

Automatic Differentiation

Benoît Legat

☐ Full Width Mode ☐ Present Mode

Table of Contents

Differentiation approaches

Chain rule

Forward Differentiation

Reverse differentiation

Comparison

Discontinuity

Neural network

Differentiation approaches

We can compute partial derivatives in different ways:

1. **Symbolically**, by fixing one of the variables and differentiating with respect to the others, either manually or using a computer.
2. **Numerically**, using the formula $f'(x) \approx (f(x + h) - f(x))/h$.
3. **Algorithmically**, either forward or reverse : this is what we will explore here.

Chain rule \Leftrightarrow

Consider $f(x) = f_3(f_2(f_1(x)))$. If we don't have the expression of f_1 but we can only evaluate $f_i(x)$ or $f'(x)$ for a given x ? The chain rule gives

$$f'(x) = f'_3(f_2(f_1(x))) \cdot f'_2(f_1(x)) \cdot f'_1(x).$$

Let's define $s_0 = x$ and $s_k = f_k(s_{k-1})$, we now have:

$$f'(x) = f'_3(s_2) \cdot f'_2(s_1) \cdot f'_1(s_0).$$

Two choices here:

Forward	Reverse
$t_0 = 1$	$r_3 = 1$
$t_k = f'_k(s_{k-1}) \cdot t_{k-1}$	$r_k = r_{k+1} \cdot f'_{k+1}(s_k)$

Forward Differentiation ⇔

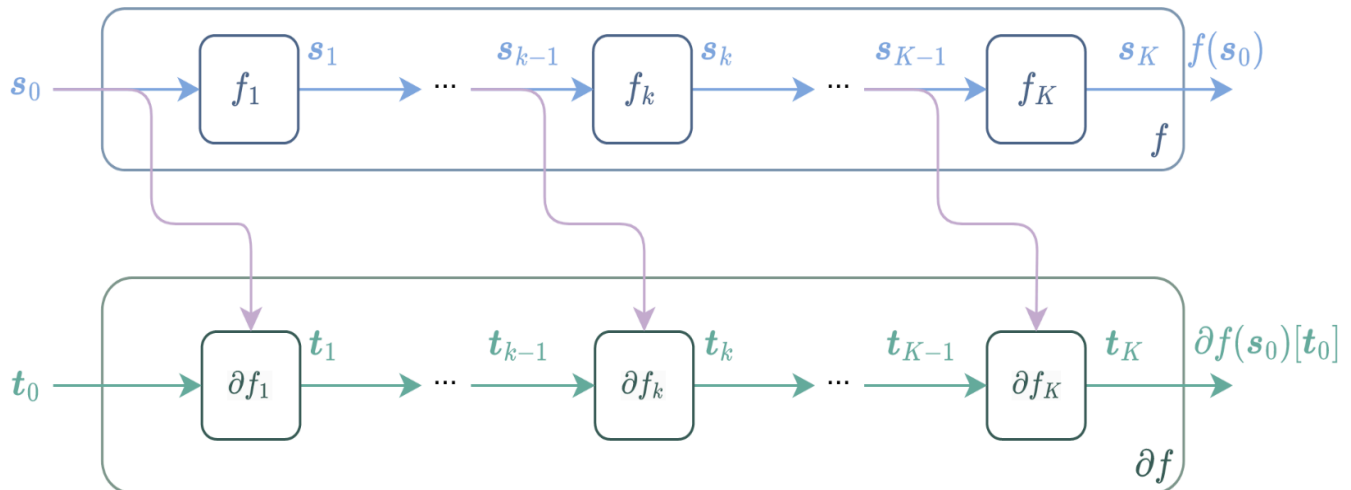


Figure 8.1

Implementation ⇔

```
1 struct Dual{T}
2     value::T # s_k
3     derivative::T # t_k
4 end
```

```
1 Base.:- (x::Dual{T}) where {T} = Dual(-x.value, -x.derivative)
```

```
1 Base.:*(x::Dual{T}, y::Dual{T}) where {T} = Dual(x.value * y.value, x.value *
y.derivative + x.derivative * y.value)
```

```
► Dual(-3, -10)
```

```
1 -Dual(1, 2) * Dual(3, 4)
```

```
f_1 (generic function with 1 method)
```

```
1 f_1(x, y) = x * y
```

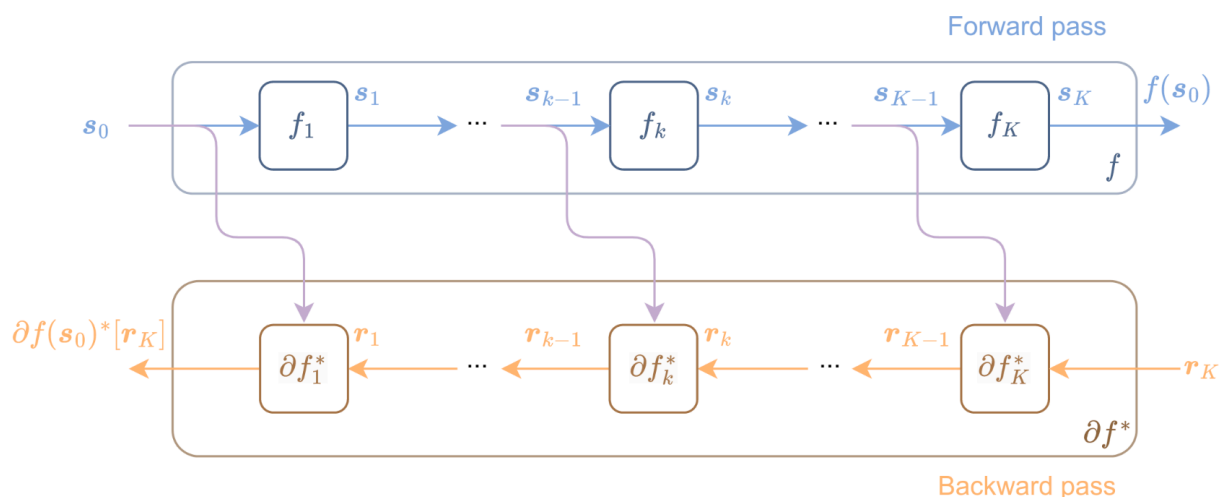
```
f_2 (generic function with 1 method)
```

```
1 f_2(s1) = -s1
```

► Dual(-3, -10)

```
1 (f_2 ◦ f_1)(Dual(1, 2), Dual(3, 4))
```

Reverse differentiation ⇔



Two different takes on the multivariate chain rule ⇔

The chain rule gives us

$$\frac{\partial f_3}{\partial x}(f_1(x), f_2(x)) = \frac{\partial f_3}{\partial s_1}(s_1, s_2) \cdot \frac{\partial s_1}{\partial x} + \frac{\partial f_3}{\partial s_2}(s_1, s_2) \cdot \frac{\partial s_2}{\partial x}$$

To compute this expression, we need the values of $g(x)$ and $h(x)$ as well as the derivatives $\partial g/\partial x$ and $\partial h/\partial x$.

Forward ⇔

$$t_3 = \frac{\partial s_3}{\partial s_1} t_1 + \frac{\partial s_3}{\partial s_2} t_2$$

- Given s_1, s_2 , computes $\frac{\partial s_3}{\partial s_1}(s_1, s_2)$ and $\frac{\partial s_3}{\partial s_2}(s_1, s_2)$
- Given t_1 and t_2 , computes $\partial f_3/\partial x$

Reverse ⇔

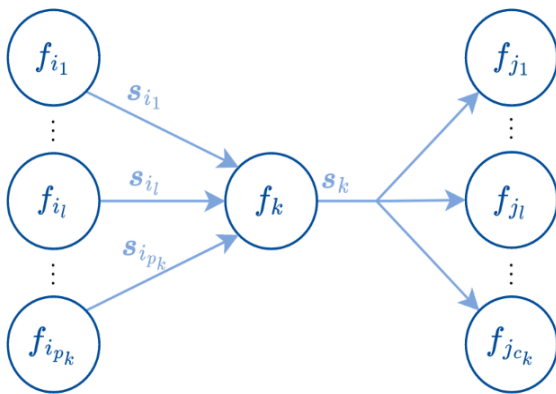
- Given s_1, s_2 , computes $\frac{\partial s_3}{\partial s_1}(s_1, s_2)$ and $\frac{\partial s_3}{\partial s_2}(s_1, s_2)$
- Given $r_3 = \partial s_K/\partial s_3$
 - Add $r_3 \cdot (\partial s_3/\partial s_1)$ to r_1
 - Add $r_3 \cdot (\partial s_3/\partial s_2)$ to r_2

When using automatic differentiation, don't forget that we must always evaluate the derivatives. For the following example we choose to evaluate it in $x = 3$

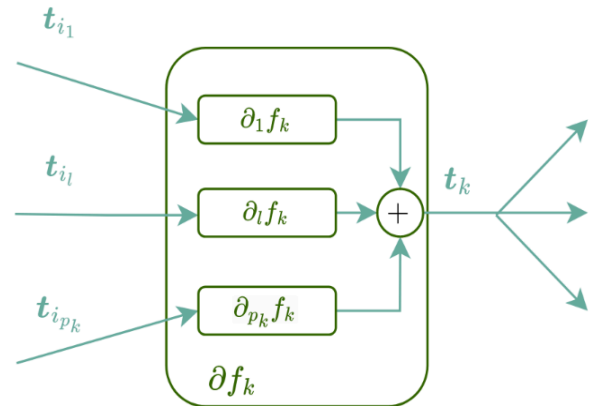
► Apply the automatic differentiation to $s_3 = f_3(s_1, s_2) = s_1 + s_2$, with $s_1 = f_1(x) = x$ and $s_2 = f_2(x) = x^2$

Forward tangents \Rightarrow

Forward pass

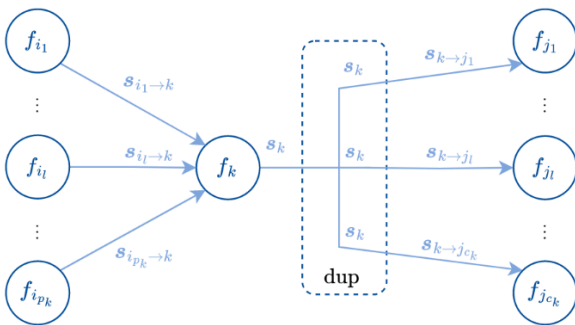


Forward mode

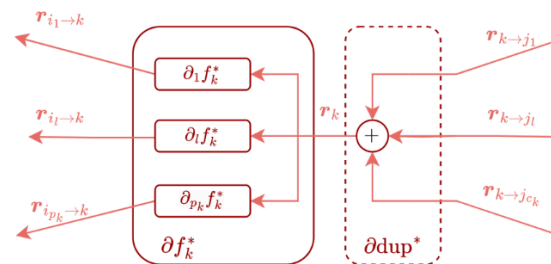


Reverse tangents \Rightarrow

Forward pass

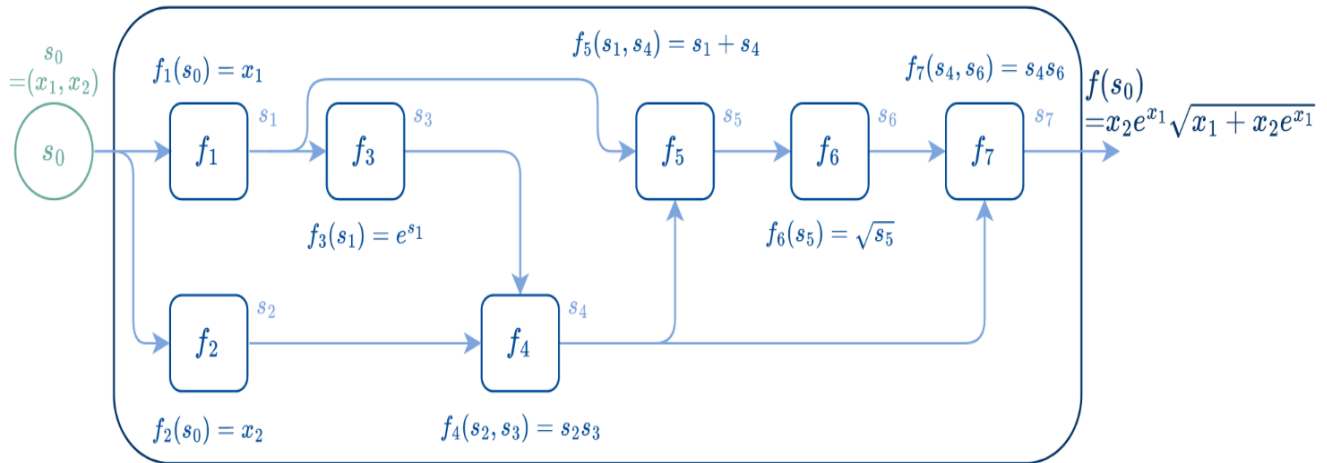


Reverse mode



► Why is ∂dup^* a sum ?

Expression graph ⇔



► Can this directed graph have cycles ?

► What happens if f_4 is handled before f_5 in the backward pass ?

► How to prevent this from happening ?

Comparison ⇔

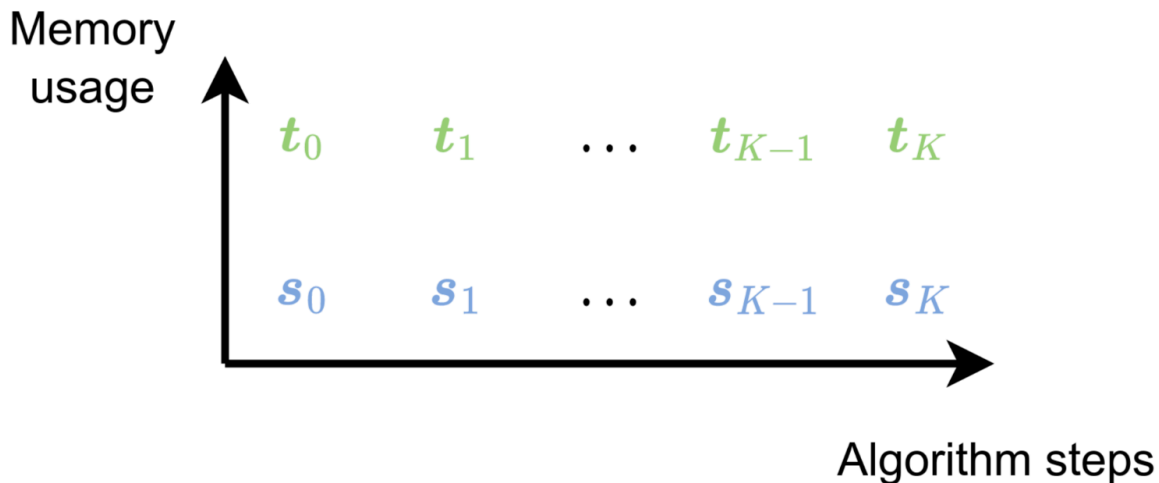
- Forward mode of $f(x)$ with dual numbers $\text{Dual.}(x, v)$ computes Jacobian-Vector Product (JVP) $J_f(x) \cdot v$
- Reverse mode of $f(x)$ computes Vector-Jacobian Product (VJP) $v^\top J_f(x)$ or in other words $J_v(x)^\top v$

► How can we compute the full Jacobian ?

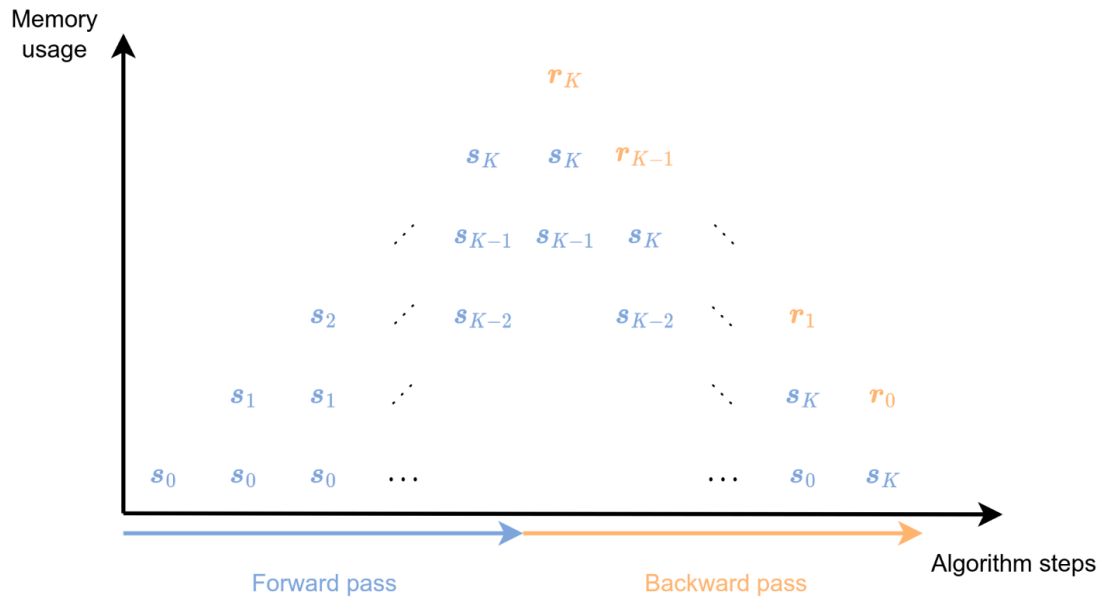
► When is each mode faster than the other one to compute the full Jacobian ?

► When is the speed of numerical differentiation comparable to autodiff ?

Memory usage of forward mode ⇔

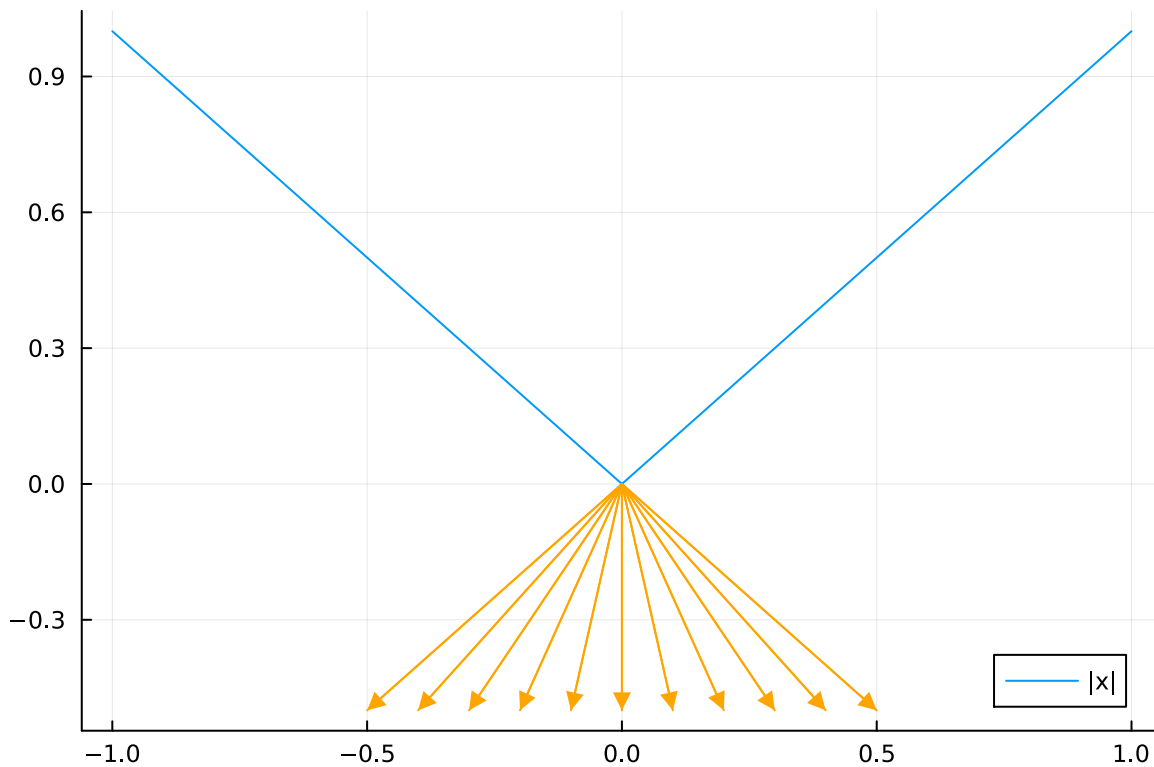


Memory usage of reverse mode \Leftrightarrow



Discontinuity ⇔

► Is the function $|x|$ is differentiable at $x = 0$?



► What about returning a convex combination of the derivative from the left and right ?

Forward mode ⇔

```
abs (generic function with 1 method)
```

```
1 abs(x) = ifelse(x < 0, -x, x)
```

```
abs_bis (generic function with 1 method)
```

```
1 abs_bis(x) = ifelse(x > 0, x, -x)
```

```
1 Base.isless(x::Dual, y::Real) = isless(x.value, y)
```

```
1 Base.isless(x::Real, y::Dual) = isless(x, y.value)
```

► Dual(0, 1)

```
1 abs(Dual(0, 1))
```

► Dual(0, -1)

```
1 abs_bis(Dual(0, 1))
```

Neural network ⇔

Two equivalent approaches, b_k is a **column** vector, S_i, X, W_i, Y are matrices.

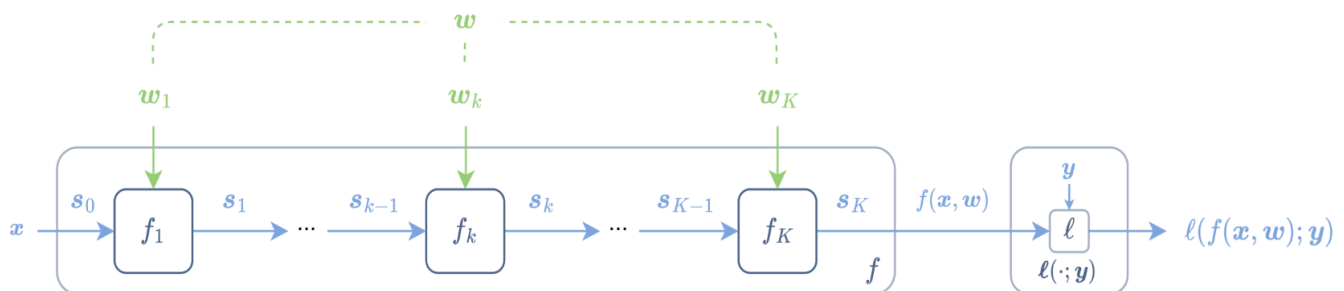
Right-to-left ⇔

$$\begin{aligned} S_0 &= X \\ S_{2k-1} &= W_k S_{2k-2} + b_k \mathbf{1}^\top \\ S_{2k} &= \sigma(S_{2k-1}) \\ S_{2H+1} &= W_{k+1} S_{2H} \\ S_{2H+2} &= \ell(S_{2H+1}; Y) \end{aligned}$$

Left-to-right ⇔

$$\begin{aligned} S_0 &= X \\ S_{2k-1} &= S_{2k-2} W_k + \mathbf{1} b_k^\top \\ S_{2k} &= \sigma(S_{2k-1}) \\ S_{2H+1} &= S_{2H} W_{k+1} \\ S_{2H+2} &= \ell(S_{2H+1}; Y) \end{aligned}$$

Evaluation ⇔



Matrix multiplication (Vectorized way) ⇔

Useful: $\text{vec}(AXB) = (B^\top \otimes A)\text{vec}(X)$

$$\begin{aligned} F(X) &= AX \\ G(\text{vec}(X)) &\triangleq \text{vec}(F(X)) = (I \otimes A)\text{vec}(X) \\ J_G &= (I \otimes A) \\ J_G^\top \text{vec}(R) &= (I \times A^\top)\text{vec}(R) \\ \partial F^*[R] &= \text{mat}(J_G^\top \text{vec}(R)) = A^\top R \end{aligned}$$

► How should we store the Jacobian in the forward pass to save it for the backward pass ?

Matrix multiplication (Scalar product way) ⇔

The adjoint of a linear map A for a given scalar product $\langle \cdot, \cdot \rangle$ is the linear map A^* such that

$$\forall x, y, \quad \langle A(x), y \rangle = \langle x, A^*(y) \rangle.$$

For the scalar product

$$\langle X, Y \rangle = \sum_{i,j} X_{ij} Y_{ij} = \langle \text{vec}(X), \text{vec}(Y) \rangle, \quad A^* = A^\top$$

Now, given a forward tangent T and a reverse tangent R

$$\langle AT, R \rangle = \langle T, A^\top R \rangle$$

so the backward pass computes $A^\top R$.

► How to prove that $A^* = A^\top$?

Broadcasting (Vectorized way) ⇔

Consider applying a scalar function f (e.g. **tanh**) to each entry of a matrix X .

$$\begin{aligned} (F(X))_{ij} &= f(X_{ij}) = f.(X) \\ G(\text{vec}(X)) &\triangleq \text{vec}(F(X)) = \text{vec}(f.(X)) \\ J_G &= \text{Diag}(\text{vec}(f'.(X))) \\ J_G^\top \text{vec}(T) &= \text{Diag}(\text{vec}(f'.(X))) \text{vec}(T) \\ \partial F[T] &= \text{mat}(J_G^\top \text{vec}(T)) = f'.(X) \odot T \\ J_G^\top \text{vec}(R) &= \text{Diag}(\text{vec}(f'.(X))) \text{vec}(R) \\ \partial F^*[R] &= \text{mat}(J_G^\top \text{vec}(R)) = f'.(X) \odot R \end{aligned}$$

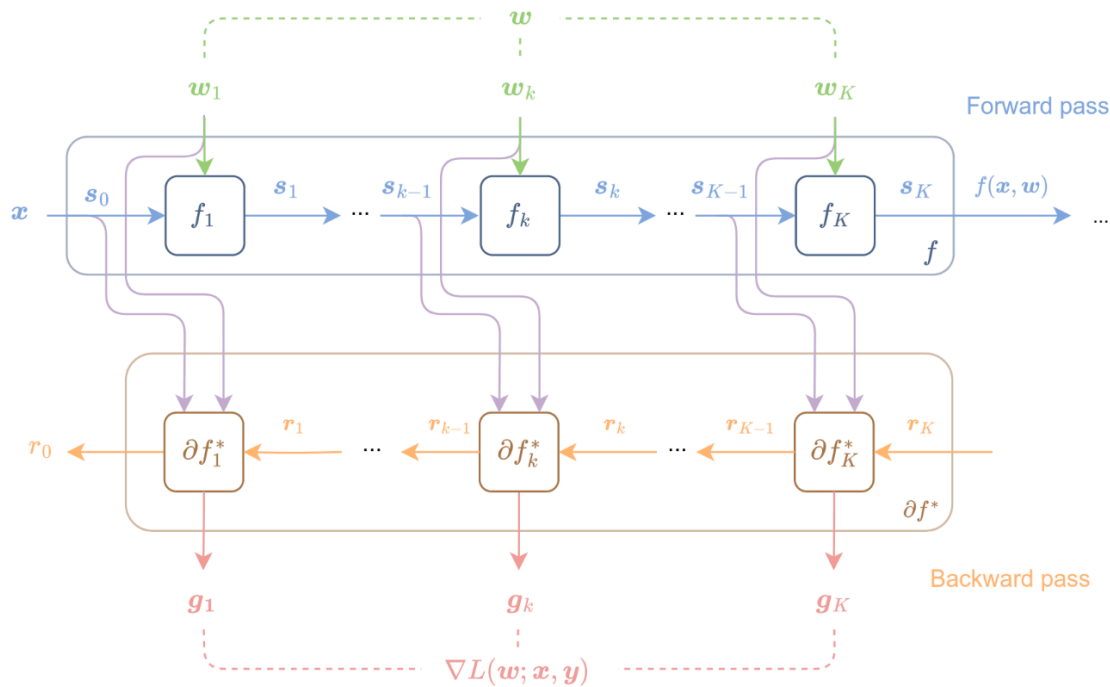
Broadcasting (Scalar product way) ⇔

$$\langle f'.(X) \odot T, R \rangle = \langle T, f'.(X) \odot R \rangle.$$

► Let $A(X) = B \odot X$, what is the adjoint A^* ?

► What should be saved for the backward pass ?

Putting everything together ⇔



Product of Jacobians ⇔

Suppose that we need to differentiate a composition of functions: $(f_n \circ f_{n-1} \circ \dots \circ f_2 \circ f_1)(w)$. For each function, we can compute a jacobian given the value of its input. So, during a forward pass, we can compute all jacobians. We now just need to take the product of these jacobians:

$$J_n J_{n-1} \dots J_2 J_1$$

While the product of matrices is associative, its computational complexity depends on the order of the multiplications! Let $d_i \times d_{i-1}$ be the dimension of J_i .

► What is the complexity of forward mode

► What is the complexity of reverse mode

► What about the complexity of meeting in the middle between k and $k + 1$?

► Which mode should be used depending on the d_i ?

► What about neural networks ?

Acknowledgements and further readings

- Dual is inspired from [ForwardDiff](#)
- Node is inspired from [micrograd](#)
- [Here](#) is a good intro to AD
- Figures are from the [The Elements of Differentiable Programming book](#)

The End

Utils

```
1 using Plots, PlutoUI, PlutoUI.ExperimentalLayout, HypertextLiteral; @html, @html_str
   PlutoTeachingTools
```

img (generic function with 3 methods)

qa (generic function with 2 methods)