VII.3) Backpropagation and the Chain Rule

When training a neural network, most of the time is being spent on computing the gradient of the loss function with respect to the many parameters of the neural net.

Finding an explicit formula for the gradient and writing the code to evaluate the gradient is not reasonable for such a large optimization problem.

There are two alternatives to finding the gradient explicitly.

One alternative is to use finite differences to approximate all the partial derivatives in the gradient:

$$\nabla f(x)_i = \frac{\partial f}{\partial x_i} \approx \frac{f(x + he_i) - f(x)}{h},$$

where e_i is the i^{th} column of the identity matrix. However, this would require us to evaluate the function $f : \mathbb{R}^n \to \mathbb{R}$ many times, and we would only get an approximation.

A better approach is to use AD (automatic differentiation) which produces a routine that will exactly evaluate the derivatives.

Two ways AD is implemented are:

(1) source code transformation (SCT),

(2) operator overloading (OO).

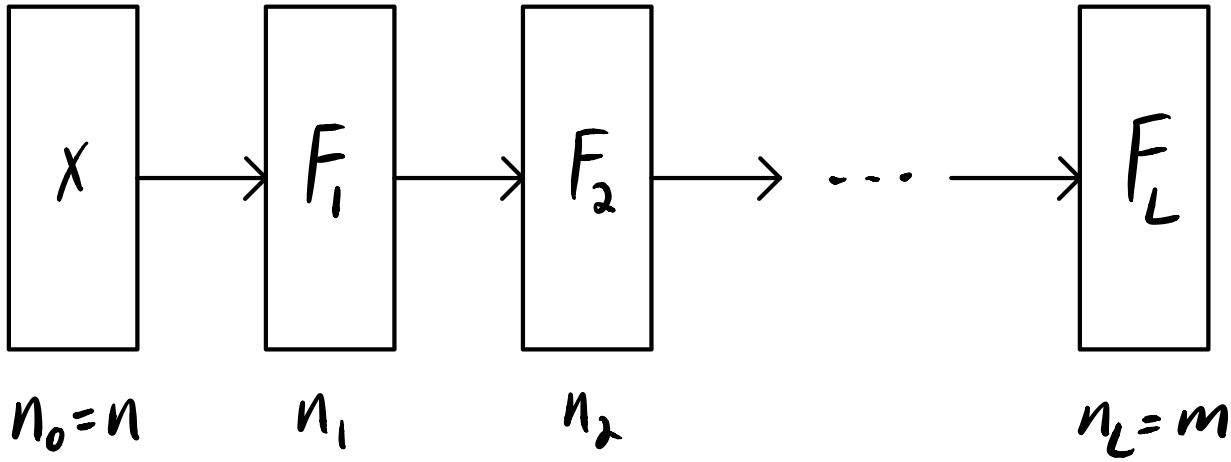
The Julia package Zygote.jl is an SCT implementation while the ForwardDiff.jl package uses OO.

Forward-mode AD

Suppose $F: \mathbb{R}^n \to \mathbb{R}^m$ is given by a composition of intermediate calculations:

$$F(x) = (F_L \circ \cdots \circ F_2 \circ F_1)(x)$$

$$= F_L(\cdots F_2(F_1(x))).$$



Here $F_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$, for $i = 1, \ldots, L$.

We use $F'(x)$ to represent the $m \times n$ Jacobian matrix of $F : \mathbb{R}^n \to \mathbb{R}^m$:

$$F(x+d) \approx F(x) + F'(x)d, \quad \forall d \in \mathbb{R}^n \text{ small}.$$

Example: $F : \mathbb{R}^2 \to \mathbb{R}^3$ $\quad F'(x)$ is 3×2

$$F(x) = \begin{bmatrix} x_1 + x_2 \\ x_1 x_2 \\ x_1^2 - x_2 \end{bmatrix} \qquad F'(x) = \begin{bmatrix} 1 & 1 \\ x_2 & x_1 \\ 2x_1 & -1 \end{bmatrix}$$

//

We can use the <u>chain rule</u> to take the derivative of a composition:

$$\boxed{F(x) = F_2(F_1(x)) \quad \Rightarrow \quad F'(x) = F_2'(F_1(x)) \cdot F_1'(x).}$$

With $F_1 : \mathbb{R}^n \to \mathbb{R}^{n_1}$, $F_2 : \mathbb{R}^{n_1} \to \mathbb{R}^m$, and $F : \mathbb{R}^n \to \mathbb{R}^m$, we have:

$$F'(x) = F_2'(F_1(x)) \cdot F_1'(x).$$

$m \times n \qquad m \times n_1 \qquad n_1 \times n$

<u>Example</u>:

$$F_1(x) = \begin{bmatrix} x_1 - x_2 \\ x_1 x_2 \end{bmatrix}, \quad F_2(x) = \begin{bmatrix} x_1^2 \\ x_1 - x_2 \end{bmatrix}$$

$$\Rightarrow F(x) = F_2(F_1(x)) = \begin{bmatrix} (x_1 - x_2)^2 \\ x_1 - x_2 - x_1 x_2 \end{bmatrix}.$$

$$F_1'(x) = \begin{bmatrix} 1 & -1 \\ x_2 & x_1 \end{bmatrix}, \quad F_2'(x) = \begin{bmatrix} 2x_1 & 0 \\ 1 & -1 \end{bmatrix}.$$

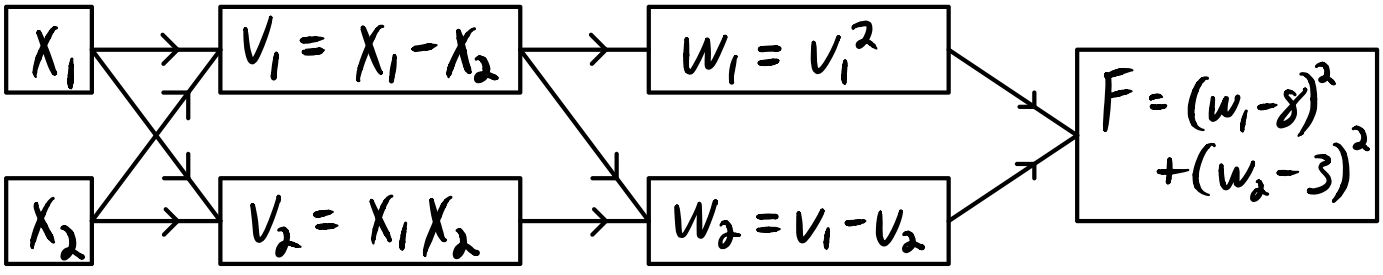
$$F'(x) = \begin{bmatrix} 2(x_1 - x_2) & -2(x_1 - x_2) \\ 1 - x_2 & -1 - x_1 \end{bmatrix}$$

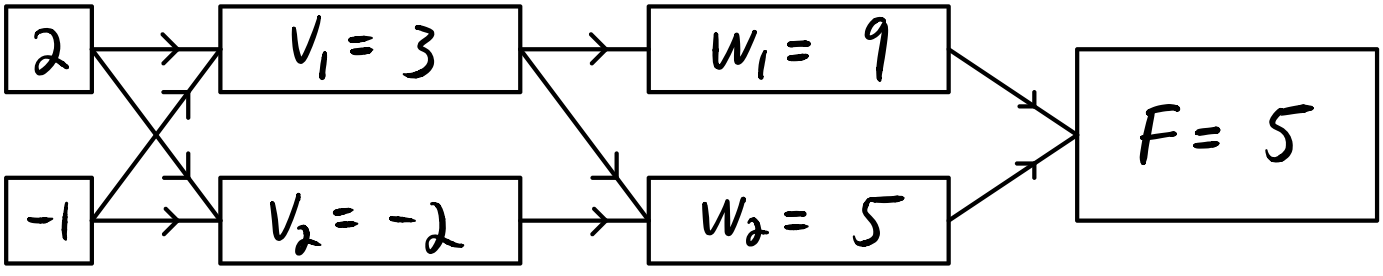
$$F_2'(F_1(x)) \cdot F_1'(x) = \begin{bmatrix} 2(x_1 - x_2) & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ x_2 & x_1 \end{bmatrix}$$

$$= \begin{bmatrix} 2(x_1 - x_2) & -2(x_1 - x_2) \\ 1 - x_2 & -1 - x_1 \end{bmatrix}.$$

Suppose $F_3(x) = (x_1 - 8)^2 + (x_2 - 3)^2$ and

$$F(x) = F_3(F_2(F_1(x))).$$

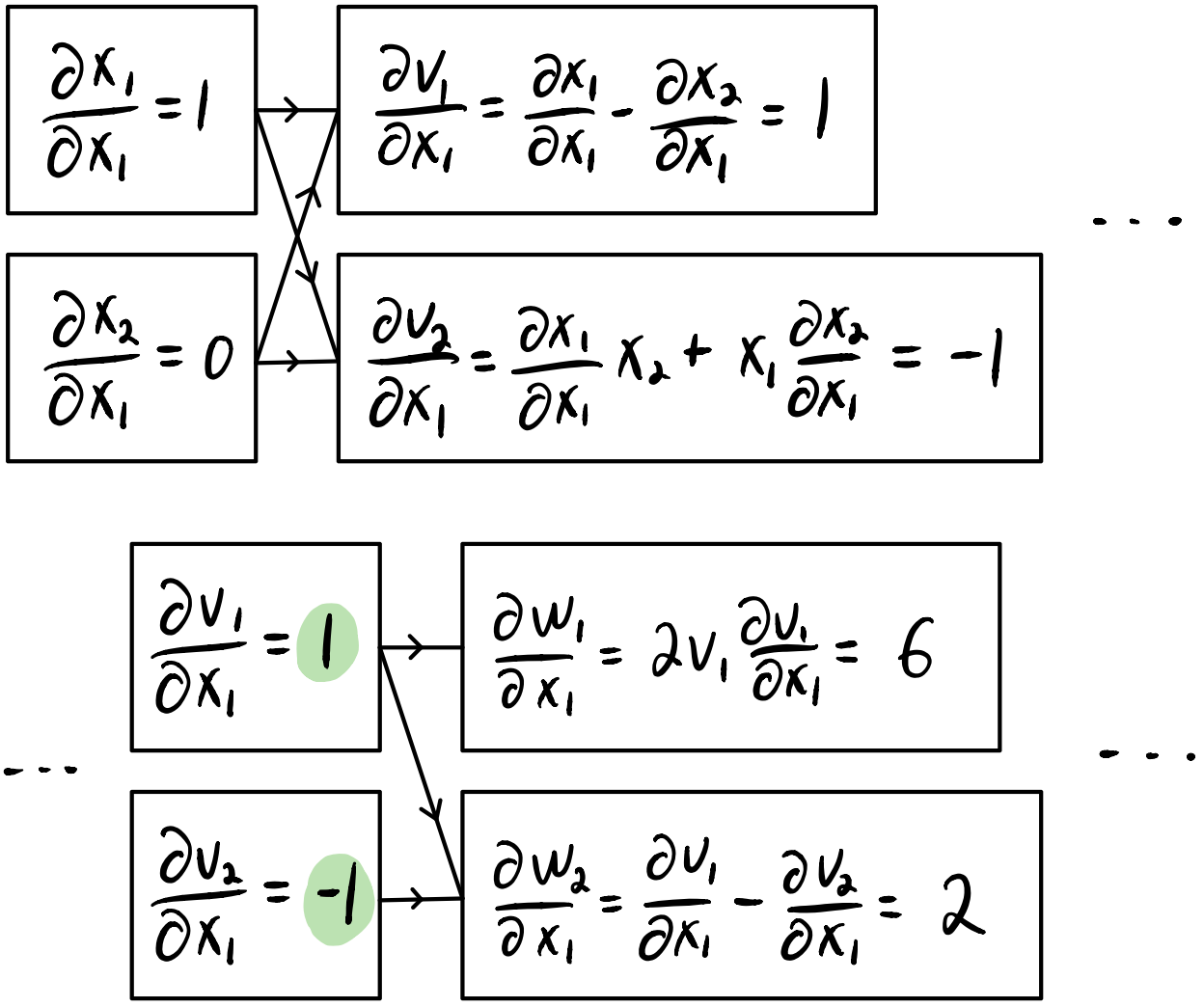


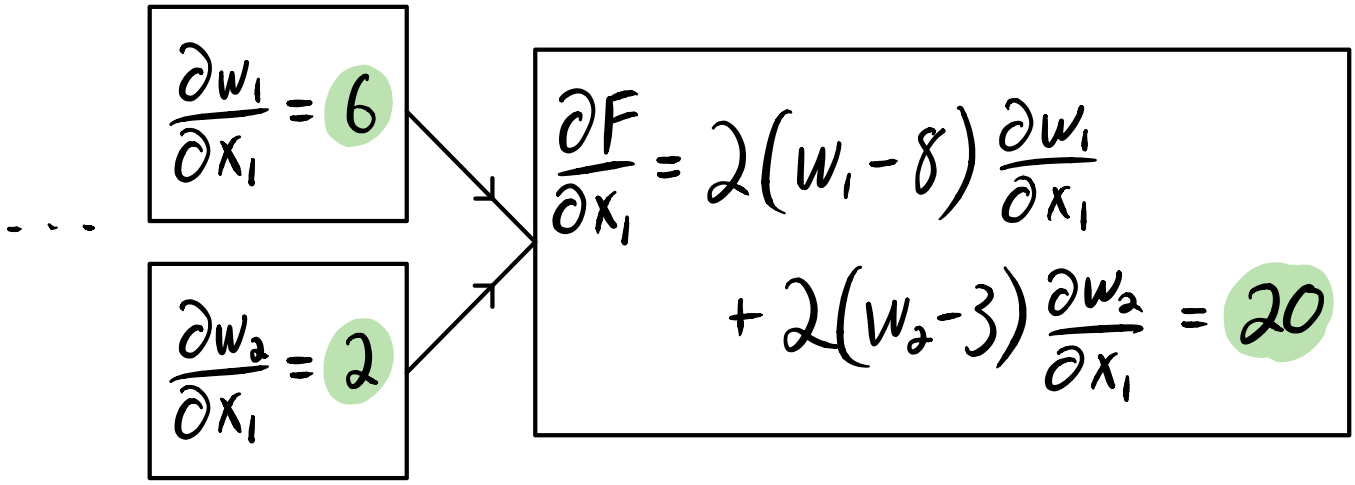


To compute F, we pass <u>forward</u> through the network. Now we want to compute

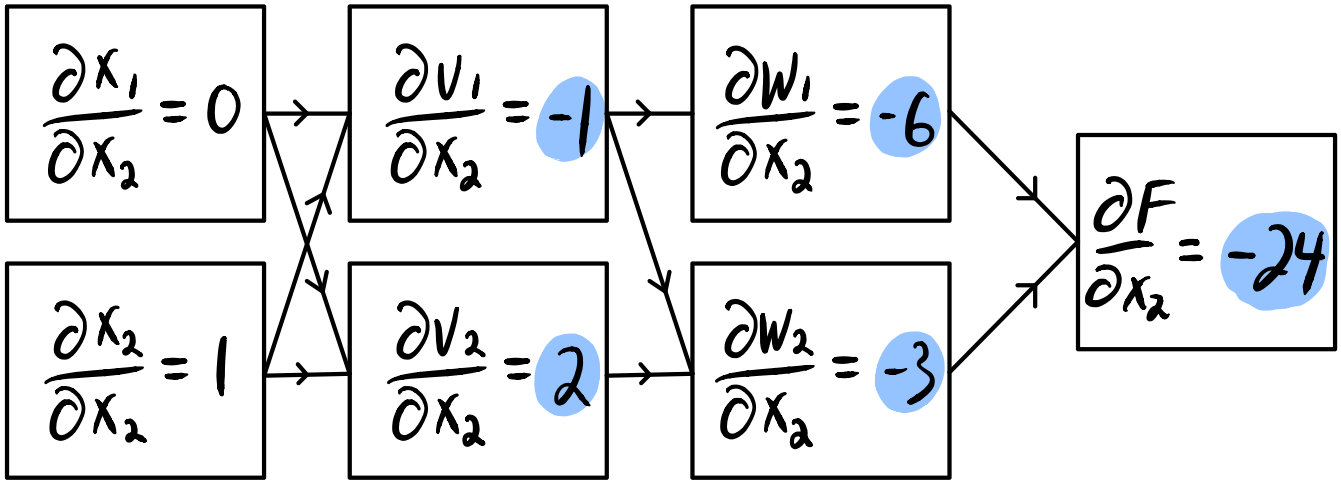
$$F'(x) = \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \frac{\partial F}{\partial x_2} \end{bmatrix}.$$

First we compute $\frac{\partial F}{\partial x_1}$ with a <u>forward</u> pass:





Computing $\frac{\partial F}{\partial x_2}$ requires another forward pass through the network:



Thus, forward-mode AD requires a separate pass through the network for each variable. This will be a lot of work if there are many variables, like in neural nets.

Let

$$v = F_1(x) = \begin{bmatrix} x_1 - x_2 \\ x_1 x_2 \end{bmatrix}, \quad w = F_2(v) = \begin{bmatrix} v_1^2 \\ v_1 - v_2 \end{bmatrix},$$

and $F = F_3(w) = (w_1 - 8)^2 + (w_2 - 3)^2$.

Then

$$F'(x) = F_3'(w) \cdot F_2'(v) \cdot F_1'(x)$$

$$= \begin{bmatrix} 2(w_1 - 8) & 2(w_2 - 3) \end{bmatrix} \begin{bmatrix} 2v_1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ x_2 & x_1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 4 \end{bmatrix} \left(\begin{bmatrix} 6 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 2 & 4 \end{bmatrix} \begin{bmatrix} 6 & -6 \\ 2 & -3 \end{bmatrix}$$

first pass

second pass

$$= \begin{bmatrix} 20 & -24 \end{bmatrix}.$$

Forward-mode AD $\equiv$ mult. from the right.

We could compute $F'(x)$ more efficiently by multiplying from the left:

$$F'(x) = \left(\begin{bmatrix} 2 & 4 \end{bmatrix} \begin{bmatrix} 6 & 0 \\ 1 & -1 \end{bmatrix} \right) \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 16 & -4 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 20 & -24 \end{bmatrix}$$

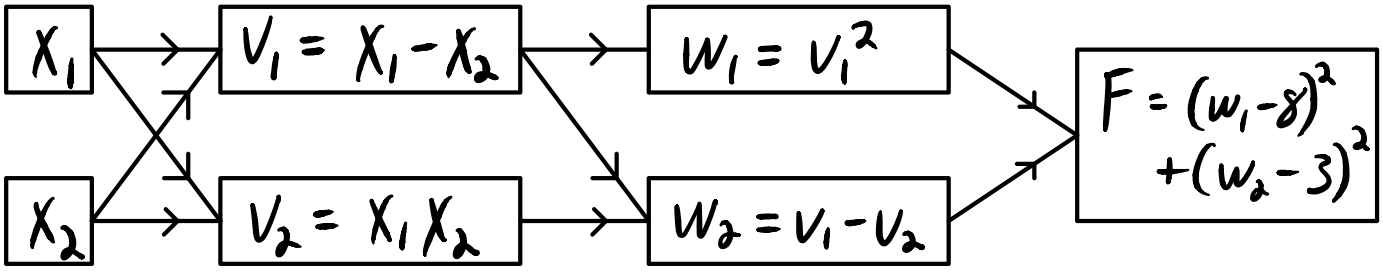
Notice that we only needed to do vector-matrix multiplication; we avoided doing the matrix-matrix multiplication. This can save us from doing a lot of extra work.

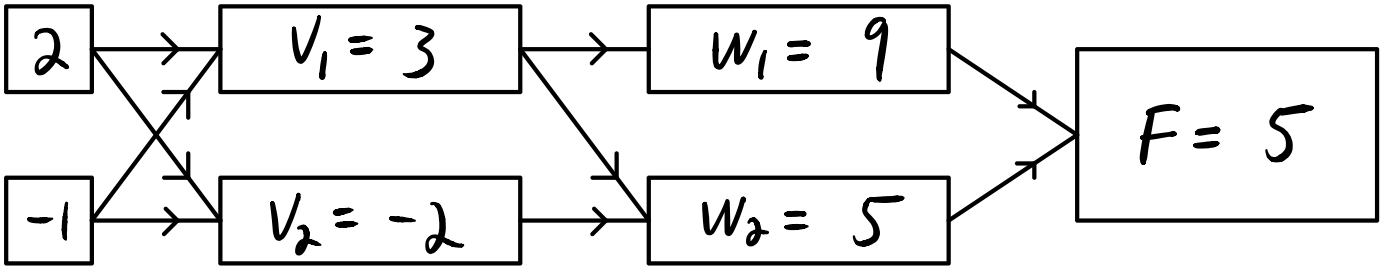
Mult. from the left $\equiv$ Backward-mode AD

Backward-mode AD

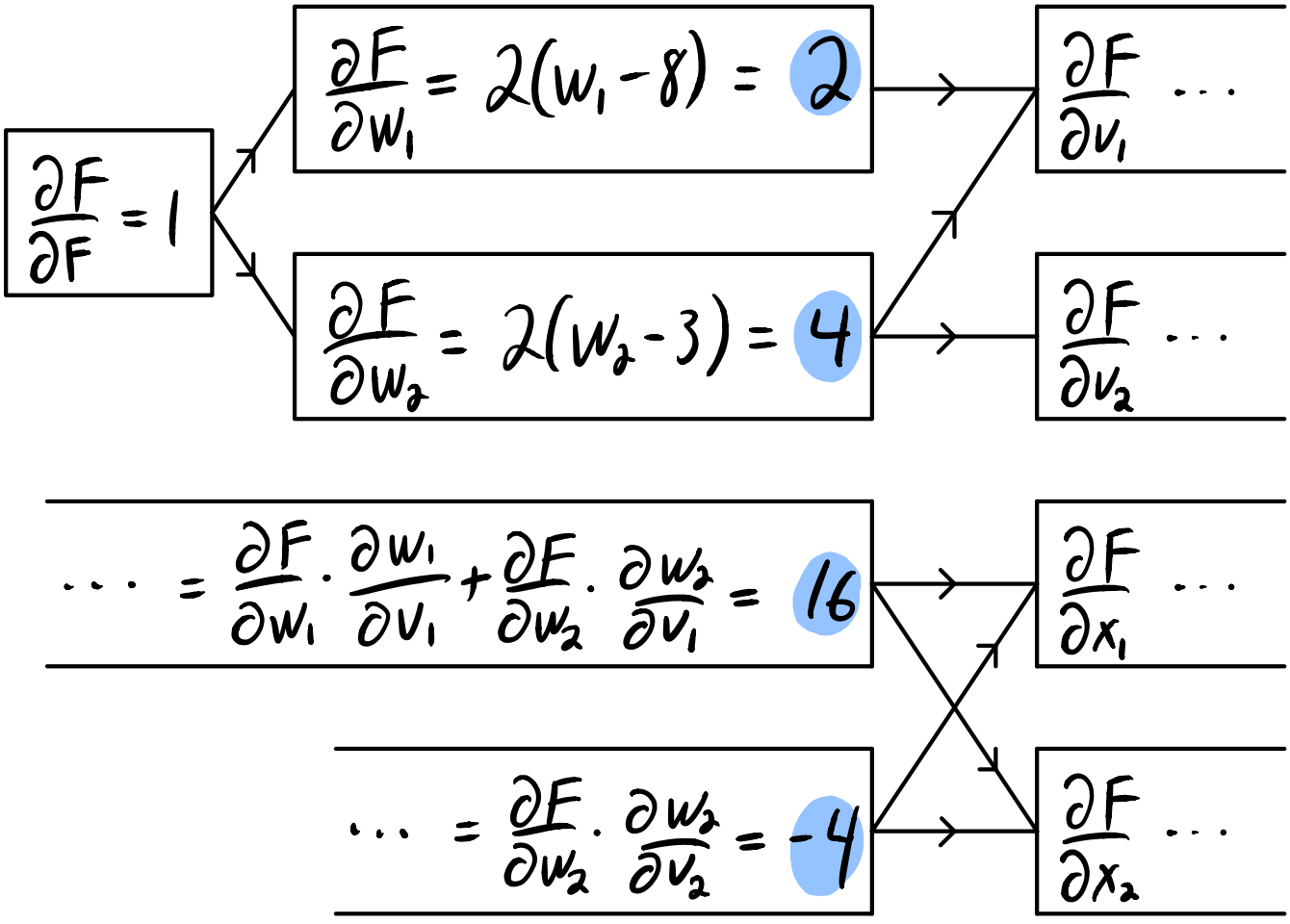
This mode is also called reverse-mode AD or backpropagation.

We still begin with a forward pass to compute $F(x)$.





Then we compute all partial derivatives with a single backward pass.



$$\cdots = \frac{\partial F}{\partial v_1} \cdot \frac{\partial v_1}{\partial x_1} + \frac{\partial F}{\partial v_2} \cdot \frac{\partial v_2}{\partial x_1} = 20$$

Single pass

$$\cdots = \frac{\partial F}{\partial v_1} \cdot \frac{\partial v_1}{\partial x_2} + \frac{\partial F}{\partial v_2} \cdot \frac{\partial v_2}{\partial x_2} = -24$$

$$F'(x) = \left(\begin{bmatrix} 2 & 4 \end{bmatrix} \begin{bmatrix} 6 & 0 \\ 1 & -1 \end{bmatrix} \right) \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 16 & -4 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 20 & -24 \end{bmatrix}$$