

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 .

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

Solution. Using the definition of adjoint recalled above and the cyclic property of the trace, we find

$$\forall U, V, \quad \langle \mathcal{A}(U), V \rangle = \langle UB, V \rangle = \text{tr}(UBV^\top) = \text{tr}(U(VB^\top)^\top) = \langle U, VB^\top \rangle = \langle U, \mathcal{A}^*(V) \rangle,$$

where therefore

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

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 .

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

Solution. First, the Jacobian in the direction U is

$$J_{F_2}(X)[U] = \left. \frac{d}{dt} F_2(X + tU) \right|_{t=0} = \left. \frac{d}{dt} ((X + tU) \odot B) \right|_{t=0} = U \odot B.$$

Hence, the corresponding linear map is

$$\mathcal{A}(U) = U \odot B.$$

Using the definition and the properties of the Frobenius inner product, we find

$$\forall U, V, \quad \langle \mathcal{A}(U), V \rangle = \langle U \odot B, V \rangle = \sum_{i,j} U_{ij} B_{ij} V_{ij} = \sum_{i,j} U_{ij} (V \odot B)_{ij} = \langle U, V \odot B \rangle = \langle U, \mathcal{A}^*(V) \rangle,$$

where therefore

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

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 .

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

Solution. First, the Jacobian in the direction U is

$$J_F(X)[U] = \left. \frac{d}{dt} F(X + tU) \right|_{t=0} = \left. \frac{d}{dt} (X + tU)^{\odot 2} \right|_{t=0} = 2X \odot U.$$

Hence, the corresponding linear map is

$$\mathcal{A}_X(U) = 2X \odot U.$$

Using the definition of adjoint recalled above and the properties of the Frobenius inner product, we find

$$\forall U, V, \quad \langle \mathcal{A}_X(U), V \rangle = \langle 2X \odot U, V \rangle = \sum_{i,j} 2X_{ij} U_{ij} V_{ij} = \sum_{i,j} U_{ij} (2X_{ij} V_{ij}) = \langle U, 2X \odot V \rangle = \langle U, \mathcal{A}_X^*(V) \rangle,$$

where therefore

$$\mathcal{A}_X^*(V) = 2X \odot V = \mathcal{A}_X(V).$$

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 .

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

Solution. First, the Jacobian in the direction U is

$$J_{F_4}(X)[U] = \left. \frac{d}{dt} F_4(X + tU) \right|_{t=0} = \left. \frac{d}{dt} \left(\sum_{i=1}^m \sum_{j=1}^n (X_{ij} + tU_{ij}) \right) \right|_{t=0} = \sum_{i=1}^m \sum_{j=1}^n U_{ij}.$$

Hence, the corresponding linear map is

$$\mathcal{A}(U) = \sum_{i=1}^m \sum_{j=1}^n U_{ij}.$$

Using the definition of adjoint recalled above, we find

$$\forall U, v, \quad \langle \mathcal{A}(U), v \rangle = v \sum_{i=1}^m \sum_{j=1}^n U_{ij} = \sum_{i=1}^m \sum_{j=1}^n U_{ij} v = \langle U, v \mathbf{1}_{m \times n} \rangle = \langle U, \mathcal{A}^*(v) \rangle,$$

where $\mathbf{1}_{m \times n}$ denotes the matrix whose entries are all equal to 1. Therefore,

$$\mathcal{A}^*(v) = v \mathbf{1}_{m \times n}.$$

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 .

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

Solution. Let

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

For each component, the Jacobian in the direction U is

$$\begin{aligned} (J_{F_5}(X)[U])_i &= \left. \frac{d}{dt} \frac{e^{X_i + tU_i}}{\sum_{j=1}^n e^{X_j + tU_j}} \right|_{t=0} \\ &= \frac{e^{X_i} U_i \sum_{j=1}^n e^{X_j} - e^{X_i} \sum_{j=1}^n e^{X_j} U_j}{\left(\sum_{j=1}^n e^{X_j} \right)^2} \\ &= S_i U_i - S_i \sum_{j=1}^n S_j U_j. \end{aligned}$$

Therefore,

$$J_{F_5}(X)[U] = S \odot U - \langle S, U \rangle S.$$

Hence, the corresponding linear map is

$$\mathcal{A}_X(U) = S \odot U - \langle S, U \rangle S.$$

Using the definition of adjoint recalled above, we find

$$\begin{aligned} \langle \mathcal{A}_X(U), V \rangle &= \langle S \odot U - \langle S, U \rangle S, V \rangle \\ &= \langle S \odot U, V \rangle - \langle S, U \rangle \langle S, V \rangle \\ &= \langle U, S \odot V \rangle - \langle U, \langle S, V \rangle S \rangle \\ &= \langle U, S \odot V - \langle S, V \rangle S \rangle \\ &= \langle U, \mathcal{A}_X^*(V) \rangle. \end{aligned}$$

Therefore,

$$\mathcal{A}_X^*(V) = S \odot V - \langle S, V \rangle S = \mathcal{A}_X(V).$$

It can also be written as

$$\mathcal{A}_X^*(V) = S \odot (V - \langle S, V \rangle \mathbf{1}^\top).$$

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

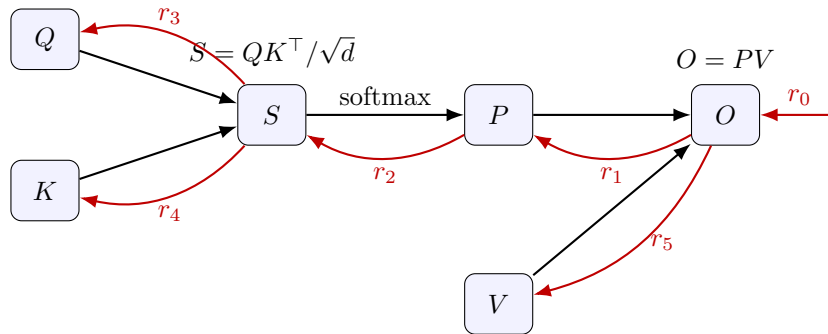
$$S = \frac{QK^\top}{\sqrt{d}}, \quad P = \text{softmax}(S) \text{ (row-wise)}, \quad O = PV.$$

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.



Solution. The naive attention computation forms

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

Since both S and P are dense $n \times n$ matrices, the memory cost is $O(n^2)$.

For the backward pass, if r_0 is the reverse tangent (at node O), then

$$r_5 = P^\top r_0, \quad r_1 = r_0 V^\top, \quad r_2 = P \odot (r_1 - ((P \odot r_1) \mathbf{1}) \mathbf{1}^\top).$$

Note each line of $D := (P \odot r_1)\mathbf{1}$ is $D_i := [(P \odot r_1)\mathbf{1}]_i = \sum_j P_{ij}(r_1)_{ij}$ (see exercise 5 for the adjoint jacobian of the softmax function applied to a row vector). Finally

$$r_3 = \frac{r_2 K}{\sqrt{d}}, \quad r_4 = \frac{r_2^\top Q}{\sqrt{d}}.$$

- **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 .

Solution. Start from

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

Since $r_1 = r_0 V^\top$, we have

$$(r_1)_{ij} = \sum_k (r_0)_{ik} V_{jk}.$$

Therefore

$$D_i = \sum_j P_{ij} \sum_k (r_0)_{ik} V_{jk} = \sum_k (r_0)_{ik} \sum_j P_{ij} V_{jk}.$$

But

$$O_{ik} = \sum_j P_{ij} V_{jk},$$

so

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

Hence

$$D = \text{rowsum}(r_0 \odot O),$$

which only requires O and r_0 , not 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 `0` 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 .

Solution. The forward pass processes the keys and values by tiles. For each block j , it computes

$$S_{\text{block}} = \frac{QK_j^\top}{\sqrt{d}},$$

updates the running row-wise maximum m , rescales the running sum l , and accumulates the output O . At the end,

$$O \leftarrow O./l, \quad L = m + \log(l).$$

So the missing logic is:

- initialize $O = 0$, $m = -\text{Inf}$, $l = 0$;
- for each tile:
 - * compute $S = Q * K[jj, :] ./ \text{sqrt}(d)$;
 - * compute $m_{\text{new}} = \max(m, \text{vec}(\text{maximum}(S, \text{dims}=2)))$;
 - * compute $P_{\text{blk}} = \exp(S ./ m_{\text{new}})$;
 - * compute $\text{corr} = \exp(m ./ m_{\text{new}})$;
 - * update l and O ;
 - * set $m = m_{\text{new}}$;
- normalize with $O ./= l$ and return O, 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.

Solution. First compute

$$D = \text{rowsum}(r_0 \odot O).$$

Then for each tile j ,

$$S_{\text{block}} = \frac{QK_j^\top}{\sqrt{d}}, \quad P_{\text{block}} = \exp(S_{\text{block}} - L),$$

and update

$$\begin{aligned} (r_5)_j &+= P_{\text{block}}^\top r_0, \\ r_{1,\text{block}} &= r_0 V_j^\top, \\ r_{2,\text{block}} &= P_{\text{block}} \odot (r_{1,\text{block}} - D), \\ r_3 &+= \frac{r_{2,\text{block}} K_j}{\sqrt{d}}, \quad (r_4)_j += \frac{r_{2,\text{block}}^\top Q}{\sqrt{d}}. \end{aligned}$$

This returns the same gradients as the naive backward pass, without storing 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.

Solution. FlashAttention can be slower for small n because it does extra work: it recomputes blocks of scores in the backward pass, manages tiled updates and rescaling, and trades arithmetic for reduced memory traffic.

At small sequence lengths, that overhead can dominate, while the naive version benefits from simpler dense matrix operations. The flash version becomes advantageous when n is large enough that the $O(n^2)$ memory traffic of the naive implementation becomes the bottleneck.

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.

Solution. A valid reverse topological order is f, g, t , then s . From $f = t - g$, we obtain

$$\bar{t} += \bar{f} = 1.$$

and

$$\bar{g} += (-1) \cdot \bar{f} = -1.$$

From $g = x^\top b$, we obtain

$$\bar{x} += \bar{g} b = \begin{bmatrix} -3 \\ 0 \end{bmatrix}, \quad \bar{b} += \bar{g} x = \begin{bmatrix} -1 \\ -2 \end{bmatrix}.$$

From $t = x^\top s$, we obtain

$$\bar{x} += \bar{t} s = \begin{bmatrix} 5 \\ 11 \end{bmatrix}, \quad \bar{s} += \bar{t} x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Finally, from $s = Qx$,

$$\bar{Q} += \bar{s} x^\top = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad \bar{x} += Q^\top \bar{s} = \begin{bmatrix} 7 \\ 10 \end{bmatrix}.$$

Hence the three contributions to $\bar{x}$ accumulate to

$$\bar{x} = \begin{bmatrix} -3 \\ 0 \end{bmatrix} + \begin{bmatrix} 5 \\ 11 \end{bmatrix} + \begin{bmatrix} 7 \\ 10 \end{bmatrix} = \begin{bmatrix} 9 \\ 21 \end{bmatrix}.$$

- 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 $x \neq 0$?

Solution. The input adjoints give

$$\nabla_x f = (Q + Q^\top)x - b, \quad \nabla_Q f = x x^\top, \quad \nabla_b f = -x.$$

At the specified values,

$$\nabla_x f = \begin{bmatrix} 12 \\ 21 \end{bmatrix}, \quad \nabla_Q f = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}, \quad \nabla_b f = \begin{bmatrix} -3 \\ 0 \end{bmatrix}.$$

For every vector v , $(xx^\top)v = x(x^\top v)$ belongs to $\text{span}(x)$. Thus $xx^\top$ has rank at most one, and its rank is exactly one when $x \neq 0$.

- 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.

Solution. Reverse mode is usually preferable. A single reverse pass computes the derivative of the scalar output with respect to every input, whereas forward mode generally requires one pass per input direction.