabla f(a,b) = (1/a, 1/b)$$

Automatic differentiation

All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives

- Called a **trace** or a **Wengert list** (Wengert,1964)
- Alternatively represented as a **computational graph** showing dependencies

```
f(a, b):
  c = a * b
  d = log(c)
  return d
```

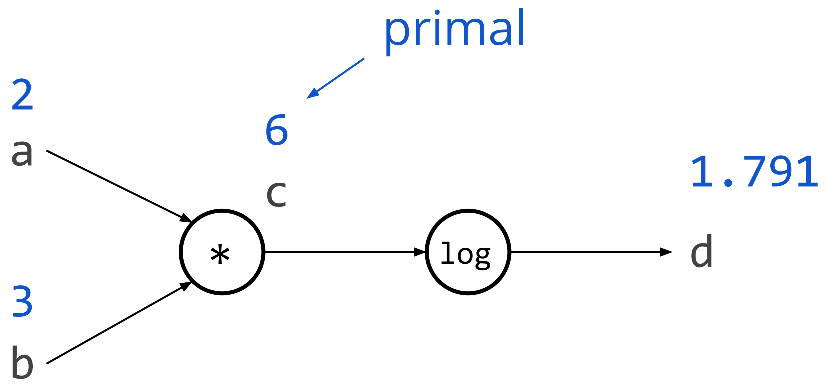


Automatic differentiation

All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives

- Called a **trace** or a **Wengert list** (Wengert,1964)
- Alternatively represented as a **computational graph** showing dependencies

```
f(a, b):
  c = a * b
  d = log(c)
  return d

1.791 = f(2, 3)
```

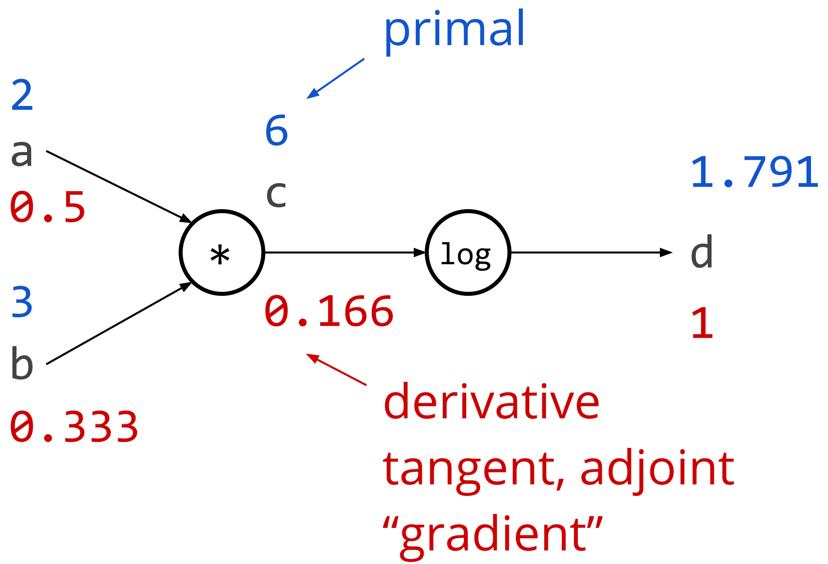


Automatic differentiation

All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives

- Called a **trace** or a **Wengert list** (Wengert,1964)
- Alternatively represented as a **computational graph** showing dependencies

```
f(a, b):
  c = a * b
  d = log(c)
  return d

1.791 = f(2, 3)
[0.5, 0.333] = f'(2, 3)
```



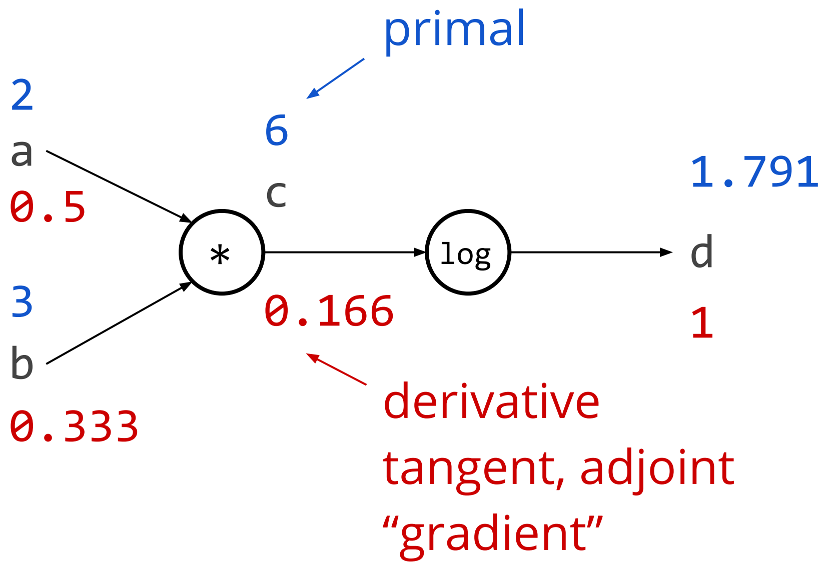
Automatic differentiation

All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives

- Called a **trace** or a **Wengert list** (Wengert,1964)
- Alternatively represented as a **computational graph** showing dependencies

```
f(a, b):
  c = a * b
  d = log(c)
  return d

1.791 = f(2, 3)
[0.5, 0.333] = f'(2, 3)
```

$$\nabla f(a,b) = (1/a, 1/b)$$



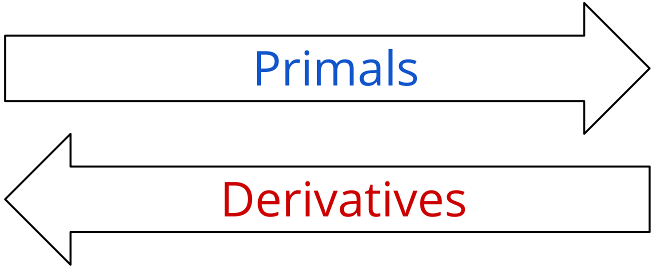
Automatic differentiation

Two main flavors

Forward mode

(Tangents)

Reverse mode (a.k.a. backprop)



(Adjoints)

Nested combinations

(higher-order derivatives, Hessian–vector products, etc.)

- Forward-on-reverse
- Reverse-on-forward
- …

What happens to control flow?

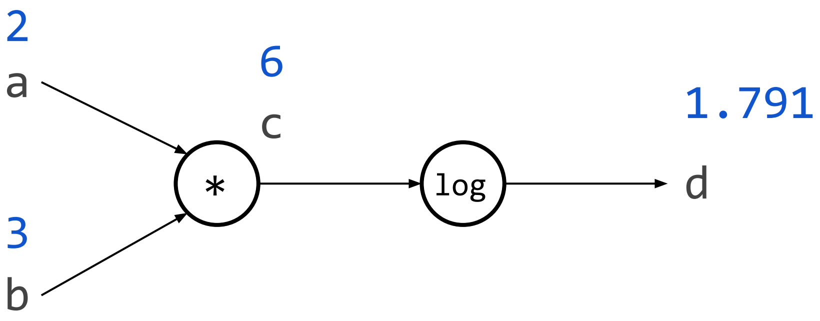
It disappears: branches are taken, loops are unrolled, functions are inlined, etc. until we are left with the linear trace of execution

```
f(a, b):
  c = a * b
  if c > 0:
    d = log(c)
  else:
    d = sin(c)
  return d
```

What happens to control flow?

It disappears: branches are taken, loops are unrolled, functions are inlined, etc. until we are left with the linear trace of execution

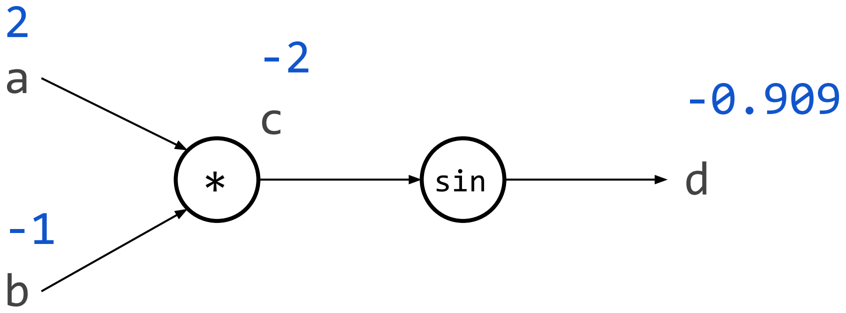
```
f(a = 2, b = 3):
  c = a * b = 6
  if c > 0:
    d = log(c) = 1.791
  else:
    d = sin(c)
  return d
```



What happens to control flow?

It disappears: branches are taken, loops are unrolled, functions are inlined, etc. until we are left with the linear trace of execution

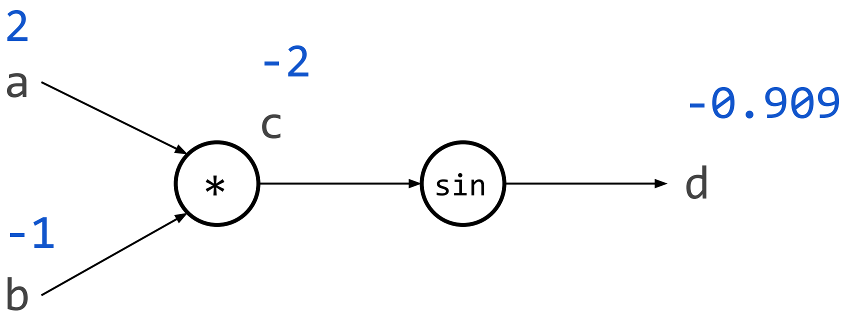
```
f(a = 2, b = -1):
  c = a * b = -2
  if c > 0:
    d = log(c)
  else:
    d = sin(c) = -0.909
  return d
```



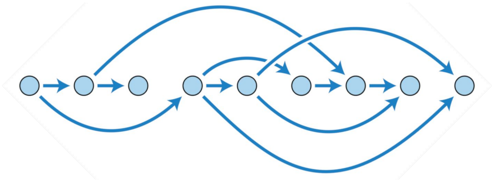
What happens to control flow?

It disappears: branches are taken, loops are unrolled, functions are inlined, etc. until we are left with the linear trace of execution

```
f(a = 2, b = -1):
  c = a * b = -2
  if c > 0:
    d = log(c)
  else:
    d = sin(c) = -0.909
  return d
```



A directed acyclic graph (DAG)



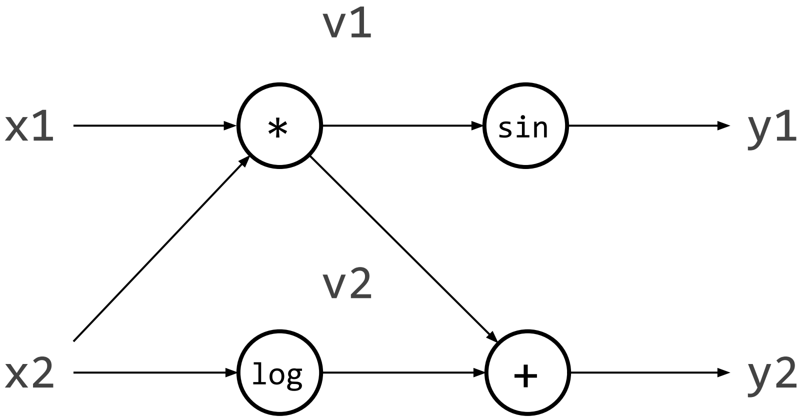
Topological ordering

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)
```



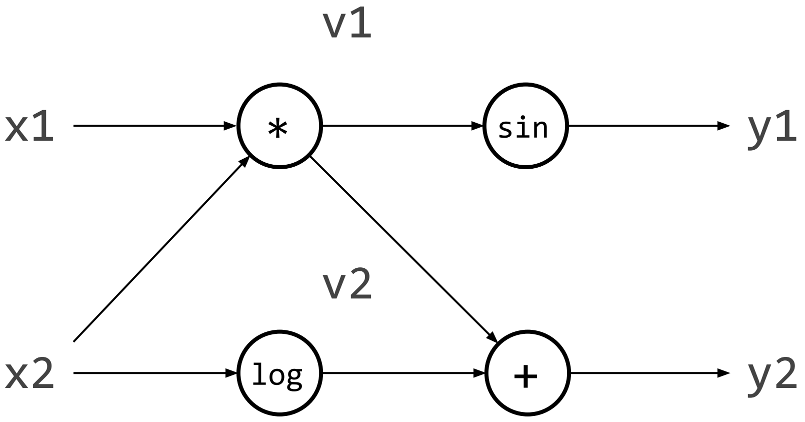
Forward mode

Primals: independent → dependent

Derivatives (tangents): independent → dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



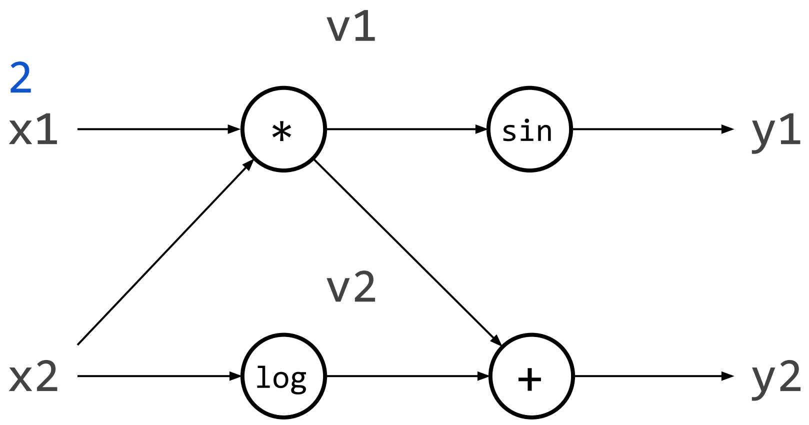
Forward mode

Primals: independent → dependent

Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

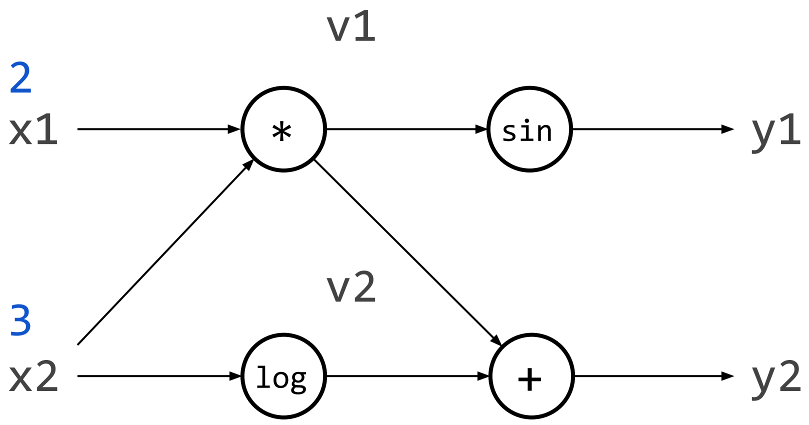


Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

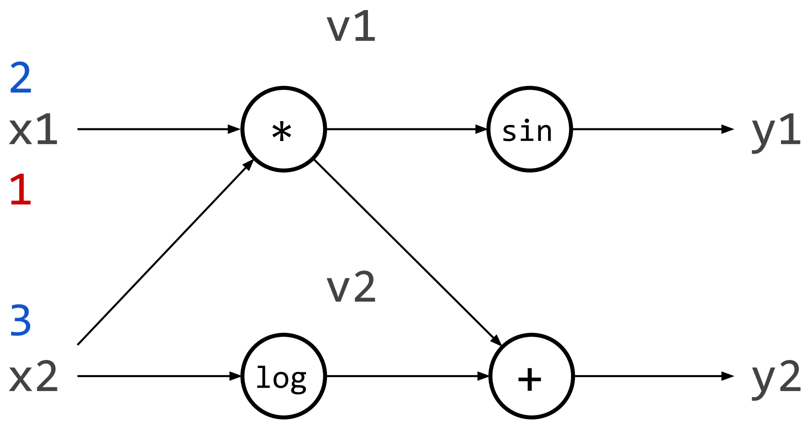


Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



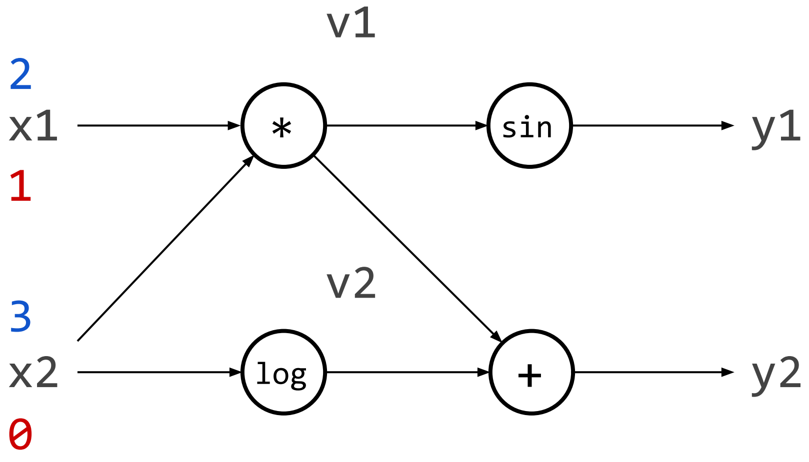
$$\frac{\partial x_1}{\partial x_1} = 1$$

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



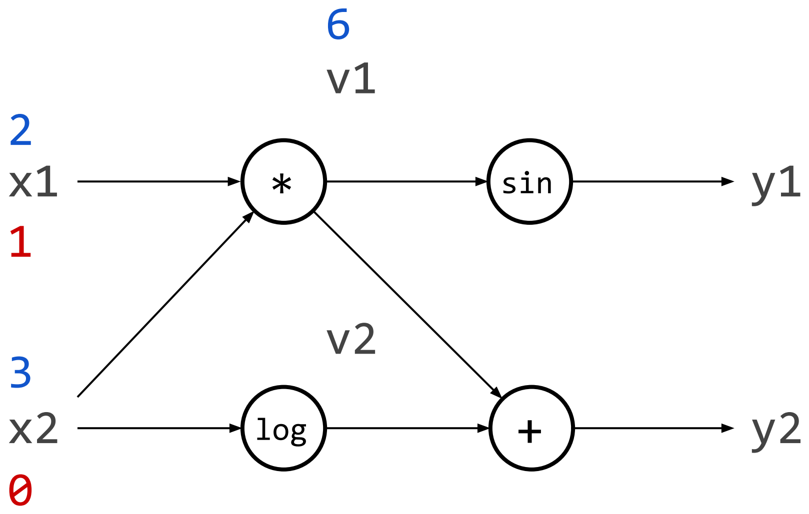
$$\frac{\partial x_2}{\partial x_1} = 0$$

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



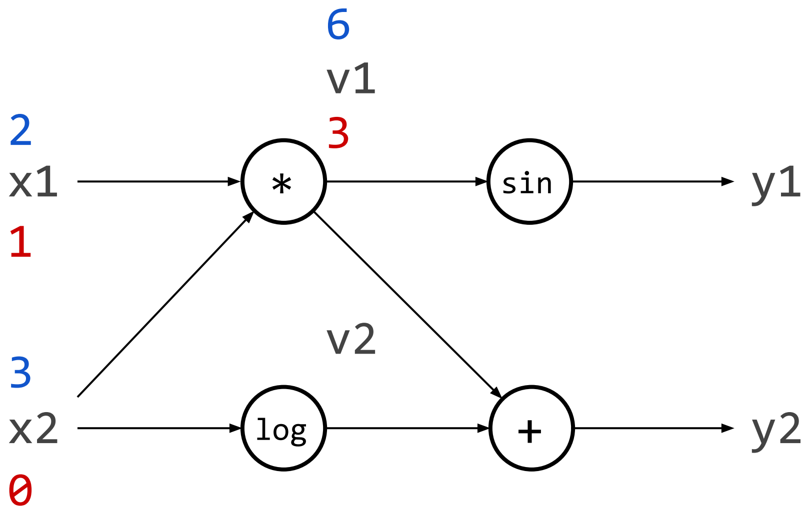
$$\frac{\partial v_1}{\partial x_1} =$$

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



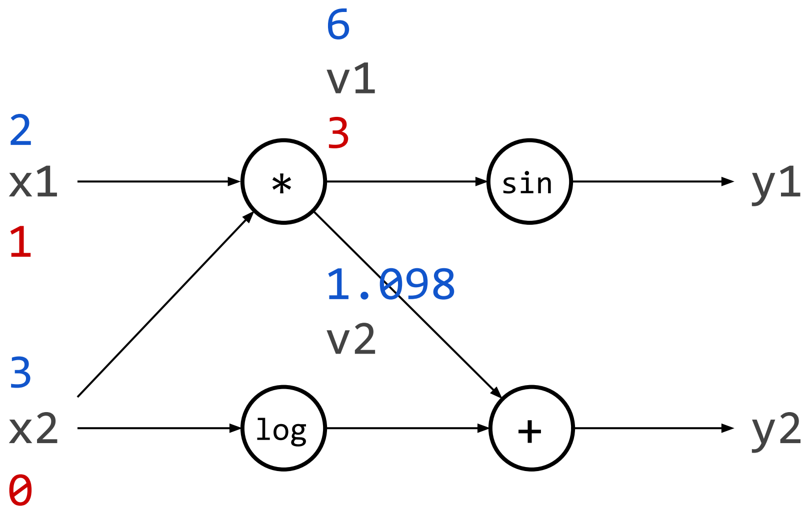
$$\frac{\partial v_1}{\partial x_1} = \frac{\partial x_1}{\partial x_1} x_2 + x_1 \frac{\partial x_2}{\partial x_1} = x_2$$

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



$$\frac{\partial v_2}{\partial x_1} =$$

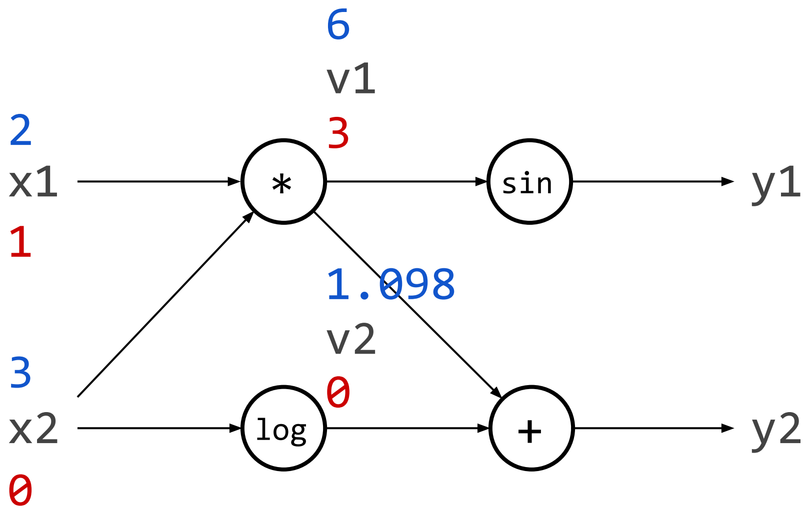
Forward mode

Primals: independent → dependent

Derivatives (tangents): independent → dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



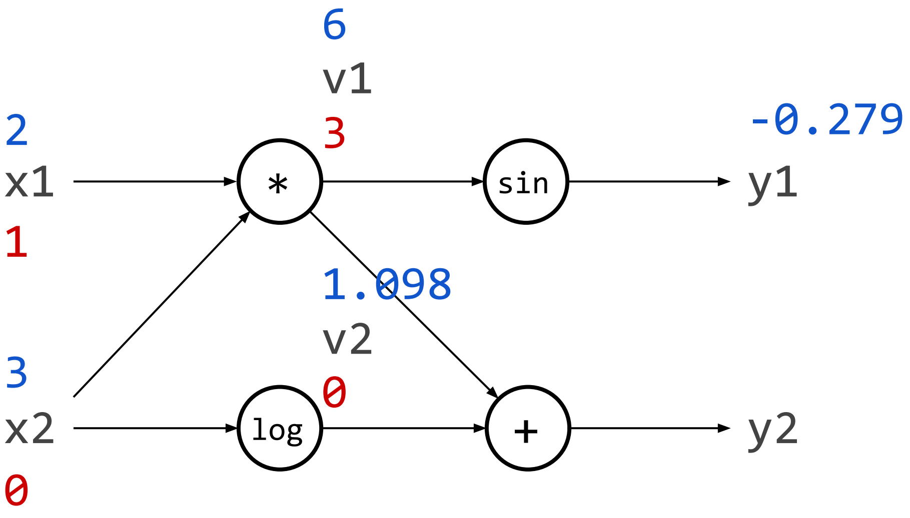
$$\frac{\partial v_2}{\partial x_1} = \frac{1}{x_2}\frac{\partial x_2}{\partial x_1} = 0$$

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



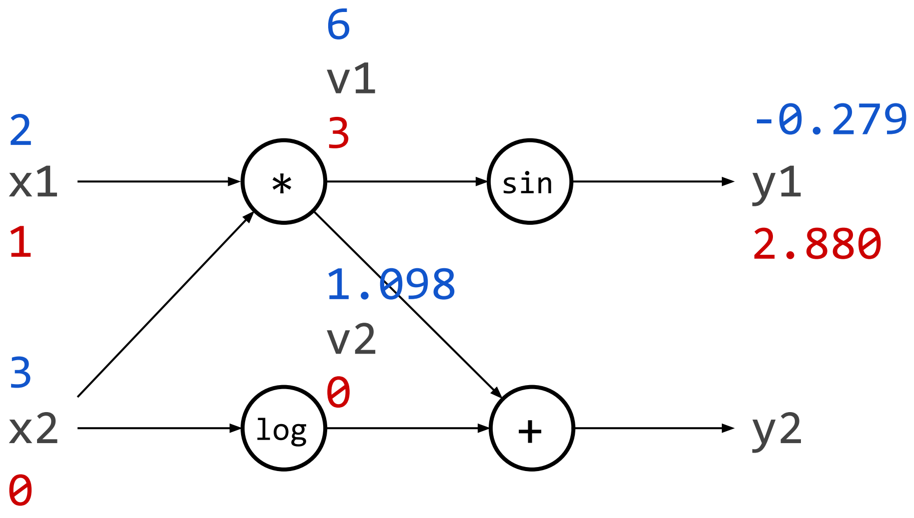
$$\frac{\partial y_1}{\partial x_1} =$$

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f: \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



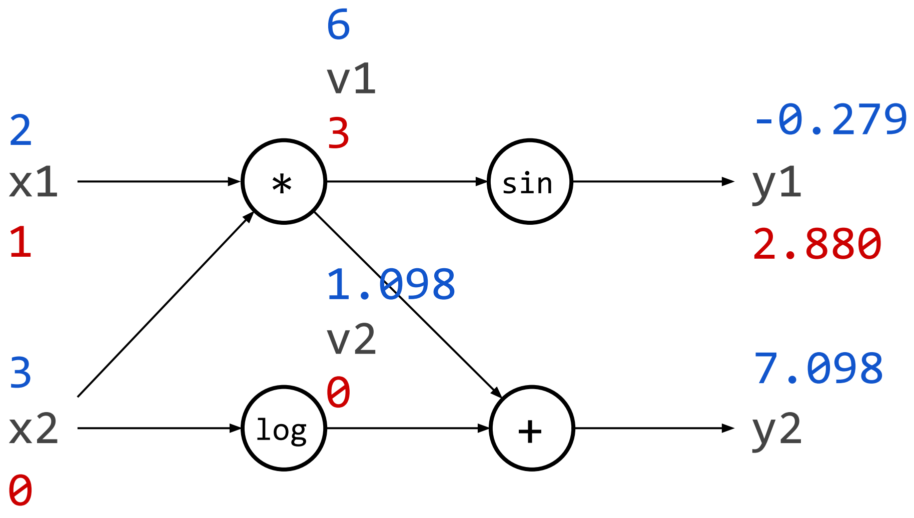
$$\frac{\partial y_1}{\partial x_1} = \cos(v_1)\frac{\partial v_1}{\partial x_1}$$

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



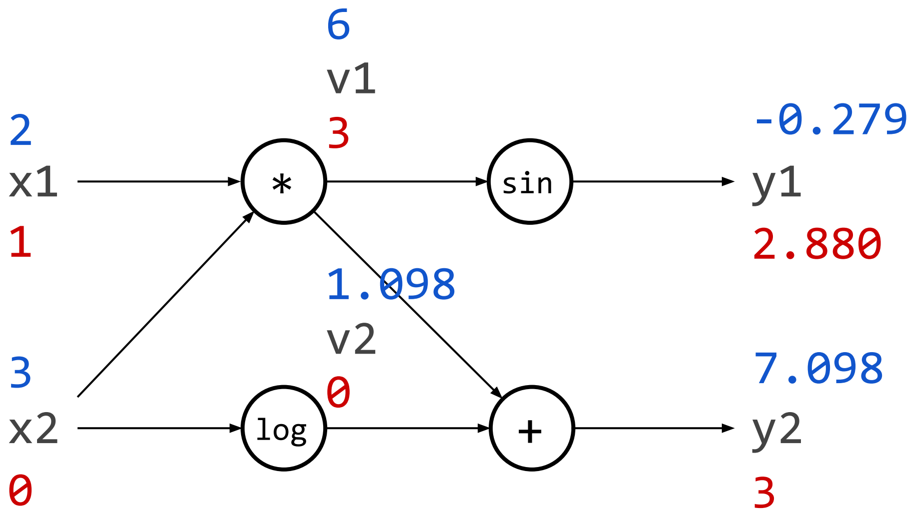
$$\frac{\partial y_2}{\partial x_1} =$$

Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



$$\frac{\partial y_2}{\partial x_1} = \frac{\partial v_1}{\partial x_1} + \frac{\partial v_2}{\partial x_1}$$

Forward mode

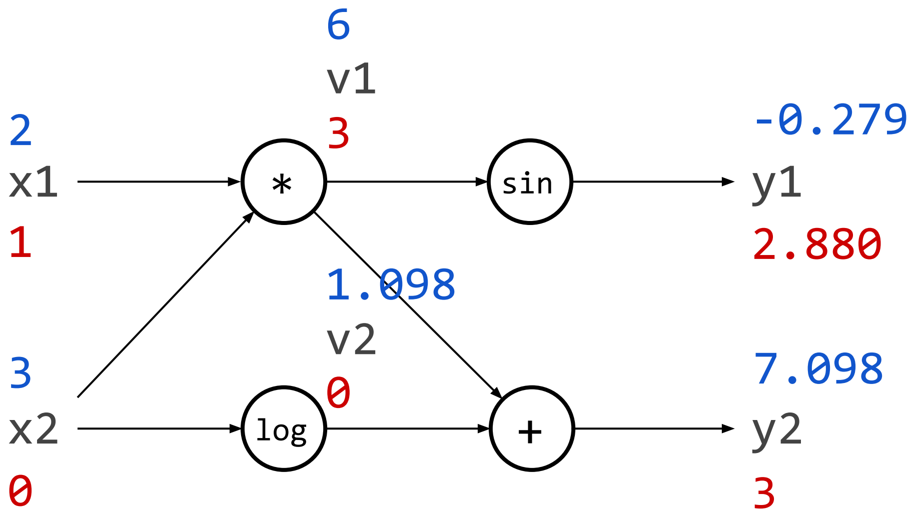
Primals: independent → dependent
Derivatives (tangents): independent → dependent

$f: \mathbb{R}^2 \to \mathbb{R}^2$

In general, forward mode evaluates a Jacobian–vector product $\mathbf{J}_f(\mathbf{x})\mathbf{v}$

So we evaluated:

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} \\ \frac{\partial y_2}{\partial x_1} \end{bmatrix}$$



Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

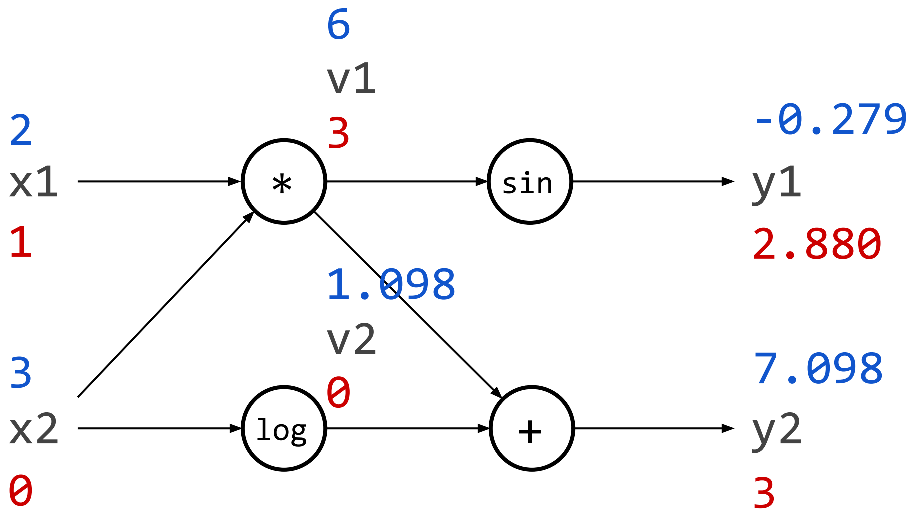
$f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

In general, forward mode evaluates a Jacobian–vector product $\mathbf{J}_f(\mathbf{x})\mathbf{v}$

So we evaluated:

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} \\ \frac{\partial y_2}{\partial x_1} \end{bmatrix}$$

Can be any $\mathbf{v} \in \mathbb{R}^2$
not only unit vectors



Forward mode

Primals: independent → dependent
Derivatives (tangents): independent → dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

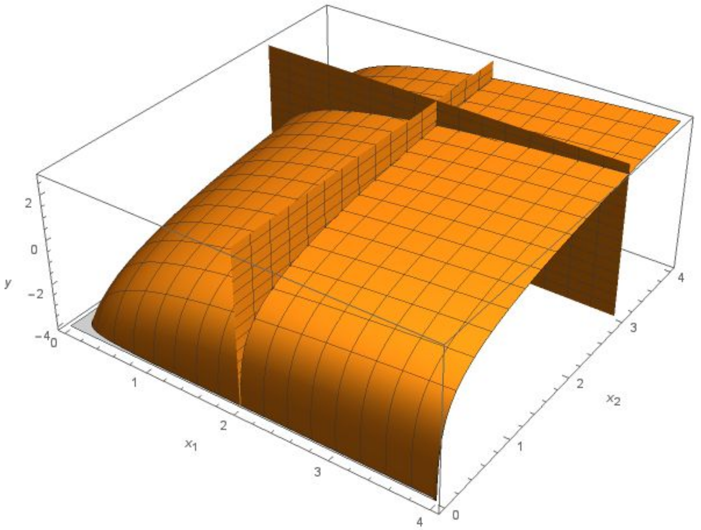
In general, forward mode evaluates a Jacobian–vector product $\mathbf{J}_f(\mathbf{x})\mathbf{v}$

For $f : \mathbb{R}^n \to \mathbb{R}$ this is a directional derivative $\nabla f(\mathbf{x}) \cdot \mathbf{v}$

So we evaluated:

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} \\ \frac{\partial y_2}{\partial x_1} \end{bmatrix}$$

Can be any $\mathbf{v} \in \mathbb{R}^2$ not only unit vectors

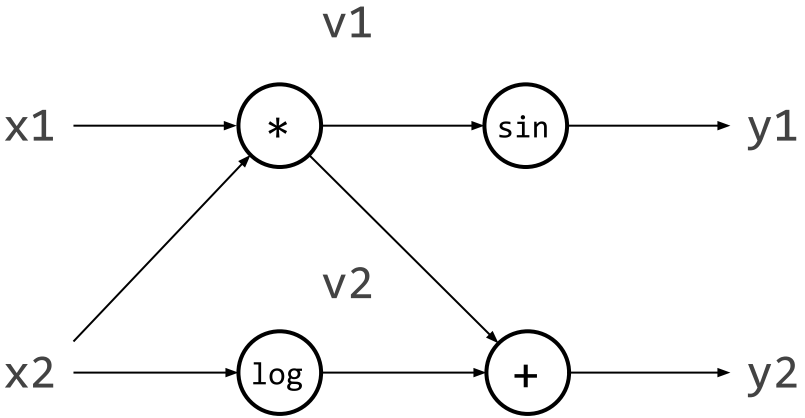


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)
```

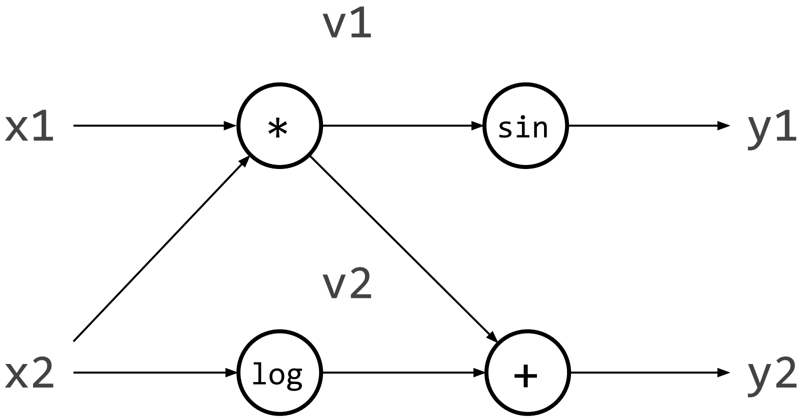


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

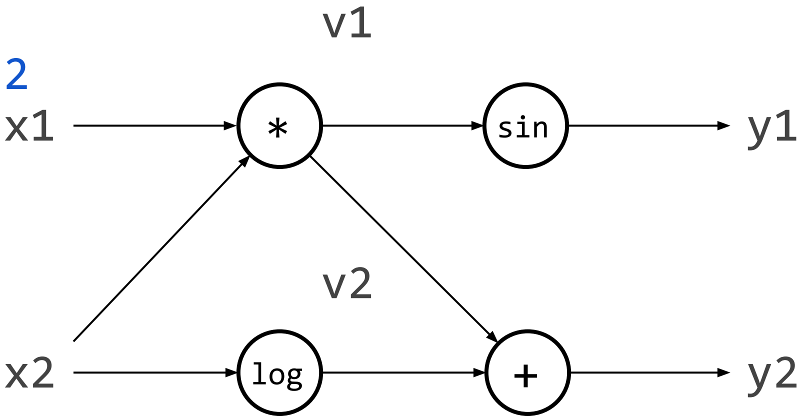


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

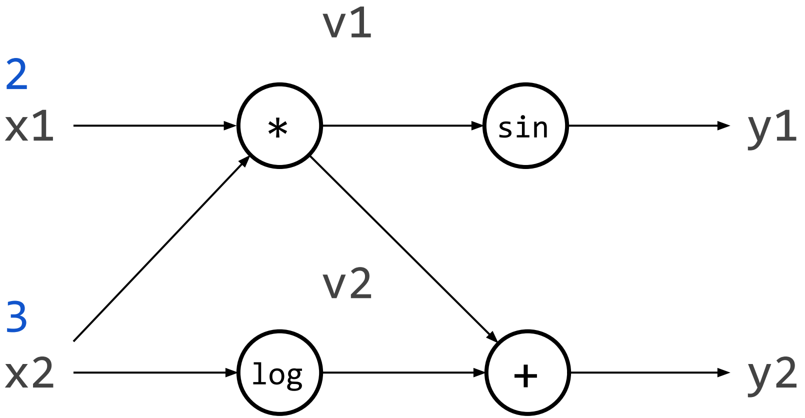


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

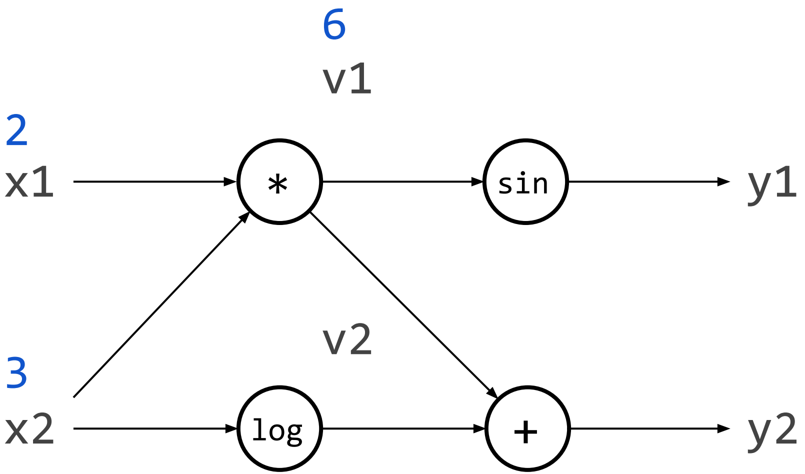


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

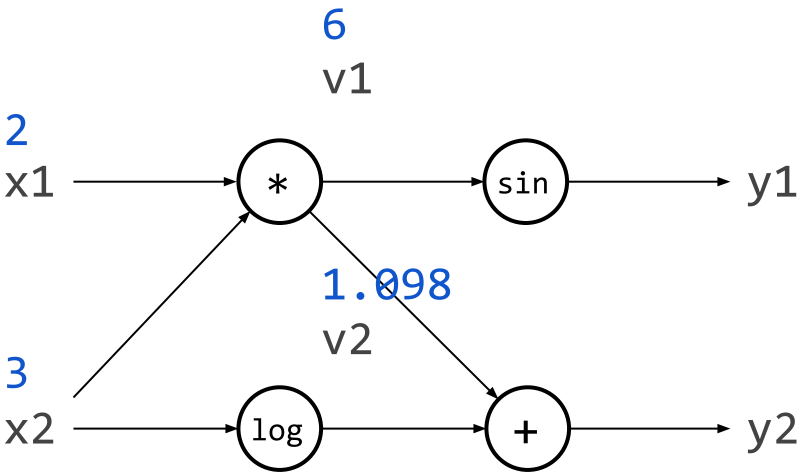


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

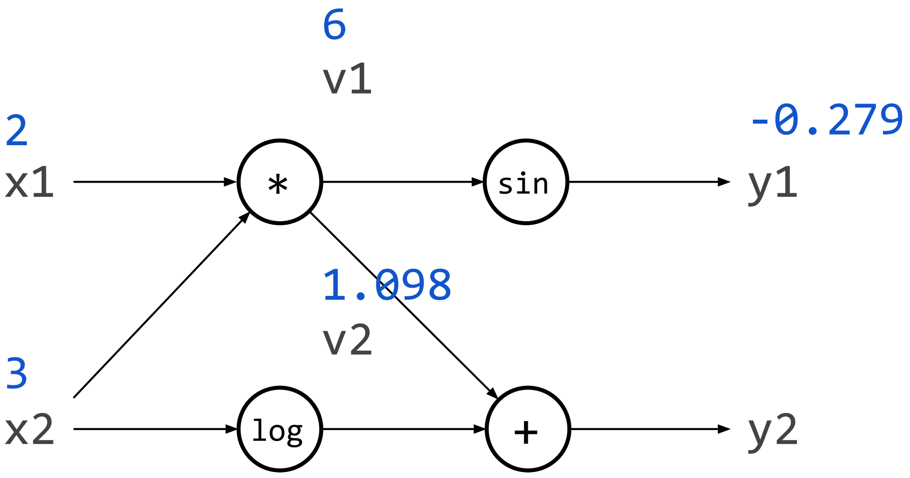


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

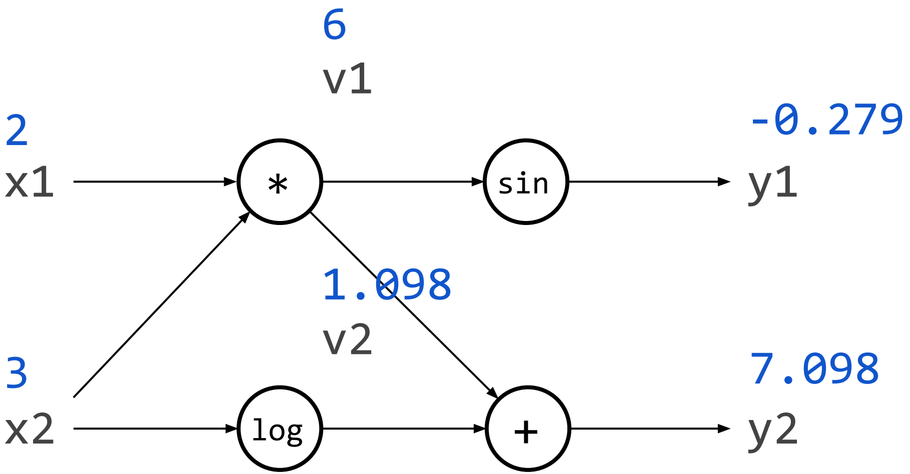


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```

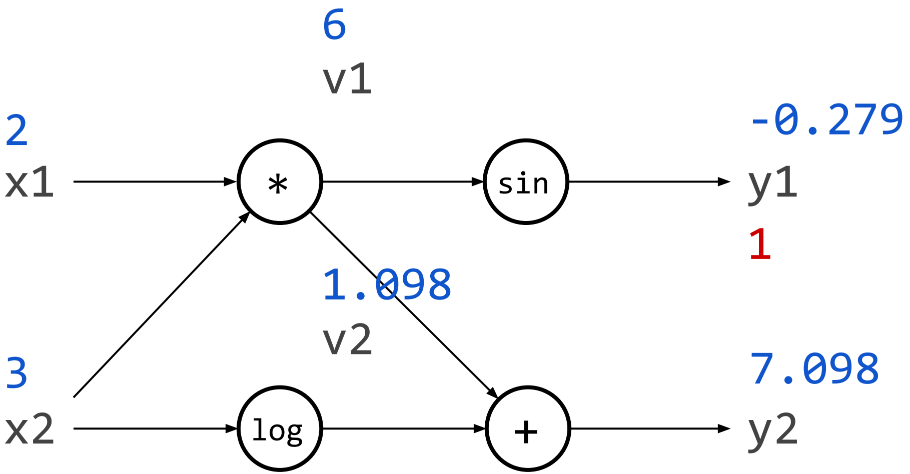


Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



$$\frac{\partial y_1}{\partial y_1} = 1$$

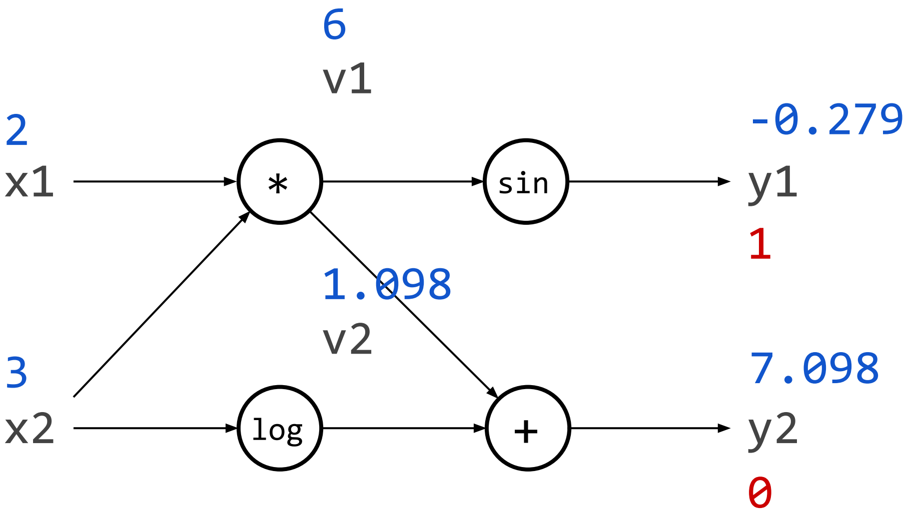
Reverse mode

Primals: independent → dependent

Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



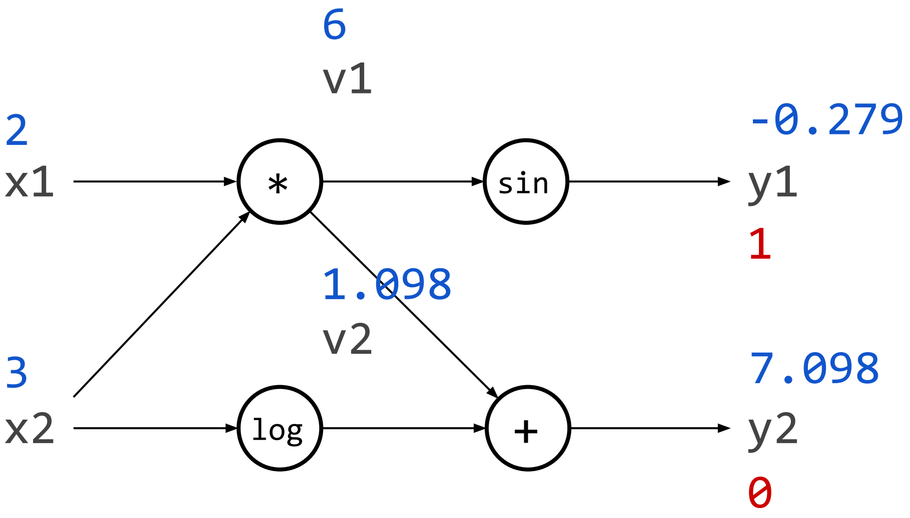
$$\frac{\partial y_1}{\partial y_2} = 0$$

Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



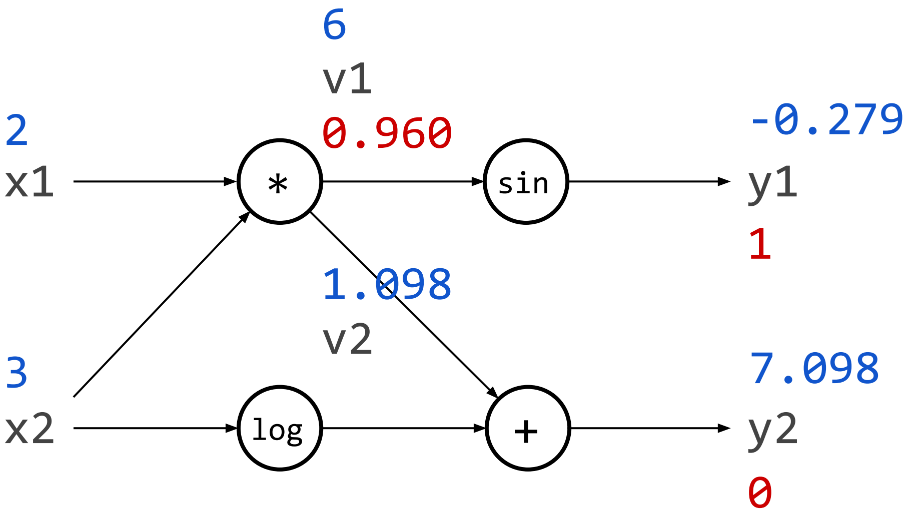
$$\frac{\partial y_1}{\partial v_1} =$$

Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



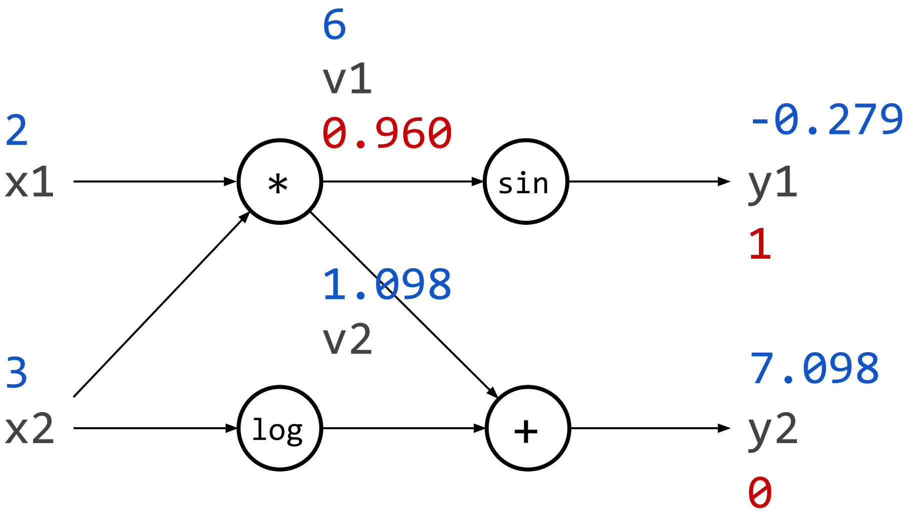
$$\frac{\partial y_1}{\partial v_1} = \cos(v1)\frac{\partial y_1}{\partial y_1}$$

Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



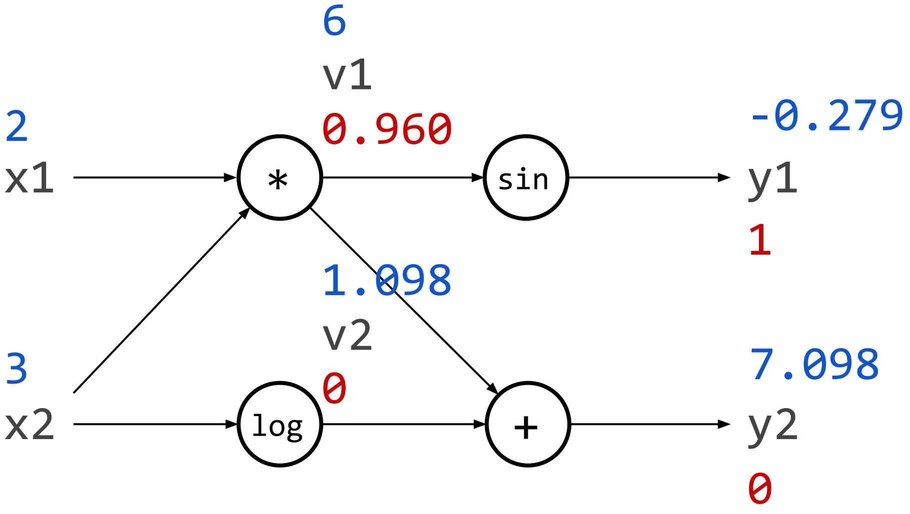
$$\frac{\partial y_1}{\partial v_2} =$$

Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



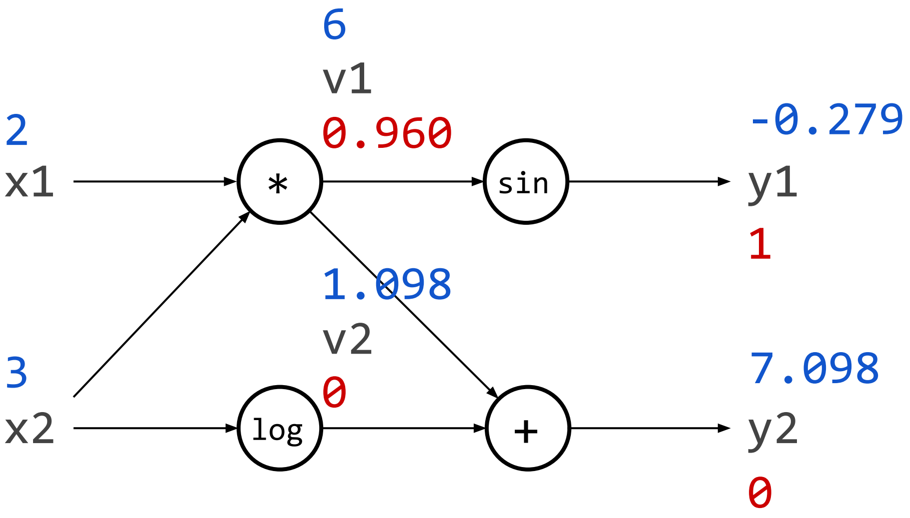
$$\frac{\partial y_1}{\partial v_2} = 0$$

Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



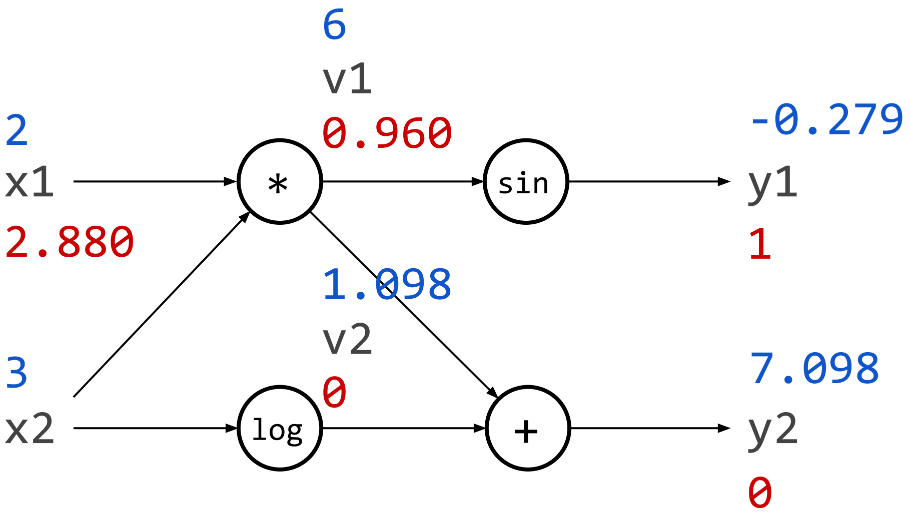
$$\frac{\partial y_1}{\partial x_1} =$$

Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



$$\frac{\partial y_1}{\partial x_1} = \frac{\partial v_1}{\partial x_1}\frac{\partial y_1}{\partial v_1} = x_2\frac{\partial y_1}{\partial v_1}$$

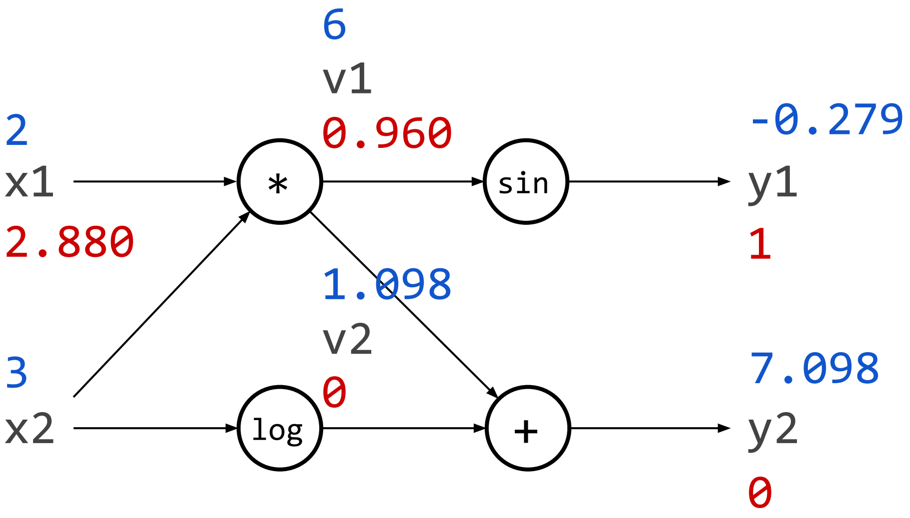
Reverse mode

Primals: independent → dependent

Derivatives (adjoints): independent ← dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



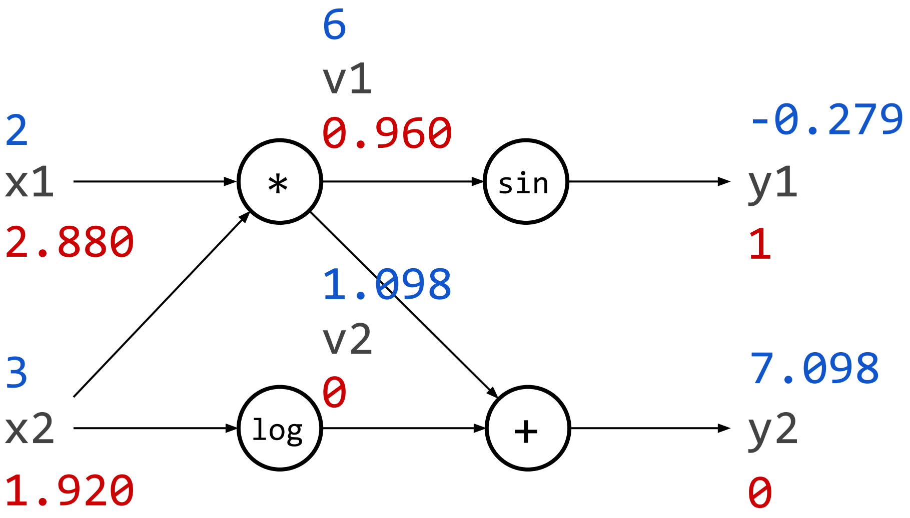
$$\frac{\partial y_1}{\partial x_2} =$$

Reverse mode

Primals: independent → dependent
Derivatives (adjoints): independent ← dependent

$f : \mathbb{R}^2 \to \mathbb{R}^2$

```
f(x1, x2):
  v1 = x1 * x2
  v2 = log(x2)
  y1 = sin(v1)
  y2 = v1 + v2
  return (y1, y2)

f(2, 3)
```



$$\frac{\partial y_1}{\partial x_2} = \frac{\partial v_1}{\partial x_2}\frac{\partial y_1}{\partial v_1} + \frac{\partial v_2}{\partial x_2}\frac{\partial y_1}{\partial v_2} = x_1 \frac{\partial y_1}{\partial v_1}$$

Reverse mode

Primals: independent $\rightarrow$ dependent
Derivatives (adjoints): independent $\leftarrow$ dependent

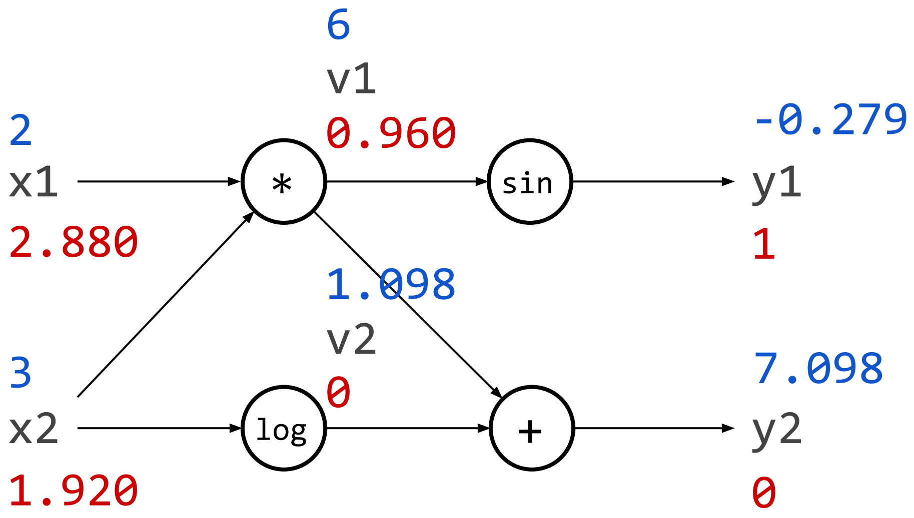
$$f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

In general, forward mode evaluates a transposed Jacobian–vector product

$$\mathbf{J}_f^{\mathsf{T}}(\mathbf{x})\mathbf{v}$$

So we evaluated:

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} \end{bmatrix}$$



Reverse mode

Primals: independent $\rightarrow$ dependent
Derivatives (adjoints): independent $\leftarrow$ dependent

$$f : \mathbb{R}^2 \to \mathbb{R}^2$$

In general, reverse mode evaluates a transposed Jacobian–vector product

$$\mathbf{J}_f^{\mathsf{T}}(\mathbf{x})\mathbf{v}$$

So we evaluated:

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} \end{bmatrix}$$

For $f : \mathbb{R}^n \to \mathbb{R}$ this is the gradient $\nabla f(\mathbf{x})$

Forward vs reverse summary

In the extreme $\mathbf{f} : \mathbb{R} \to \mathbb{R}^m$
use forward mode to evaluate

$$(\frac{\partial f_1}{\partial x}, \cdots, \frac{\partial f_m}{\partial x})$$

In the extreme $f : \mathbb{R}^n \to \mathbb{R}$
use reverse mode to evaluate

$$\nabla f(\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n})$$

Forward vs reverse summary

In the extreme $\mathbf{f} : \mathbb{R} \to \mathbb{R}^m$
use forward mode to evaluate

$$(\frac{\partial f_1}{\partial x}, \cdots, \frac{\partial f_m}{\partial x})$$

In the extreme $f : \mathbb{R}^n \to \mathbb{R}$
use reverse mode to evaluate

$$\nabla f(\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n})$$

In general $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ the Jacobian $\mathbf{J}_f(\mathbf{x}) \in \mathbb{R}^{m \times n}$ can be evaluated in

- $O(n \,\mathrm{time}(\mathbf{f}))$ with forward mode
- $O(m \,\mathrm{time}(\mathbf{f}))$ with reverse mode

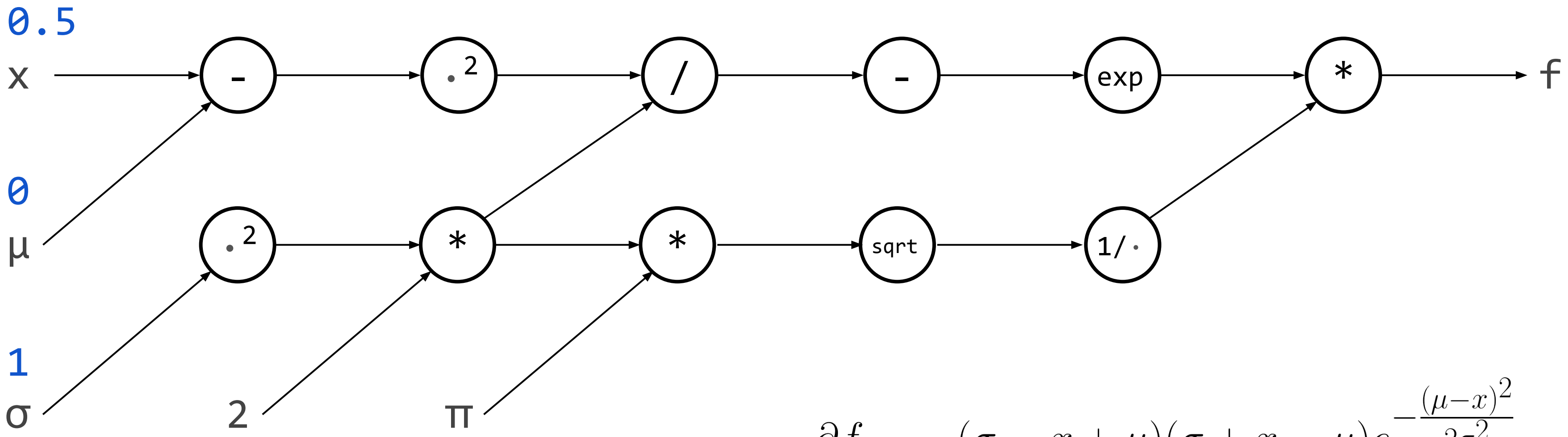
Reverse performs better when $n \gg m$

Backprop through normal PDF

Backprop through normal PDF

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\frac{\partial f}{\partial x} = \frac{(\mu - x)e^{-\frac{(\mu-x)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma^3} \qquad \frac{\partial f}{\partial \mu} = \frac{(x - \mu)e^{-\frac{(\mu-x)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma^3}$$



$$\frac{\partial f}{\partial \sigma} = -\frac{(\sigma - x + \mu)(\sigma + x - \mu)e^{-\frac{(\mu-x)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma^4}$$

Summary

Summary

This lecture:

- Derivatives in machine learning
- Review of essential concepts (what is a derivative, etc.)
- How do we compute derivatives
- Automatic differentiation

Next lecture:

- Current landscape of tools
- Implementation techniques
- Advanced concepts (higher-order API, checkpointing, etc.)

References

Baydin, A.G., Pearlmutter, B.A., Radul, A.A. and Siskind, J.M., 2017. Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research (JMLR), 18(153), pp.1-153.

Baydin, Atılım Güneş, Barak A. Pearlmutter, and Jeffrey Mark Siskind. 2016. "Tricks from Deep Learning." In 7th International Conference on Algorithmic Differentiation, Christ Church Oxford, UK, September 12–15, 2016.

Baydin, Atılım Güneş, Barak A. Pearlmutter, and Jeffrey Mark Siskind. 2016. "DiffSharp: An AD Library for .NET Languages." In 7th International Conference on Algorithmic Differentiation, Christ Church Oxford, UK, September 12–15, 2016.

Baydin, Atılım Güneş, Robert Cornish, David Martínez Rubio, Mark Schmidt, and Frank Wood. 2018. "Online Learning Rate Adaptation with Hypergradient Descent." In Sixth International Conference on Learning Representations (ICLR), Vancouver, Canada, April 30 – May 3, 2018.

Griewank, A. and Walther, A., 2008. Evaluating derivatives: principles and techniques of algorithmic differentiation (Vol. 105). SIAM.

Nocedal, J. and Wright, S.J., 1999. Numerical Optimization. Springer.

Extra slides

Forward mode

Primals: independent dependent

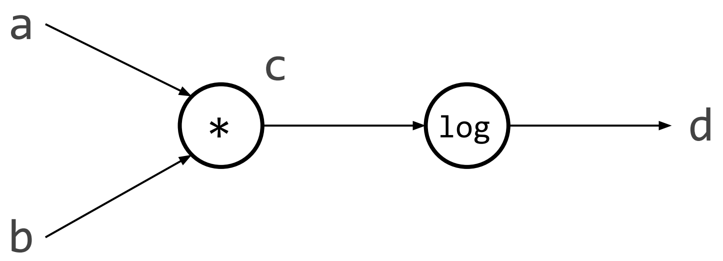
Derivatives (tangents): independent dependent

Forward mode

Primals: independent dependent

Derivatives (tangents): independent dependent

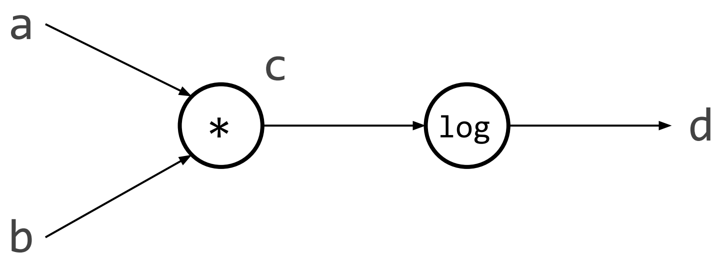
```
f(a, b):
  c = a * b
  d = log(c)
  return d
```



Forward mode

Primals: independent dependent
Derivatives (tangents): independent dependent

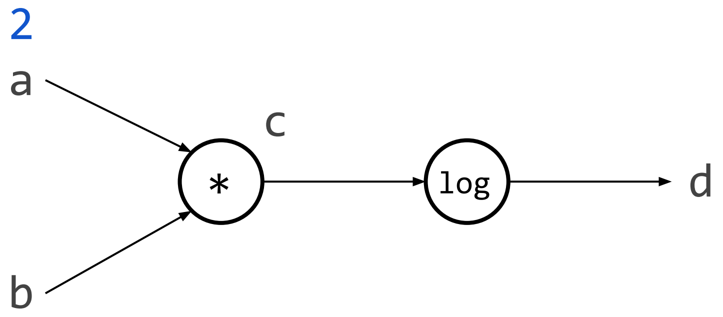
```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```



Forward mode

Primals: independent dependent
Derivatives (tangents): independent dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

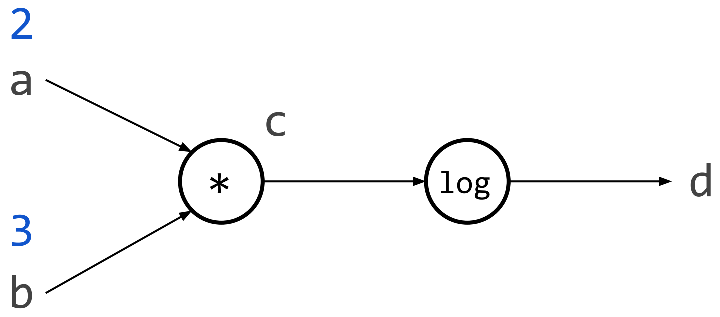


Forward mode

Primals: independent dependent

Derivatives (tangents): independent dependent

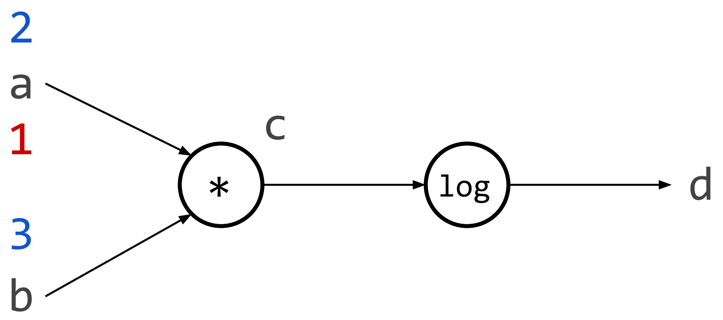
```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```



Forward mode

Primals: independent dependent
Derivatives (tangents): independent dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

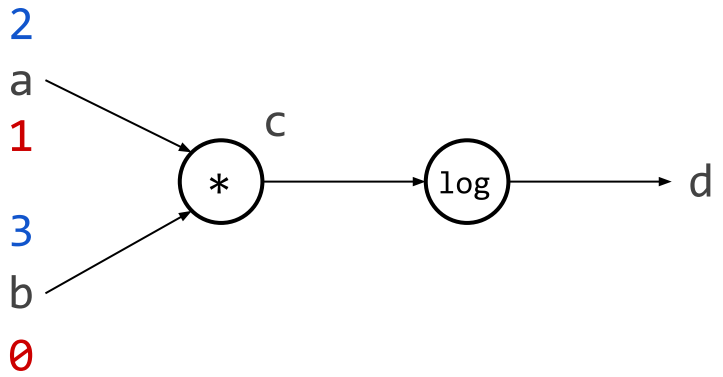


$$\frac{\partial a}{\partial a} = 1$$

Forward mode

Primals: independent □ dependent
Derivatives (tangents): independent □ dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

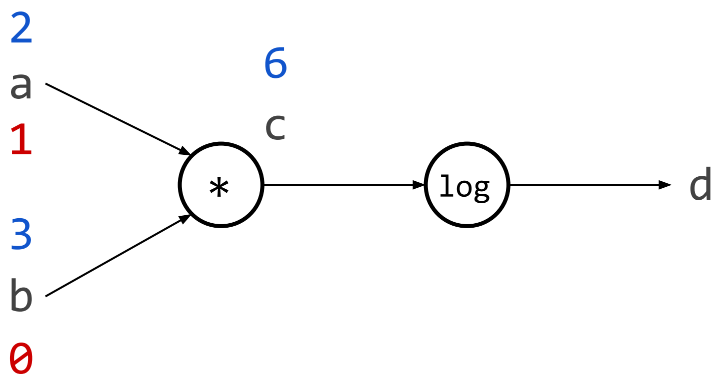


$$\frac{\partial b}{\partial a} = 0$$

Forward mode

Primals: independent dependent
Derivatives (tangents): independent dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

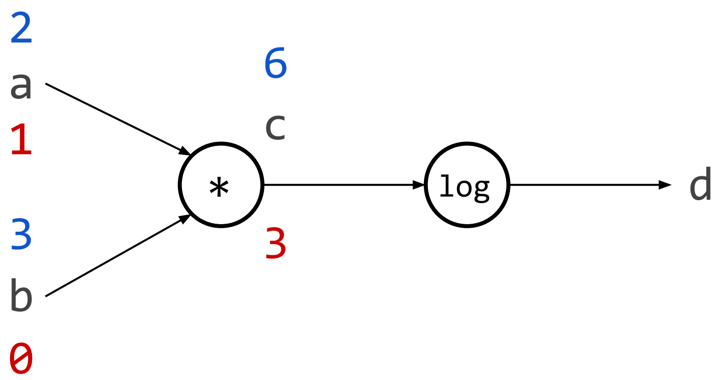


$$\frac{\partial c}{\partial a} =$$

Forward mode

Primals: independent dependent
Derivatives (tangents): independent dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

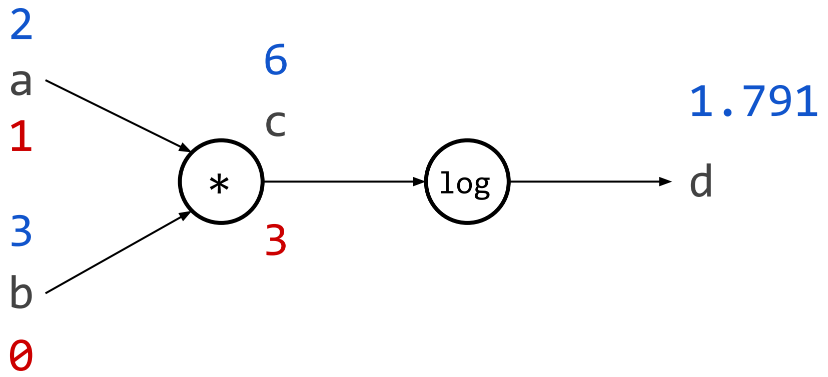


$$\frac{\partial c}{\partial a} = \frac{\partial a}{\partial a} b + a \frac{\partial b}{\partial a} = b$$

Forward mode

Primals: independent □ dependent
Derivatives (tangents): independent □ dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```



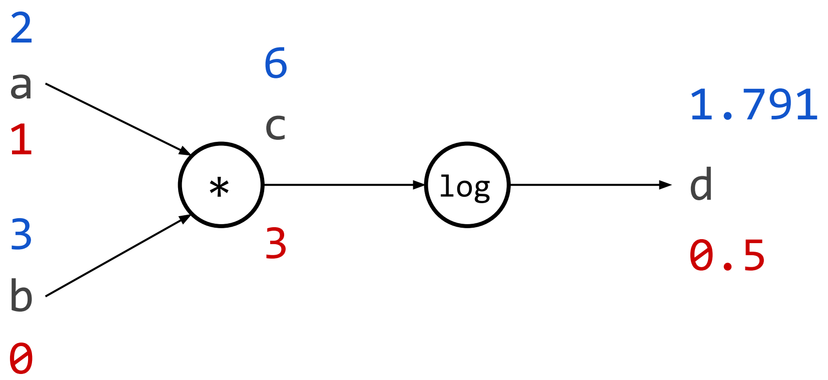
$$\frac{\partial d}{\partial a} =$$

Forward mode

Primals: independent □ dependent

Derivatives (tangents): independent □ dependent

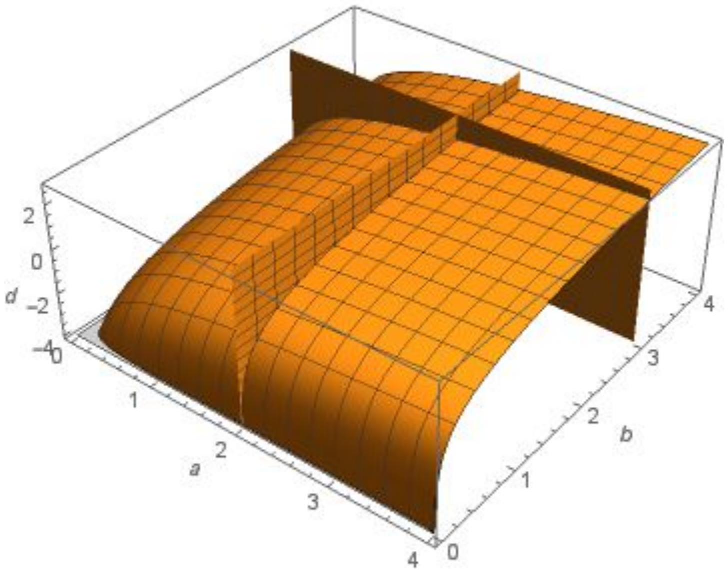
```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

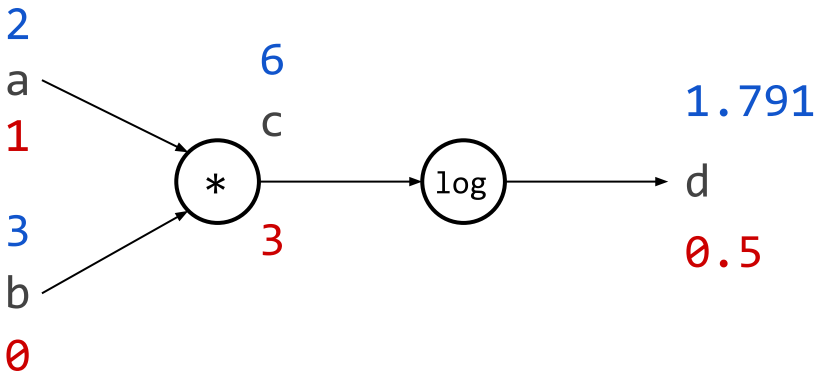


$$\frac{\partial d}{\partial a} = \frac{1}{c}\frac{\partial c}{\partial a}$$

Forward mode

Primals: independent □ dependent
Derivatives (tangents): independent □ dependent





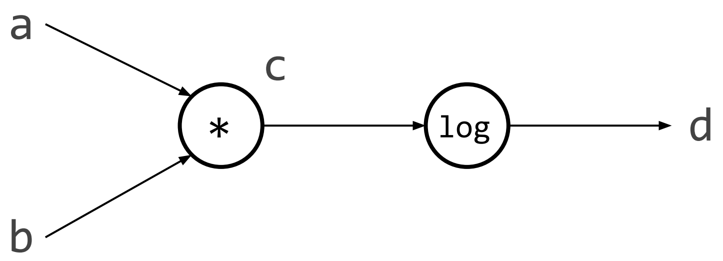
In general, forward mode evaluates a Jacobian–vector product $\mathbf{J}_f(\mathbf{x})\mathbf{v}$

We evaluated the partial derivative $\dfrac{\partial d}{\partial a}$ with $\mathbf{x} = (a, b), \mathbf{v} = (1, 0)$

Reverse mode

Primals: independent dependent
Derivatives (adjoints): independent dependent

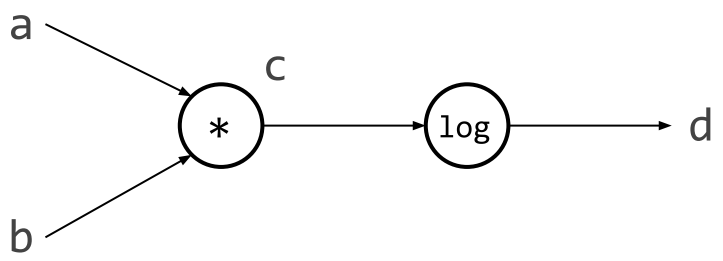
```
f(a, b):
  c = a * b
  d = log(c)
  return d
```



Reverse mode

Primals: independent dependent
Derivatives (adjoints): independent dependent

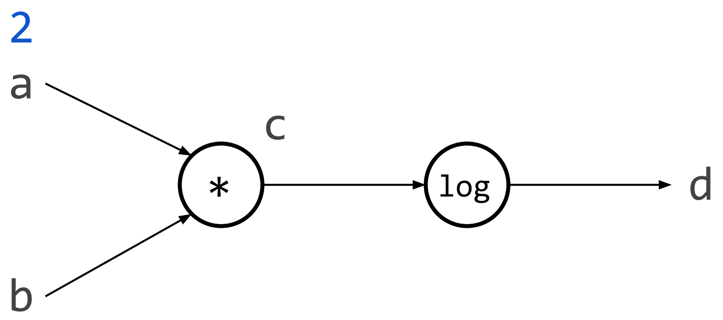
```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```



Reverse mode

Primals: independent dependent
Derivatives (adjoints): independent dependent

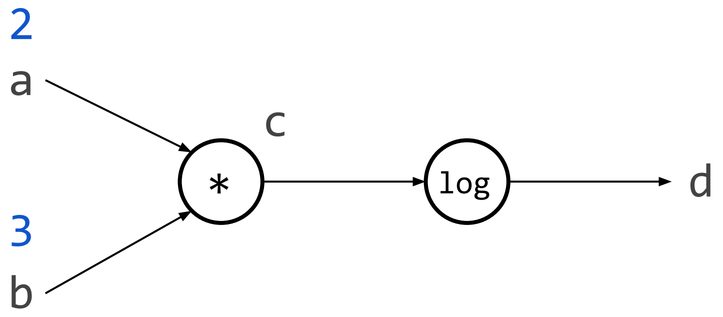
```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```



Reverse mode

Primals: independent dependent
Derivatives (adjoints): independent dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

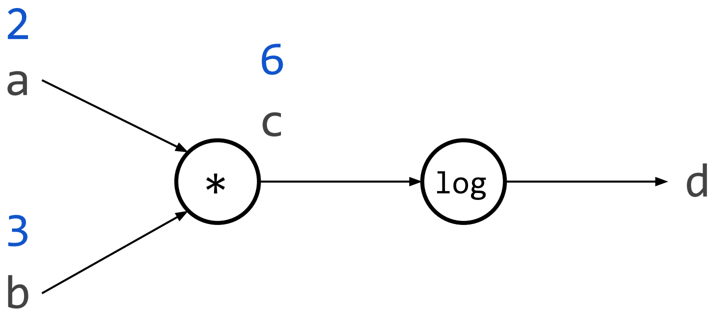


Reverse mode

Primals: independent □ dependent

Derivatives (adjoints): independent □ dependent

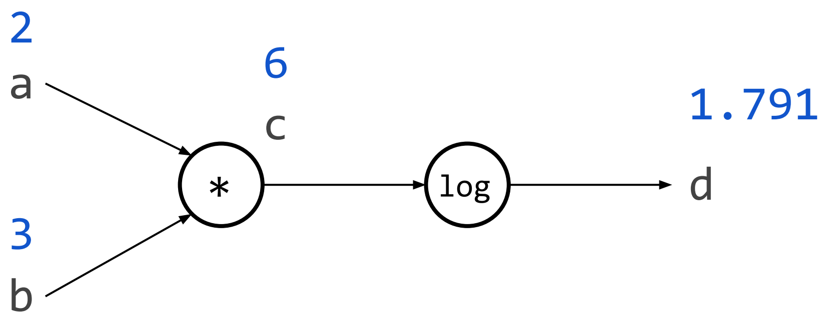
```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```



Reverse mode

Primals: independent □ dependent
Derivatives (adjoints): independent □ dependent

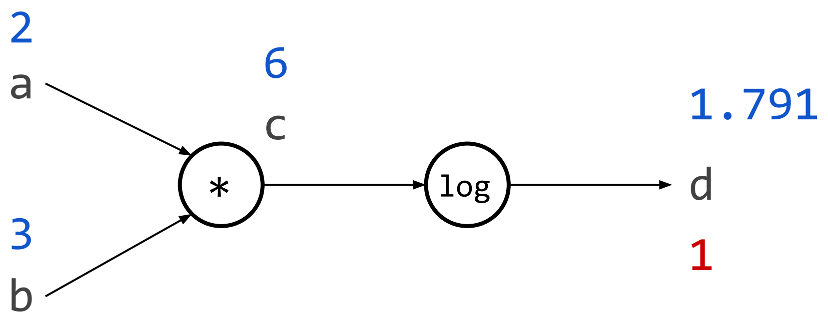
```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```



Reverse mode

Primals: independent dependent
Derivatives (adjoints): independent dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

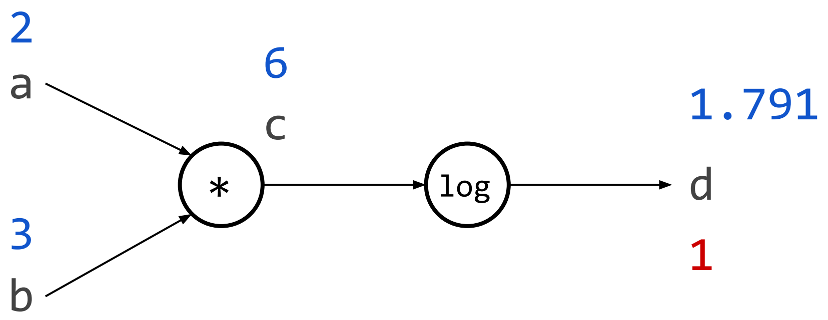


$$\frac{\partial d}{\partial d} = 1$$

Reverse mode

Primals: independent □ dependent
Derivatives (adjoints): independent □ dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

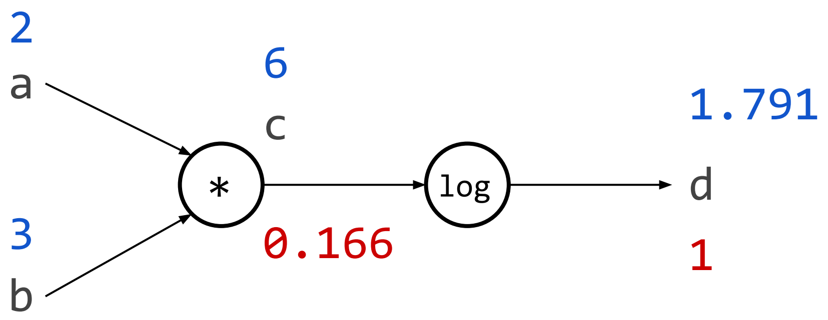


$$\frac{\partial d}{\partial c} =$$

Reverse mode

Primals: independent dependent
Derivatives (adjoints): independent dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

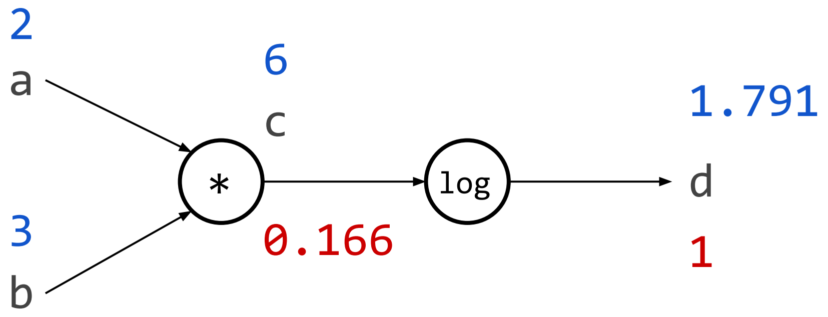


$$\frac{\partial d}{\partial c} = \frac{1}{c}\frac{\partial d}{\partial d}$$

Reverse mode

Primals: independent □ dependent
Derivatives (adjoints): independent □ dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```



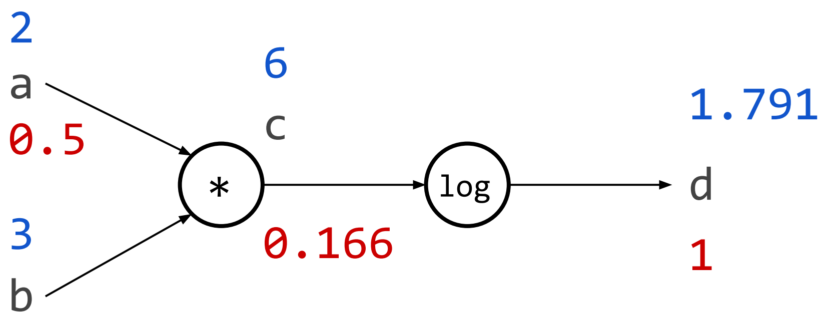
$$\frac{\partial d}{\partial a} =$$

Reverse mode

Primals: independent dependent

Derivatives (adjoints): independent dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

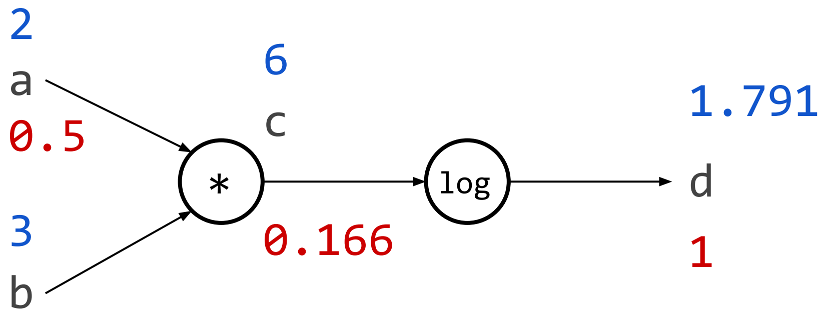


$$\frac{\partial d}{\partial a} = \frac{\partial c}{\partial a}\frac{\partial d}{\partial c} = b\,\frac{\partial d}{\partial c}$$

Reverse mode

Primals: independent □ dependent
Derivatives (adjoints): independent □ dependent

```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

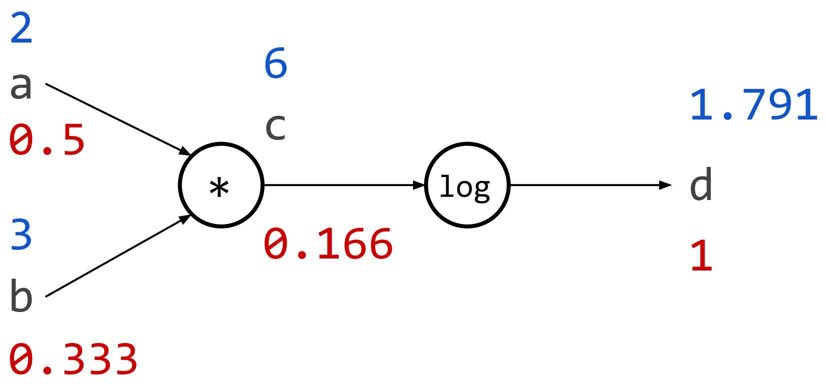


$$\frac{\partial d}{\partial b} =$$

Reverse mode

Primals: independent □ dependent
Derivatives (adjoints): independent □ dependent

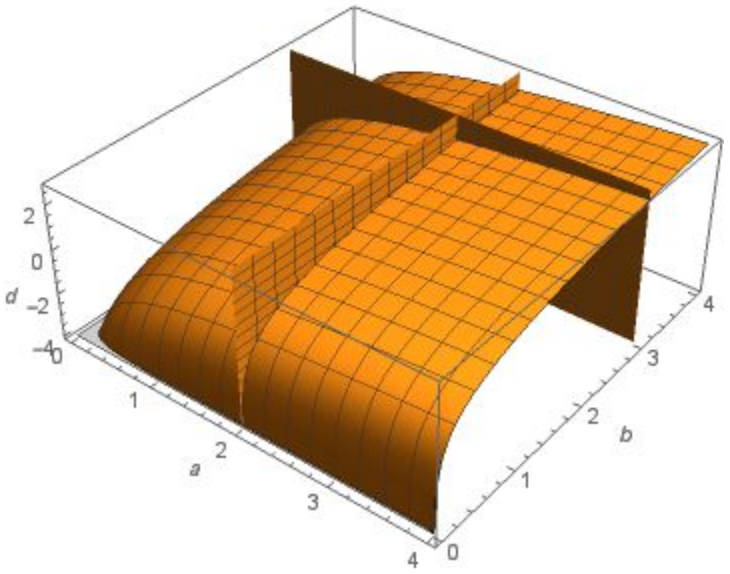
```
f(a, b):
  c = a * b
  d = log(c)
  return d

f(2, 3)
```

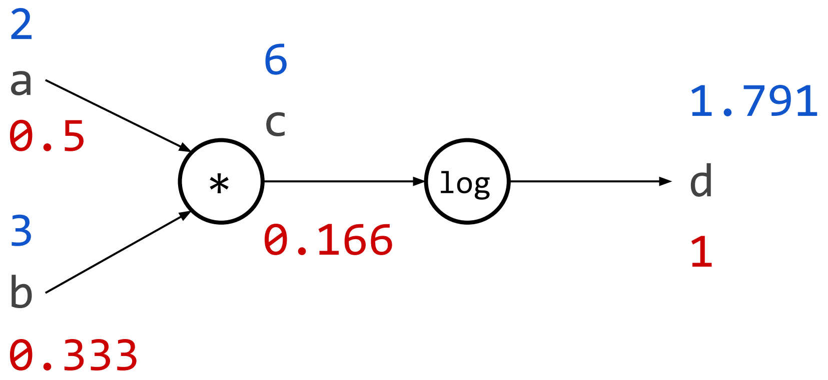


$$\frac{\partial d}{\partial b} = \frac{\partial c}{\partial b}\frac{\partial d}{\partial c} = a\frac{\partial d}{\partial c}$$

Reverse mode

Primals: independent □ dependent
Derivatives (adjoints): independent □ dependent





In general, reverse mode evaluates a transposed Jacobian–vector product $\mathbf{J}_f^{\mathsf{T}}(\mathbf{x})\mathbf{v}$

We evaluated the gradient $\nabla f(a,b) = (\frac{\partial d}{\partial a}, \frac{\partial d}{\partial b})$ with $\mathbf{x} = (a,b), \mathbf{v} = (1)$

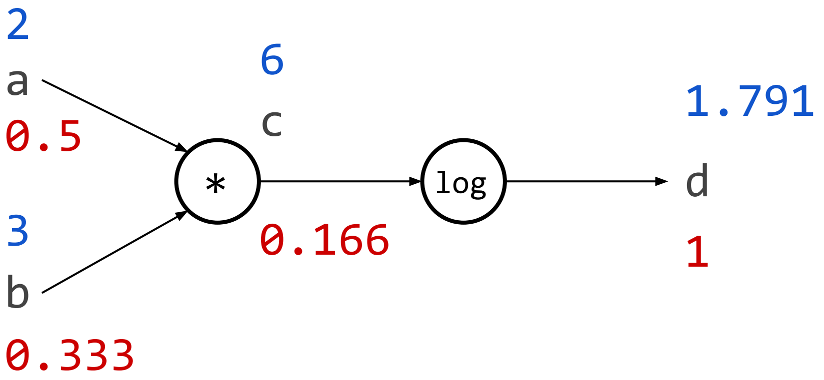
Reverse mode

Primals: independent □ dependent
Derivatives (adjoints): independent □ dependent

```
import torch
```

```
def f(x):
    c = x[0] * x[1]
    if c > 0:
        d = torch.log(c)
    else:
        d = torch.sin(c)
    return d

x = torch.tensor([2., 3.], requires_grad=True)
y = f(x)
y.backward()
print(y)
print(x.grad)
```

```
tensor(1.7918, grad_fn=<LogBackward>)
tensor([0.5000, 0.3333])
```



In general, reverse mode evaluates a transposed Jacobian–vector product $\mathbf{J}_f^\mathsf{T}(\mathbf{x})\mathbf{v}$

We evaluated the gradient $\nabla f(a,b) = (\frac{\partial d}{\partial a}, \frac{\partial d}{\partial b})$ with $\mathbf{x} = (a,b), \mathbf{v} = (1)$