

Course: LINMA2472: Algorithms in Data Science

Professor: Benoît Legat (benoit.legat@uclouvain.be)

Teaching assistants: Adeline Colson (adeline.colson@uclouvain.be), Sophie Lequeu (sophie.lequeu@uclouvain.be), Bastien Massion (bastien.massion@uclouvain.be)

Reverse Automatic Differentiation and the Adjoint Jacobian

List of exercises

Reminder: Let function F be defined by the composition

$$F := f_L \circ f_{L-1} \circ \cdots \circ f_1.$$

Then by the Chain Rule,

$$J_F(x) = J_{f_L}(s_{L-1}) \cdots J_{f_2}(s_1) \cdot J_{f_1}(s_0),$$

where $s_0 := x$ and $s_k := f_k(s_{k-1})$.

Application to Automatic Differentiation:

- Forward AD:

$$J_F(x) \cdot u = J_{f_L}(s_{L-1}) \cdots J_{f_2}(s_1) \cdot J_{f_1}(s_0) \cdot u. \quad (\text{JVP})$$

We compute

$$t_k := J_{f_k}(s_{k-1}) \cdot t_{k-1}, \quad \text{for } k = 1, \dots, L \text{ and with } t_0 := u.$$

- Reverse AD:

$$J_F(x)^\top \cdot v = J_{f_1}(s_0)^\top \cdot J_{f_2}(s_1)^\top \cdots J_{f_L}(s_{L-1})^\top \cdot v. \quad (\text{VJP})$$

We compute

$$r_k := J_{f_{k+1}}(s_k)^\top \cdot r_{k+1}, \quad \text{for } k = L-1, \dots, 0 \text{ and with } r_L := v.$$

In AD, for computational and storage reasons, we consider Jacobians (and Adjoint Jacobians) as **Linear Maps**, and not as matrices.

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

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

1. Adjoint Jacobian of matrix multiplication on the right.

Consider the elementary operation

$$F_1 : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times p}, \quad F_1(X) = XB,$$

for some $B \in \mathbb{R}^{n \times p}$ and the Frobenius inner product $\langle X, Y \rangle = \text{tr}(XY^\top)$.

The action of the Jacobian of this elementary operation on a perturbation U is given by $J_{F_1}(X)[U] = UB$, which can be viewed as the linear map $\mathcal{A}(U) = UB$ applied on the direction U . Using the definition above, find the Adjoint Jacobian of operation F_1 .

$\mathcal{A}^*(V) =$

2. Adjoint Jacobian of the Hadamard product.

Consider the elementary operation

$$F_2 : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times n}, \quad F_2(X) = X \odot B,$$

for some $B \in \mathbb{R}^{m \times n}$, where $\odot$ denotes the Hadamard (element-wise) product, and the Frobenius inner product is given by $\langle X, Y \rangle = \text{tr}(XY^\top)$.

The Jacobian in the direction V can be viewed as a linear map, $J_{F_2}(X)[U] = \mathcal{A}(U)$. Find the expression of this linear map. Then, using the definition above, find the adjoint Jacobian of the operation F_2 .

$$\mathcal{A}^*(Y) =$$

3. Adjoint Jacobian of the broadcasted square.

Consider the elementary operation

$$F_3 : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times n}, \quad F_3(X) = X^{\odot 2},$$

where $X^{\odot 2}$ denotes the element-wise square of X , and the Frobenius inner product is given by $\langle X, Y \rangle = \text{tr}(XY^\top)$.

The Jacobian in the direction U can be viewed as a linear map, $J_{F_3}(X)[U] = \mathcal{A}_X(U)$. Find the expression of this linear map. Then, using the definition above, find the adjoint Jacobian of the operation F_3 .

$$\mathcal{A}_X^*(Y) =$$

4. Adjoint Jacobian of the sum of all matrix elements.

Consider the elementary operation

$$F_4 : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}, \quad F_4(X) = \sum_{i=1}^m \sum_{j=1}^n X_{ij},$$

where $\mathbb{R}^{m \times n}$ is equipped with the Frobenius inner product $\langle X, Y \rangle = \text{tr}(XY^\top)$ and $\mathbb{R}$ with the usual scalar product.

The Jacobian in the direction U can be viewed as a linear map, $J_{F_4}(X)[U] = \mathcal{A}(U)$. Find the expression of this linear map. Then, using the definition above, find the adjoint Jacobian of the operation F_4 .

$$\mathcal{A}^*(v) =$$

5. Adjoint Jacobian of the softmax.

Consider the elementary operation

$$F_5 : \mathbb{R}^{1 \times n} \rightarrow \mathbb{R}^{1 \times n}, \quad F_5(X) = \text{softmax}(X),$$

where

$$(F_5(X))_i = \frac{e^{X_i}}{\sum_{j=1}^n e^{X_j}},$$

and $\mathbb{R}^{1 \times n}$ is equipped with the Frobenius inner product $\langle X, Y \rangle = \text{tr}(XY^\top)$.

Let

$$S = \text{softmax}(X).$$

The Jacobian in the direction U can be viewed as a linear map, $J_{F_5}(X)[U] = \mathcal{A}_X(U)$. Find the expression of this linear map. Then, using the definition above, find the adjoint Jacobian of the operation F_5 .

$$\mathcal{A}_X^*(V) =$$

6. FlashAttention in Transformers.

In this exercise, we introduce the attention mechanism used in transformers, then derive a memory-efficient variant known as *FlashAttention*. You will complete a Julia implementation in which some lines have been removed.

Let $Q, K, V \in \mathbb{R}^{n \times d}$ be the query, key, and value matrices. The standard scaled dot-product attention is defined by

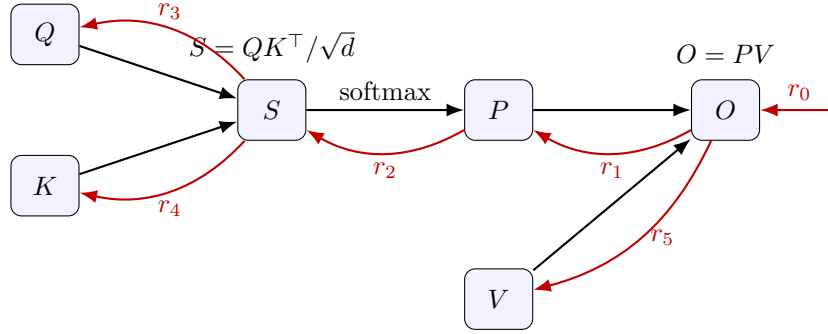
$$S = \frac{QK^\top}{\sqrt{d}}, \quad P = \text{softmax}(S) \text{ (row-wise)}, \quad O = PV.$$

The naive implementation materializes the two $n \times n$ matrices S and P , which becomes expensive when n is large. FlashAttention avoids storing these matrices. Instead, it computes the output in tiles, stores only the output O and one scalar per row

$$L_i = \log \sum_j \exp(S_{ij}),$$

and recomputes the needed blocks during the backward pass.

- **Attention.** Explain why the naive implementation of attention requires $O(n^2)$ memory. Then write the formula for the backward pass with respect to Q , K , and V , based on the expression graph given hereafter.



- **The key identity.** In the backward pass, when computing r_2 , the naive softmax derivative contains the term

$$D_i = \sum_j P_{ij} (r_1)_{ij}.$$

Show that this term can also be written as

$$D_i = \sum_k (r_0)_{ik} O_{ik}.$$

Explain why this identity makes it possible to compute the backward pass without storing P .

Note about the implementation, for the following two questions.

To compute attention without storing the full matrices S and P , we process the keys and values tile by tile, accumulating the numerator and denominator of

$$O_{i,:} = \frac{\sum_j \exp(S_{ij}) V_{j,:}}{\sum_j \exp(S_{ij})}.$$

For numerical stability, we subtract a row-wise maximum m_i from the scores before exponentiating. This leaves the ratio unchanged, since the factor $\exp(-m_i)$ cancels between the numerator and denominator. Because the scores are computed tile by tile, we maintain a *running maximum*. After processing a set of key indices J , the running variables satisfy

$$m_i = \max_{j \in J} S_{ij}, \quad l_i = \sum_{j \in J} \exp(S_{ij} - m_i), \quad \tilde{O}_{i,:} = \sum_{j \in J} \exp(S_{ij} - m_i) V_{j,:}.$$

The unnormalized accumulator $\tilde{O}$ is stored in the variable `O` in the code.

When a new tile increases the maximum from m_i to m_i^{new} , we rescale the previous contributions using

$$\exp(S_{ij} - m_i^{\text{new}}) = \exp(m_i - m_i^{\text{new}}) \exp(S_{ij} - m_i).$$

Thus, we multiply both running sums by $\exp(m_i - m_i^{\text{new}})$ before adding the contributions of the new tile, computed relative to the new maximum.

After processing all tiles, we recover

$$O_{i,:} = \frac{\tilde{O}_{i,:}}{l_i}, \quad L_i = m_i + \log(l_i).$$

Tiling avoids storing the full attention matrices, while the running maximum keeps the exponential computations numerically stable.

- **FlashAttention forward pass.** In the file `flash.jl`, the function `attn_flash_fwd` computes the forward pass by iterating over tiles of keys and values. Complete the missing parts of the algorithm so that the function returns the correct output O and the vector L .
- **FlashAttention backward pass.** In the file `flash.jl`, the function `attn_flash_back` recomputes each tile during the backward pass. Fill in the missing lines so that the function returns the gradients

$$r_3, \quad r_4, \quad r_5.$$

Your implementation should use the previously stored values O and L , and it should never materialize the full attention matrix.

- **Discussion.** Use the provided benchmark to compare naive attention and FlashAttention as a function of sequence length. Explain why the flash version can be slower for small n , even though it uses less memory.

7. Adapted from Exam of August 2026.

Let

$$f(x, Q) = x^\top Q x - x^\top b, \quad x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad Q = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

where Q is not assumed to be symmetric. We evaluate this function as

$$s = Qx, \quad t = x^\top s, \quad g = x^\top b, \quad f = t - g.$$

- Draw the corresponding expression graph and indicate the forward values. An arrow points from each argument to the operation that uses it.
- Starting with $\bar{f} = \partial f / \partial f = 1$, perform the backward pass. Give the order in which you process the operation nodes and show how the adjoint of x accumulates its three contributions.
- Give $\nabla_x f$, $\nabla_Q f$ and $\nabla_b f$, first as formulas valid for arbitrary x , Q and b , and then at the values above. Why does $\nabla_Q f$ have rank one whenever $x \neq 0$?
- More generally, suppose that a scalar-valued function has many inputs. To compute its full gradient, which mode is usually preferable: forward or reverse? Briefly justify your answer.