Useful definitions and notations

We will treat all vectors as column vectors by default. The space of real vectors of length n is denoted by $\mathbb{R}^n$, while the space of real-valued $m \times n$ matrices is denoted by $\mathbb{R}^{m \times n}$.

$X \in \mathbb{R}^n$ $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ $n \times 1$

Basic linear algebra background

The standard **inner product** between vectors x and y from $\mathbb{R}^n$ is given by

строк столбец

$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ $\begin{pmatrix} -3 \\ 5 \end{pmatrix}$

$$\boxed{\langle x, y \rangle = \underset{1\times n}{x^\top} \underset{n\times 1}{y} = \sum_{i=1}^n x_i y_i} = y^\top x = \langle y, x \rangle$$

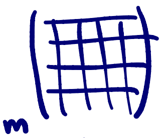
$1\cdot(-3)+2\cdot 5 =$
$= 7$

Here x_i and y_i are the scalar i-th components of corresponding vectors.

The standard **inner product** between matrices X and Y from $\mathbb{R}^{m \times n}$ is given by

след матрицы (сумма её диагональных элементов)

n столбцов



m строк

$$\boxed{\langle \underset{m\times n}{X}, \underset{m\times n}{Y} \rangle = \mathrm{tr}(\underbrace{\underset{n\times m}{X^\top} \underset{m\times n}{Y}}_{n\times n}) = \sum_{i=1}^m \sum_{j=1}^n X_{ij} Y_{ij}} = \mathrm{tr}(Y^\top X) = \langle Y, X \rangle$$

$\mathrm{tr}(I_n) = n$

$Y, X \in \mathbb{R}^{m\times n}$

The determinant and trace can be expressed in terms of the eigenvalues

$A \cdot e = \lambda \cdot e$
$\underset{n\times n}{A}\ \underset{n\times 1}{e}\ \underset{1\times 1}{\lambda}\ \underset{n\times 1}{e}$

$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ 2×3

$\det A = ?$

$$\boxed{\det A = \prod_{i=1}^n \lambda_i,} \qquad \boxed{\mathrm{tr} A = \sum_{i=1}^n \lambda_i}$$

n собств. чисел

e - собств. вектор
λ - собств. число

СПЕКТР МАТРИЦЫ

Don't forget about the cyclic property of a trace for a square matrices A, B, C, D:

$$\boxed{\mathrm{tr}(ABCD)} = \mathrm{tr}(DABC) = \mathrm{tr}(CDAB) = \mathrm{tr}(BCDA)$$

The largest and smallest eigenvalues satisfy

$$\lambda_{\min}(A) = \inf_{x \neq 0} \frac{x^\top A x}{x^\top x}, \qquad \lambda_{\max}(A) = \sup_{x \neq 0} \frac{x^\top A x}{x^\top x}$$

A^T
$\parallel$

and consequently $\forall x \in \mathbb{R}^n$ (Rayleigh quotient):

$A = A^T$

$$\lambda_{\min}(A) x^\top x \leq x^\top A x \leq \lambda_{\max}(A) x^\top x$$

$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \to \left(\begin{pmatrix} 1 \\ 2 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \end{pmatrix} \right)$

$A + A^T$

A matrix $A \in \mathbb{S}^n$ (set of square symmetric matrices of dimension n) is called **positive (semi)definite** if for all $x \neq 0 (\text{for all } x) : x^\top A x > (\geq) 0$. We denote this as

$\forall x \in \mathbb{R}^n: \ \langle x, Ax \rangle = x^T A x > (\geq) 0$

$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ $A = A^T$

$A^T = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$

$A \succ (\succeq) 0.$

если $A \in \mathbb{S}^n_{++}$, то все $\lambda(A) > 0$

$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$

The **condition number** of a nonsingular matrix is defined as

$x \in \mathbb{R}^2$ $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ $x^T A x$

$$\kappa(A) = \|A\|\|A^{-1}\|$$

$(x_1\ x_2)\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

$(x_1\ x_2) \cdot \begin{pmatrix} x_1 \cdot 1 + 0 \cdot x_2 \\ 0 \cdot x_1 + 2 \cdot x_2 \end{pmatrix} =$

$= (x_1\ \ x_2)\begin{pmatrix} x_1 \\ 2x_2 \end{pmatrix} =$

$= x_1^2 + 2x_2^2 > 0$

Matrix and vector multiplication

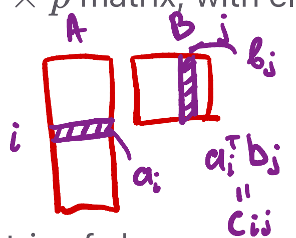
Let A be a matrix of size $m \times n$, and B be a matrix of size $n \times p$, and let the product AB be:

$O(n^3)$ $n^{2.3\ldots}$

кв. матрицы (n)

$$\underset{m \times p}{C} = \underset{m \times n}{A}\,\underset{n \times p}{B}$$

then C is a $m \times p$ matrix, with element (i, j) given by:



$$\boxed{c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}}$$

$(x_1\ x_2)\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

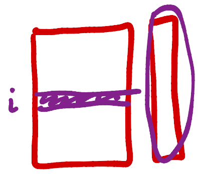
$(x_1\ x_2)\begin{pmatrix} x_1 + x_2 \\ x_1 + x_2 \end{pmatrix} = x_1^2 + x_1x_2 + x_1x_2 + x_2^2 =$

$= x_1^2 + 2x_1x_2 + x_2^2 =$

$= (x_1 + x_2)^2 \geq 0$

Let A be a matrix of shape $m \times n$, and x be $n \times 1$ vector, then the i-th component of the product:

$$\underset{m \times 1}{z} = \underset{m \times n}{A}\,\underset{n \times 1}{x}$$



is given by:

$$z_i = \sum_{k=1}^n a_{ik} x_k$$

Finally, just to remind:

- $\underset{m \times p}{C} = \underset{m \times n}{A}\,\underset{n \times p}{B}$ $\boxed{\underset{p \times m}{C^\top} = \underset{p \times n}{B^\top}\,\underset{n \times m}{A^\top}}$
- $AB \neq BA$ $\cancel{\underset{n \times m}{A^T}\,\underset{p \times n}{B^T}}$
- $e^A = \sum\limits_{k=0}^{\infty} \frac{1}{k!} A^k$
- $e^{A+B} \neq e^A e^B$ (but if A and B are commuting matrices, which means that $AB = BA$, $e^{A+B} = e^A e^B$)
- $\langle \underset{n \times 1}{x}, \underset{n \times m}{A}\,\underset{m \times 1}{y} \rangle = \langle \underset{m \times n}{A^\top}\,\underset{n \times 1}{x}, \underset{m \times 1}{y} \rangle$ $A^{T^T} = A$

$x^T A y$ $(A^T x)^T y =$
$= x^T A y$

Gradient

Let $\boxed{f(x) : \mathbb{R}^n \to \mathbb{R}}$, then vector, which contains all first order partial derivatives:

$$\nabla f(x) = \frac{df}{dx} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

named gradient of $f(x)$. This vector indicates the direction of steepest ascent. Thus, vector $-\nabla f(x)$ means the direction of the steepest descent of the function in the point. Moreover, the gradient vector is always orthogonal to the contour line in the point.

$f=0$ ∪ ∩ x3 (седловые)
f'' ← отличать между собой крит. точки

Hessian

Гессиан

Let $f(x) : \mathbb{R}^n \to \mathbb{R}$, then matrix, containing all the second order partial derivatives:

$$f''(x) = \frac{\partial^2 f}{\partial x_i \partial x_j} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}$$

In fact, Hessian could be a tensor in such a way: $(f(x) : \mathbb{R}^n \to \mathbb{R}^m)$ is just 3d tensor, every slice is just hessian of corresponding scalar function $(H(f_1(x)), H(f_2(x)), \ldots, H(f_m(x)))$.

Jacobian

Якобиан

The extension of the gradient of multidimensional $f(x) : \mathbb{R}^n \to \mathbb{R}^m$ is the following matrix:

$f(x) = 2 \cdot x$

$x \in \mathbb{R}^n$

$f(x) = [a, x]$

$$f'(x) = \frac{df}{dx^T} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

$f(x) = \underset{m \times n}{A} \cdot \underset{n \times 1}{x}$ $\quad \mathbb{R}^n \to \mathbb{R}^m$

Summary

$$
f(x) : X \to Y; \qquad \frac{\partial f(x)}{\partial x} \in G
$$

X	Y	G	Name
$\mathbb{R}$ число	$\mathbb{R}$ число	$\mathbb{R}$ число	$f'(x)$ (derivative)
$\mathbb{R}^n$ вектор	$\mathbb{R}$ число	$\mathbb{R}^n$ вектор	$\dfrac{\partial f}{\partial x_i}$ (gradient)
$\mathbb{R}^n$	$\mathbb{R}^m$	$\mathbb{R}^{m \times n}$	$\dfrac{\partial f_i}{\partial x_j}$ (jacobian)
$\mathbb{R}^{m \times n}$ матрица	$\mathbb{R}$ число	$\mathbb{R}^{m \times n}$	$\left(\dfrac{\partial f}{\partial x_{ij}}\right)_{ij = \overline{1,m};\ \overline{1,n}}$

пример: $f(X) = \det X$
$= \mathrm{rg}\, X$ $\qquad f(X) = \mathrm{tr}\, X$

General concept

Naive approach

The basic idea of naive approach is to reduce matrix/vector derivatives to the well-known scalar derivatives.

Matrix notation of a function

$$
f(x) = c^\top x
$$

Scalar notation of a function

$$
f(x) = \sum_{i=1}^n c_i x_i
$$

Matrix notation of a gradient

$$
\nabla f(x) = c
$$

$$
\frac{\partial f(x)}{\partial x_k} = c_k
$$

$$
\frac{\partial(c_1x_1 + c_2x_2 + \ldots + c_nx_n)}{\partial x_k} = \frac{\partial(c_kx_k)}{\partial x_k} = c_k
$$

Simple derivative

$$
\frac{\partial f(x)}{\partial x_k} = \frac{\partial \left(\sum_{i=1}^n c_i x_i\right)}{\partial x_k}
$$

One of the most important practical tricks here is to separate indices of sum (i) and partial derivatives (k). Ignoring this simple rule tends to produce mistakes.

Differential approach

The guru approach implies formulating a set of simple rules, which allows you to calculate derivatives just like in a scalar case. It might be convenient to use the differential notation here.

$f: \mathbb{R}^n \to \mathbb{R}$

$f = c^T x$

$f(x) = c^T x$

$f(x+dx) = c^T(x+dx)$

$df = f(x+dx) - f(x)$

$df = c^T x + c^T dx - c^T x = c^T dx = \langle c, dx \rangle$

Differentials

After obtaining the differential notation of df we can retrieve the gradient using following formula:

1. Посчитать df
2. Представить в виде $df = \langle \ldots, dx \rangle$
3. $\nabla f = \ldots$

$$
\boxed{df(x) = \langle \nabla f(x), dx \rangle}
$$

Then, if we have differential of the above form and we need to calculate the second derivative of the matrix/vector function, we treat "old" dx as the constant dx_1, then calculate $d(df) = d^2 f(x)$

$$
d^2 f(x) = \langle \nabla^2 f(x) dx_1, dx_2 \rangle = \langle H_f(x) dx_1, dx_2 \rangle
$$

Properties

Let A and B be the constant matrices, while X and Y are the variables (or matrix functions).

- $dA = 0$
- $d(\alpha X) = \alpha(dX)$
- $d(AXB) = A(dX)B$
- $d(X + Y) = dX + dY$
- $d(X^\top) = (dX)^\top$
- $d(XY) = (dX)Y + X(dY)$
- $d\langle X, Y\rangle = \langle dX, Y\rangle + \langle X, dY\rangle$
- $d\left(\dfrac{X}{\phi}\right) = \dfrac{\phi dX - (d\phi)X}{\phi^2}$
- $d\left(\det X\right) = \det X \langle X^{-\top}, dX\rangle$
- $d\left(\text{tr } X\right) = \langle I, dX\rangle$
- $df(g(x)) = \dfrac{df}{dg} \cdot dg(x)$
- $H = (J(\nabla f))^T$

- $d(X^{-1}) = -X^{-1}(dX)X^{-1}$

References

- Convex Optimization book by S. Boyd and L. Vandenberghe - Appendix A. Mathematical background.
- Numerical Optimization by J. Nocedal and S. J. Wright. - Background Material.
- Matrix decompositions Cheat Sheet.
- Good introduction
- The Matrix Cookbook
- MSU seminars (Rus.)
- Online tool for analytic expression of a derivative.
- Determinant derivative

Пример: $f(x) = \ln\langle x, Ax\rangle$

$A \in S^n_{++}$

$\langle x, Ax\rangle > 0$

$x \in \mathbb{R}^n \quad f: \mathbb{R}^n \to \mathbb{R}$

Задача: $df = ? \quad \nabla f = ?$

$(\ln f(x))' = \frac{f'(x)}{f(x)}$

$d\ln f(x) = \frac{df(x)}{f(x)}$

$df = \langle \dots, dx\rangle$

Решение:

1. $df = d\left(\ln\langle x, Ax\rangle\right) = \frac{d(\langle x, Ax\rangle)}{\langle x, Ax\rangle} =$

$$= \frac{\langle dx, Ax\rangle + \langle x, d(Ax)\rangle}{\langle x, Ax\rangle} = \frac{\langle Ax, dx\rangle + \langle x, Adx\rangle}{\langle x, Ax\rangle} =$$

$$= \frac{\langle Ax, dx\rangle + \langle A^T x, dx\rangle}{\langle x, Ax\rangle} = \frac{\langle (A + A^T)x, dx\rangle}{\langle x, Ax\rangle}$$

$$\boxed{\nabla f = \frac{(A + A^T)x}{\langle x, Ax\rangle}}$$

Решить:

$$f(x) = \frac{1}{2}x^T A x - b^T x + c$$

$x \in \mathbb{R}^n; \quad A \in \mathbb{R}^{n \times n}; \quad b \in \mathbb{R}^n; \quad c \in \mathbb{R}$

Найти $df = ? \qquad \nabla f = ?$

$$f = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle + c$$

$$x^T y = \langle x, y\rangle$$

$$\langle \underbrace{db}_{=0}, x\rangle = 0$$

$$d\left(\frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle + c\right) =$$

$$= \frac{1}{2}\, d\left(\langle x, Ax\rangle\right) - d\left(\langle b, x\rangle\right) + \overset{=0}{dc} =$$

$$= \frac{1}{2}\left(\langle dx, Ax\rangle + \langle x, A\,dx\rangle\right) - \langle b, dx\rangle =$$

$$= \frac{1}{2}\left(\langle Ax, dx\rangle + \langle A^T x, dx\rangle\right) - \langle b, dx\rangle =$$

$$= \left\langle \frac{1}{2}(A + A^T)x - b,\ dx\right\rangle$$

$$\Rightarrow \boxed{\nabla f = \frac{1}{2}(A + A^T)x - b}$$

$$f(X) = tr(X) \qquad \nabla f = ? \qquad X \in \mathbb{R}^{n\times n}$$

матрица $\in \mathbb{R}^{n\times n}$

$$f(X) = tr(X) = tr(I^T \cdot X) = \langle I, X\rangle \qquad \boxed{\nabla f = I}$$

$$df = d\left(\langle I, X\rangle\right) = \langle I, dX\rangle$$

$$= \langle \underset{0}{\overset{=0}{dI}}, X\rangle + \langle I, dX\rangle$$

$$f(X) = tr\, X = \sum_{i=1}^{n} x_{ii}$$

$$\frac{\partial f}{\partial x_{kp}} = \frac{\partial \left(\sum_{i=1}^{n} x_{ii}\right)}{\partial x_{kp}}$$

$k = p = i$: 1

иначе: 0

$$\begin{pmatrix} 1 & & 0 \\ & 1 & \\ 0 & & 1 \end{pmatrix}$$

$$\boxed{f(X) = \langle S, X \rangle - \ln \det X}$$

$S = \text{const}$

$S \in \mathbb{R}^{n \times n}$

$X \in \mathbb{R}^{n \times n}$

$d(\det X) = \det X \cdot \langle X^{-T}, dX \rangle$

$\nabla f = ?$

1. $df = ?$

$\det X \neq 0$

$$df = \langle S, dX \rangle - \frac{d(\det X)}{\det X} =$$

$$= \langle S, dX \rangle - \frac{\cancel{\det X} \cdot \langle X^{-T}, dX \rangle}{\cancel{\det X}}$$

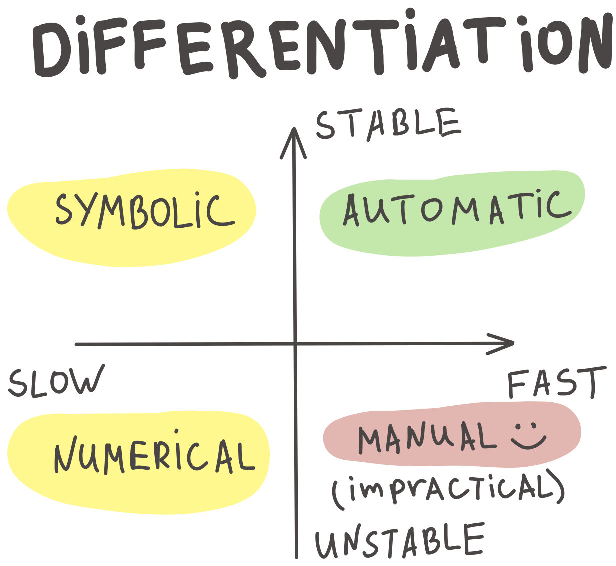
$dS = 0$

$$= \langle S - X^{-T}, dX \rangle$$

$$\boxed{\nabla f = S - X^{-T}}$$

$$X^{-T} = (X^{-1})^{T} = (X^{T})^{-1}$$

Idea



Automatic differentiation is a scheme, that allows you to compute a value of gradient of function with a cost of computing function itself only twice.

Chain rule

We will illustrate some important matrix calculus facts for specific cases

Univariate chain rule

Suppose, we have the following functions $R: \mathbb{R} \to \mathbb{R}, L: \mathbb{R} \to \mathbb{R}$ and $W \in \mathbb{R}$. Then

$$
\frac{\partial R}{\partial W} = \frac{\partial R}{\partial L} \frac{\partial L}{\partial W}
$$

Multivariate chain rule

The simplest example:

$$
\frac{\partial}{\partial t} f(x_1(t), x_2(t)) = \frac{\partial f}{\partial x_1} \frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2} \frac{\partial x_2}{\partial t}
$$

Now, we'll consider $f: \mathbb{R}^n \to \mathbb{R}$:

$$
\frac{\partial}{\partial t} f(x_1(t), \ldots, x_n(t)) = \frac{\partial f}{\partial x_1} \frac{\partial x_1}{\partial t} + \ldots + \frac{\partial f}{\partial x_n} \frac{\partial x_n}{\partial t}
$$

But if we will add another dimension $f: \mathbb{R}^n \to \mathbb{R}^m$, than the j-th output of f will be:

$$\frac{\partial}{\partial t} f_j(x_1(t), \dots, x_n(t)) = \sum_{i=1}^{n} \frac{\partial f_j}{\partial x_i} \frac{\partial x_i}{\partial t} = \sum_{i=1}^{n} J_{ji} \frac{\partial x_i}{\partial t},$$

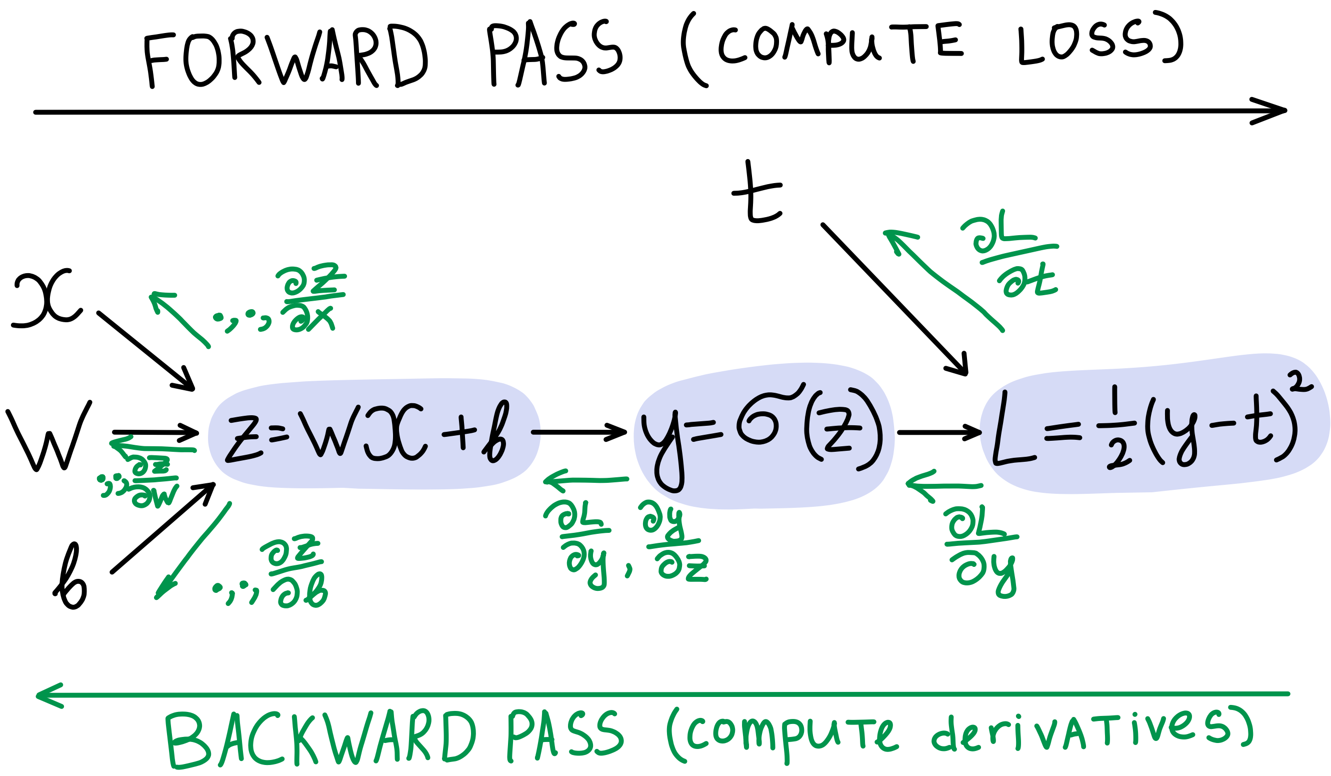
where matrix $J \in \mathbb{R}^{m \times n}$ is the jacobian of the f. Hence, we could write it in a vector way:

$$\frac{\partial f}{\partial t} = J \frac{\partial x}{\partial t} \quad \iff \quad \left(\frac{\partial f}{\partial t}\right)^{\top} = \left(\frac{\partial x}{\partial t}\right)^{\top} J^{\top}$$

Backpropagation

The whole idea came from the applying chain rule to the computation graph of primitive operations

$$L = L\left(y\left(z(w, x, b)\right), t\right)$$



$$\begin{aligned}
z &= wx + b & \frac{\partial z}{\partial w} &= x, \ \frac{\partial z}{\partial x} = w, \ \frac{\partial z}{\partial b} = 0 \\
y &= \sigma(z) & \frac{\partial y}{\partial z} &= \sigma'(z) \\
L &= \frac{1}{2}(y - t)^2 & \frac{\partial L}{\partial y} &= y - t, \ \frac{\partial L}{\partial t} = t - y
\end{aligned}$$

All frameworks for automatic differentiation construct (implicitly or explicitly) computation graph. In deep learning we typically want to compute the derivatives of the loss function L w.r.t. each intermediate parameters in order to tune them via gradient descent. For this purpose it is convenient to use the following notation:

$$\overline{v_i} = \frac{\partial L}{\partial v_i}$$

Let $v_1, \ldots, v_N$ be a topological ordering of the computation graph (i.e. parents come before children). v_N denotes the variable we're trying to compute derivatives of (e.g. loss).

Forward pass:

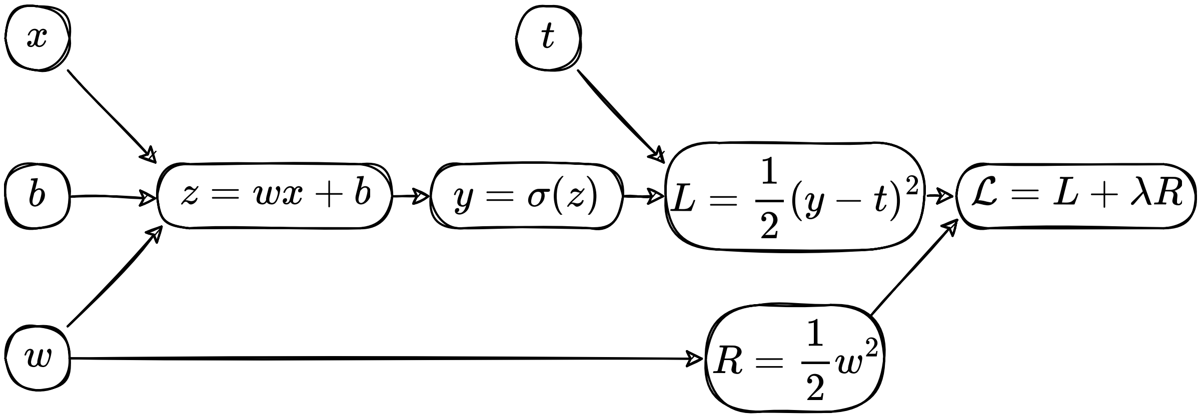
- For $i = 1, \ldots, N$:
  - Compute v_i as a function of its parents.

Backward pass:

- $\overline{v_N} = 1$
- For $i = N - 1, \ldots, 1$:
  - Compute derivatives $\overline{v_i} = \sum_{j \in \mathrm{Children}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$

Note, that $\overline{v_j}$ term is coming from the children of $\overline{v_i}$, while $\frac{\partial v_j}{\partial v_i}$ is already precomputed effectively.

Univariate logistic least squares regression



Forward pass

$$
\begin{aligned}
z &= wx + b \\
y &= \sigma(z) \\
L &= \frac{1}{2}(y-t)^2 \\
R &= \frac{1}{2}w^2 \\
\mathcal{L} &= L + \lambda R
\end{aligned}
$$

Backward pass

$$
\begin{aligned}
\overline{\mathcal{L}} &= 1 \\
\overline{R} &= \overline{\mathcal{L}}\frac{d\mathcal{L}}{dR} = \overline{\mathcal{L}}\lambda \\
\overline{L} &= \overline{\mathcal{L}}\frac{d\mathcal{L}}{dL} = \overline{\mathcal{L}} \\
\overline{y} &= \overline{L}\frac{dL}{dy} = \overline{L}(y-t)
\end{aligned}
$$

$$
\begin{aligned}
\overline{z} &= \overline{y}\frac{dy}{dz} = \overline{y}\sigma'(z) \\
\overline{w} &= \overline{z}\frac{dz}{dw} + \overline{R}\frac{dR}{dw} = \overline{z}x + \overline{R}w \\
\overline{b} &= \overline{z}\frac{dz}{db} = \overline{z} \\
\overline{x} &= \overline{z}\frac{dz}{dx} = \overline{z}w
\end{aligned}
$$

Jacobian vector product

The reason why it works so fast in practice is that the Jacobian of the operations are already developed in effective manner in automatic differentiation frameworks. Typically, we even do not construct or store the full Jacobian, doing matvec directly instead.

Example: element-wise exponent

$$
y = \exp(z) \qquad J = \text{diag}(\exp(z)) \qquad \overline{z} = \overline{y}J
$$

See the examples of Vector-Jacobian Products from autodidact library:

```
defvjp(anp.add,         lambda g, ans, x, y : unbroadcast(x, g),
                        lambda g, ans, x, y : unbroadcast(y, g))
defvjp(anp.multiply,    lambda g, ans, x, y : unbroadcast(x, y * g),
                        lambda g, ans, x, y : unbroadcast(y, x * g))
defvjp(anp.subtract,    lambda g, ans, x, y : unbroadcast(x, g),
                        lambda g, ans, x, y : unbroadcast(y, -g))
defvjp(anp.divide,      lambda g, ans, x, y : unbroadcast(x,   g / y),
                        lambda g, ans, x, y : unbroadcast(y, - g * x / y**2))
```

```
defvjp(anp.true_divide, lambda g, ans, x, y : unbroadcast(x,   g / y),
                        lambda g, ans, x, y : unbroadcast(y, - g * x / y**2))
```

Hessian vector product

Interesting, that the similar idea could be used to compute Hessian-vector products, which is essential for second order optimization or conjugate gradient methods. For a scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$ with continuous second derivatives (so that the Hessian matrix is symmetric), the Hessian at a point $x \in \mathbb{R}^n$ is written as $\partial^2 f(x)$. A Hessian-vector product function is then able to evaluate

$$v \mapsto \partial^2 f(x) \cdot v$$

for any vector $v \in \mathbb{R}^n$.

The trick is not to instantiate the full Hessian matrix: if n is large, perhaps in the millions or billions in the context of neural networks, then that might be impossible to store. Luckily, `grad` (in the jax/autograd/pytorch/tensorflow) already gives us a way to write an efficient Hessian-vector product function. We just have to use the identity

$$\partial^2 f(x) v = \partial [x \mapsto \partial f(x) \cdot v] = \partial g(x),$$

where $g(x) = \partial f(x) \cdot v$ is a new vector-valued function that dots the gradient of f at x with the vector v. Notice that we're only ever differentiating scalar-valued functions of vector-valued arguments, which is exactly where we know `grad` is efficient.

```
import jax.numpy as jnp

def hvp(f, x, v):
    return grad(lambda x: jnp.vdot(grad(f)(x), v))(x)
```

Code

Open in Colab

Materials

- Autodidact - a pedagogical implementation of Autograd
- CSC321 Lecture 6
- CSC321 Lecture 10
- Why you should understand backpropagation :)
- JAX autodiff cookbook

Matrix calculus

1 Find the derivatives of $f(x) = Ax, \quad \nabla_x f(x) = ?, \nabla_A f(x) = ?$

2 Find $\nabla f(x)$, if $f(x) = c^T x$.

3 Find $\nabla f(x)$, if $f(x) = \dfrac{1}{2}x^T Ax + b^T x + c$.

4 Find $\nabla f(x), f''(x)$, if $f(x) = -e^{-x^T x}$.

5 Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \dfrac{1}{2}\|Ax - b\|_2^2$.

6 Find $\nabla f(x)$, if $f(x) = \|x\|_2, x \in \mathbb{R}^p \setminus \{0\}$.

7 Find $\nabla f(x)$, if $f(x) = \|Ax\|_2, x \in \mathbb{R}^p \setminus \{0\}$.

8 Find $\nabla f(x), f''(x)$, if $f(x) = \dfrac{-1}{1 + x^\top x}$.

9 Calculate $df(x)$ and $\nabla f(x)$ for the function $f(x) = \log(x^\top \mathrm{A} x)$.

10 Find $f'(X)$, if $f(X) = \det X$

Note: here under $f'(X)$ assumes first order approximation of $f(X)$ using Taylor series: $f(X + \Delta X) \approx f(X) + \mathbf{tr}(f'(X)^\top \Delta X)$

11 Find $f''(X)$, if $f(X) = \log\det X$

Note: here under $f''(X)$ assumes second order approximation of $f(X)$ using Taylor series: $f(X + \Delta X) \approx f(X) + \mathbf{tr}(f'(X)^\top \Delta X) + \frac{1}{2}\mathbf{tr}(\Delta X^\top f''(X) \Delta X)$

12 Find gradient and hessian of $f : \mathbb{R}^n \to \mathbb{R}$, if:

$$f(x) = \log \sum_{i=1}^{m} \exp(a_i^\top x + b_i), \quad a_1, \ldots, a_m \in \mathbb{R}^n; \quad b_1, \ldots, b_m \in \mathbb{R}$$

13 What is the gradient, Jacobian, Hessian? Is there any connection between those three definitions?

14 Calculate: $\dfrac{\partial}{\partial X} \sum \text{eig}(X), \quad \dfrac{\partial}{\partial X} \prod \text{eig}(X), \quad \dfrac{\partial}{\partial X} \text{tr}(X), \quad \dfrac{\partial}{\partial X} \det(X)$

15 Calculate the Frobenious norm derivative: $\dfrac{\partial}{\partial X} \|X\|_F^2$

16 Calculate the gradient of the softmax regression $\nabla_\theta L$ in binary case ($K = 2$) n - dimensional objects:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

$$L(\theta) = -\left[\sum_{i=1}^{n} (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) + y^{(i)} \log h_\theta(x^{(i)})\right]$$

17 Find $\nabla f(X)$, if $f(X) = \text{tr } AX$

18 Find $\nabla f(X)$, if $f(X) = \langle S, X\rangle - \log \det X$

19 Find $\nabla f(X)$, if $f(X) = \ln \langle Ax, x\rangle, A \in \mathbb{S}^n_{++}$

20 Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if

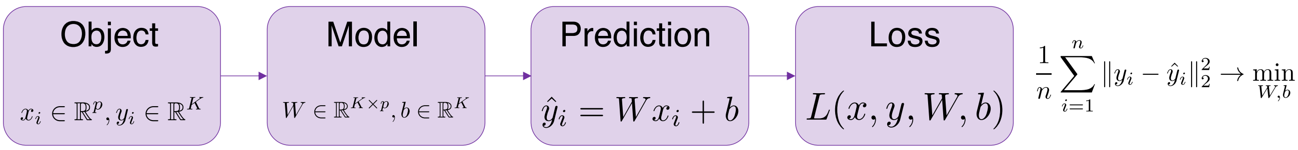
$$f(x) = \ln\left(1 + \exp\langle a, x\rangle\right)$$

21 Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \frac{1}{3}\|x\|_2^3$

22 Calculate $\nabla f(X)$, if $f(X) = \|AX - B\|_F, X \in \mathbb{R}^{k \times n}, A \in \mathbb{R}^{m \times k}, B \in \mathbb{R}^{m \times n}$

23 Calculate the derivatives of the loss function with respect to parameters $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$ for the single object x_i (or, $n = 1$)

Learning



24 Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \langle x, x\rangle^{\langle x, x\rangle}, x \in \mathbb{R}^p \setminus \{0\}$

25 Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \frac{\langle Ax, x\rangle}{\|x\|_2^2}, x \in \mathbb{R}^p \setminus \{0\}, A \in \mathbb{S}^n$

26 Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \frac{1}{2}\|A - xx^\top\|_F^2, A \in \mathbb{S}^n$

27 Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \|xx^\top\|_2$

28 Find the gradient $\nabla f(x)$ and hessian $f''(x)$, if $f(x) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(a_i^\top x)\right) + \frac{\mu}{2}\|x\|_2^2,\ a_i \in \mathbb{R}^n,\ \mu > 0.$

29 Match functions with their gradients:

- [ ] $f(\mathrm{X}) = \mathrm{Tr}\mathrm{X}$

- [ ] $f(\mathrm{X}) = \mathrm{Tr}\mathrm{X}^{-1}$

- [ ] $f(\mathrm{X}) = \det \mathrm{X}$

- [ ] $f(\mathrm{X}) = \ln \det \mathrm{X}$

a $\nabla f(\mathrm{X}) = \mathrm{X}^{-1}$
b $\nabla f(\mathrm{X}) = \mathrm{I}$
c $\nabla f(\mathrm{X}) = \det(\mathrm{X}) \cdot (\mathrm{X}^{-1})^\top$
d $\nabla f(\mathrm{X}) = -\left(\mathrm{X}^{-2}\right)^\top$

30 Calculate the first and the second derivative of the following function $f : S \to \mathbb{R}$
$f(t) = \det(A - tI_n)$, where $A \in \mathbb{R}^{n \times n}$, $S := \{t \in \mathbb{R} : \det(A - tI_n) \neq 0\}$.

31 Find the gradient $\nabla f(x)$, if $f(x) = \mathrm{tr}\left(AX^2BX^{-\top}\right)$.

Automatic differentiation

1. Calculate the gradient of a Taylor series of a $\cos(x)$ using `autograd` library:

```
import autograd.numpy as np # Thinly-wrapped version of Numpy
from autograd import grad

def taylor_cosine(x): # Taylor approximation to cosine function
  # Your np code here
  return ans
```

2. In the following code for the gradient descent for linear regression change the manual gradient computation to the PyTorch/jax autograd way. Compare those two approaches in time.

   In order to do this, set the tolerance rate for the function value $\varepsilon = 10^{-9}$. Compare the total time required to achieve the specified value of the function for analytical and automatic differentiation. Perform measurements for different values of n from `np.logspace(1,4)`.

   For each n value carry out at least 3 runs.

```
import numpy as np

# Compute every step manually

# Linear regression
# f = w * x

# here : f = 2 * x
X = np.array([1, 2, 3, 4], dtype=np.float32)
Y = np.array([2, 4, 6, 8], dtype=np.float32)

w = 0.0

# model output
def forward(x):
    return w * x
```

```python
# loss = MSE
def loss(y, y_pred):
    return ((y_pred - y)**2).mean()

# J = MSE = 1/N * (w*x - y)**2
# dJ/dw = 1/N * 2x(w*x - y)
def gradient(x, y, y_pred):
    return np.dot(2*x, y_pred - y).mean()

print(f'Prediction before training: f(5) = {forward(5):.3f}')

# Training
learning_rate = 0.01
n_iters = 20

for epoch in range(n_iters):
    # predict = forward pass
    y_pred = forward(X)

    # loss
    l = loss(Y, y_pred)

    # calculate gradients
    dw = gradient(X, Y, y_pred)

    # update weights
    w -= learning_rate * dw

    if epoch % 2 == 0:
        print(f'epoch {epoch+1}: w = {w:.3f}, loss = {l:.8f}')

print(f'Prediction after training: f(5) = {forward(5):.3f}')
```

3. Calculate the 4th derivative of hyperbolic tangent function using `Jax` autograd.
4. Compare analytic and autograd (with any framework) approach for the hessian of:

$$f(x) = \frac{1}{2}x^T A x + b^T x + c$$

5. Compare analytic and autograd (with any framework) approach for the gradient of:

$$f(X) = tr(AXB)$$

6. Compare analytic and autograd (with any framework) approach for the gradient and hessian of:

$$f(x) = \frac{1}{2}\|Ax - b\|_2^2$$

7. Compare analytic and autograd (with any framework) approach for the gradient and hessian of:

$$f(x) = \ln\left(1 + \exp\langle a, x\rangle\right)$$

Materials

- HIPS autograd
- PyTorch autograd
- Jax Autodiff cookbook