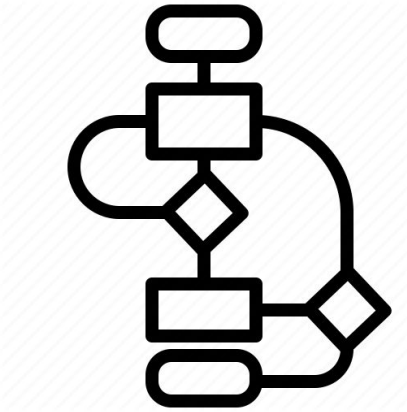

LINMA2472 - Algorithm in data science

Simon Desmidt

Academic year 2025-2026 - Q1

UCLouvain

Contents

Automatic differentiation

The Automatic differentiation is an algorithmic technique to compute automatically the derivative (gradient) of a function defined in a computer program. Unlike symbolic differentiation (done by hand) and numerical differentiation (finite difference approximation), automatic differentiation exploits the fact that every function can be decomposed into a sequence of elementary operations (addition, multiplication, sine, exponential, etc.) and so that we can apply the chain rule to compute the derivative of the whole function. Thus we can compute the gradient of a function exactly and efficiently.
Automatic differentiation is widely used in machine learning because for the neural networks, we need to compute the gradient of a loss function with respect to the parameters of the model (weights and biases) to update them during the training process and it would be difficult to compute this manually for each node.

1.1 Chain rule

There is two ways to apply the chain rule to compute the gradient of a function: forward differentiation and backward differentiation. Suppose that we have a composition of m functions. The chain rule gives us:

$$f'(x) = f'_m(f_{m-1}(f_{m-2}(...f_1(x)...))) \cdot ... \cdot f'_2(f_1(x)) \cdot f'_1(x) \tag{1.1}$$

Let's define:

$$\begin{cases} s_0 &= x \\ s_k &= f_k(s_{k-1}) \end{cases} \tag{1.2}$$

We thus get:

$$f'(x) = f'_m(s_{m-1}) \cdot ... \cdot f'_2(s_1) \cdot f'_1(s_0) \tag{1.3}$$

Based on this, we can define the forward and backward differentiation algorithms.

1.1.1 Forward differentiation

Also called forward mode, this algorithm consists in propagating forward the derivative and the values at the same time. It can be represented by this graph where the blue part represents the values and the green part the derivatives:

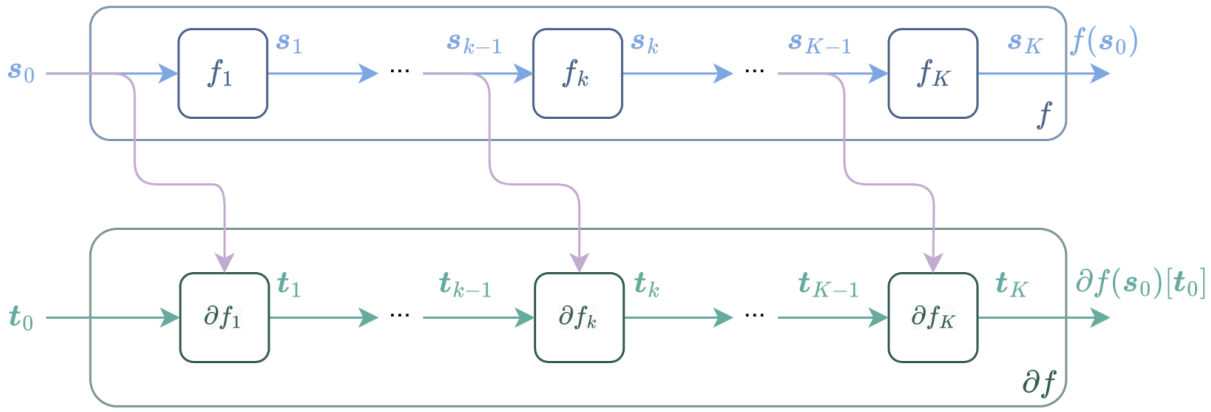

Figure 1.1: Forward differentiation

And it can be computed with the following recurrence relation:

$$\begin{cases} t_0 &= 1 \\ t_k &= f'_k(s_{k-1}) \cdot t_{k-1} \end{cases} \tag{1.4}$$

It is simple to implement and very efficient for functions with a small number of input variables. However, it becomes inefficient for functions with a large number of input variables because we need to compute the derivative for each input variable separately. So in practice for neural networks where we have a large number of input variables (weights and biases), we use the backward differentiation.

1.1.2 Backward differentiation

Also called backward mode, this algorithm consists in propagating the derivative backward and the values forward at the same time. It can be represented by this graph where the blue part represents the values and the orange part the derivatives:

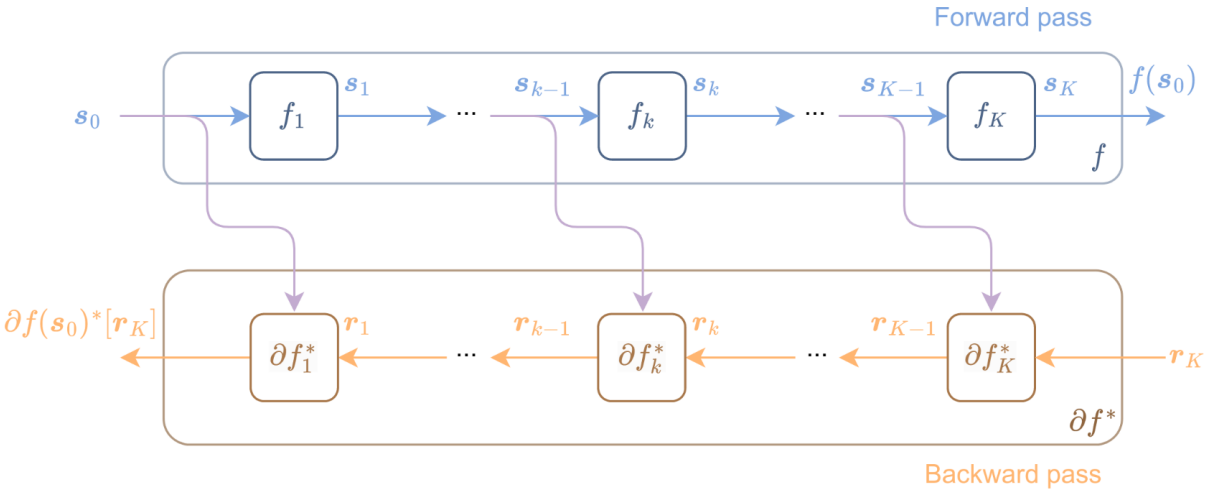

Figure 1.2: Backward differentiation

The idea is to compute all the intermediate values s_k in a forward pass and then compute the derivatives r_k based on the output in a backward pass. It can be computed

with the following recurrence relation:

$$\begin{cases} r_m & = 1 \\ r_k & = r_{k+1} \cdot f'_{k+1}(s_k) \end{cases} \tag{1.5}$$

This method is more complex to implement but it is very efficient for functions with a large number of input variables and a small number of output variables typically 1, the loss function.