

Análisis Matemático para Inteligencia Artificial

Verónica Pastor (vpastor@fi.uba.ar),
Martín Errázquin (merrazquin@fi.uba.ar)

Especialización en Inteligencia Artificial

Clase 5

Análisis Matemático - Optimización

Repaso

- 1 En los videos de repaso definimos funciones de cuyo dominio y codominio eran los reales, la gráfica de la función se representa en \mathbb{R}^2 .
- 2 Toda función f describe el cambio de una magnitud (v. dependiente) en términos de otra (v. independiente), cuando esta variable se mueve en cierto intervalo $[x_0, x_0 + h]$ la variación total se mide como $f(x_0 + h) - f(x_0)$.
- 3 Mientras que la variación media es $\frac{f(x_0 + h) - f(x_0)}{(x_0 + h) - x_0}$. Geométricamente, podemos ver la variación media como la pendiente de la recta secante.
- 4 Cuando hacemos que $h \rightarrow 0$, ...

$$(x_1, y_1) ; (x_2, y_2) \quad \frac{y_2 - y_1}{x_2 - x_1}$$



... esto nos conduce a la definición de derivada de f en x_0 :

$$\lim_{h \rightarrow 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Clasificación de funciones

Dada $f : D \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$.

- Si $m = 1$ diremos que es una función
 - **escalar**, si $n = 1$,
 - **campo escalar**, $n > 1$.
- Si $m > 1$ diremos que es una función
 - **vectorial**, si $n = 1$,
 - **campo vectorial**, $n > 1$.

m		$= 1$	> 1
n	$= 1$	función escalar	función vectorial
	> 1	campo escalar	campo vectorial

Conjuntos de Nivel Dada $f : D \subset \mathbb{R}^n \rightarrow \mathbb{R}$ el conjunto de nivel k de $f(x)$ $L_k \subset \mathbb{R}^n$, definido por:

$$L_k = \{x \in \mathbb{R}^n / x \in D \wedge f(x) = k\}$$



La representación geométrica de L_k se obtiene identificando gráficamente los puntos del dominio de la función para los cuales el valor de f es igual a k , para graficar no es necesario agregar un eje.

Derivando campos ...

- escalares: Sea $f : D \subset \mathbb{R}^n \rightarrow \mathbb{R}$, $(x_1, \dots, x_n)^T \mapsto f((x_1, \dots, x_n)^T)$, se definen las **derivadas parciales** como:



$$\frac{\partial f}{\partial x_1} = \lim_{h \rightarrow 0} \frac{f(x_1 + h, x_2, \dots, x_n) - f(x_1, x_2, \dots, x_n)}{h}$$

$$\frac{\partial f}{\partial x_n} = \lim_{h \rightarrow 0} \frac{f(x_1, x_2, \dots, x_n + h) - f(x_1, x_2, \dots, x_n)}{h}$$

Se define el **gradiente** como: $\nabla f = \left(\frac{\partial f}{\partial x_1} \dots \frac{\partial f}{\partial x_n} \right)$. $\mathbb{R}^n \rightarrow \mathbb{R}^n$

Importante: El gradiente apunta en la dirección de máximo crecimiento.

- vectoriales: Sea $f : D \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$, $(x_1, \dots, x_n)^T \mapsto (f_1((x_1, \dots, x_n)^T), \dots, f_m((x_1, \dots, x_n)^T))$, se define el **jacobiano** como:



$$J_{f(i,j)} = \frac{\partial f_i}{\partial x_j}$$

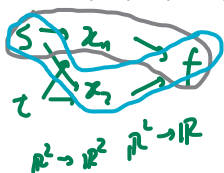
$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$= \begin{pmatrix} -\nabla f_1 - \\ \vdots \\ -\nabla f_m - \end{pmatrix}$$

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

Regla de la Cadena en forma matricial

Sea $f(x_1(s, t), x_2(s, t)) : \mathbb{R}^2 \rightarrow \mathbb{R}$



$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x_1} \frac{\partial x_1}{\partial s} + \frac{\partial f}{\partial x_2} \frac{\partial x_2}{\partial s}$$

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

Y luego

$\nabla f(x_1, x_2)$ $J_{x_1, x_2}(s, t)$

$$\nabla f(s, t) = \frac{df}{d(s, t)} = \frac{df}{dx} \frac{dx}{d(s, t)} = \underbrace{\left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \right]}_{\mathbb{R}^{1 \times 2}} \underbrace{\begin{bmatrix} \frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t} \end{bmatrix}}_{\mathbb{R}^{2 \times 2}} \in \mathbb{R}^{1 \times 2}$$

Recordemos reglas de derivación:

- $\frac{\partial (f+g)(s)}{\partial s} = \frac{\partial f}{\partial s} + \frac{\partial g}{\partial s}$
- $\frac{\partial (fg)(s)}{\partial s} = \frac{\partial f}{\partial s} g(s) + f(s) \frac{\partial g}{\partial s}$

Derivada de matrices

$$f: \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^{a+b}$$

$$\frac{df}{dx} \in \mathbb{R}^{a+b \times n \times m}$$

$$A \in \mathbb{R}^{4 \times 2} \quad x \in \mathbb{R}^3$$


$$f(x) = y$$

$f: \mathbb{R}^3 \rightarrow \mathbb{R}^{4 \times 2}$

desorden el in

$$A \in \mathbb{R}^{4 \times 2} \quad x \in \mathbb{R}^3$$


flatten del out

(4×2)

8×3

Partial derivatives:

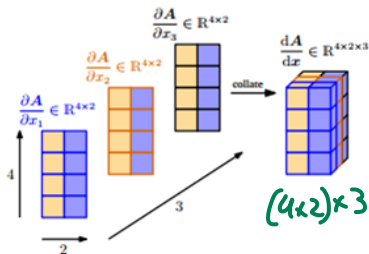
Partial derivatives:

$\frac{\partial A}{\partial x_1} \in \mathbb{R}^{4 \times 2}$, $\frac{\partial A}{\partial x_2} \in \mathbb{R}^{4 \times 2}$, $\frac{\partial A}{\partial x_3} \in \mathbb{R}^{4 \times 2}$

collate

$\frac{dA}{dx} \in \mathbb{R}^{4 \times 2 \times 3}$

$(4 \times 2) \times 3$



(a) Approach 1: We compute the partial derivative $\frac{\partial A}{\partial x_1}$, $\frac{\partial A}{\partial x_2}$, $\frac{\partial A}{\partial x_3}$, each of which is a 4×2 matrix, and collate them in a $4 \times 2 \times 3$ tensor.

re-shape

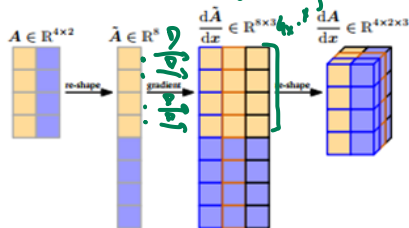
$\tilde{A} \in \mathbb{R}^8$

gradient

$\frac{d\tilde{A}}{dx} \in \mathbb{R}^{8 \times 3}$

re-shape

$\frac{dA}{dx} \in \mathbb{R}^{4 \times 2 \times 3}$



(b) Approach 2: We re-shape (flatten) $A \in \mathbb{R}^{4 \times 2}$ into a vector $\tilde{A} \in \mathbb{R}^8$. Then, we compute the gradient $\frac{d\tilde{A}}{dx} \in \mathbb{R}^{8 \times 3}$. We obtain the gradient tensor by re-shaping this gradient as illustrated above.

escalar \rightarrow rango 0

vector \rightarrow " "

matriz \rightarrow " "

"cubo" \rightarrow " "

tensor

"rango superior"

Diferenciación Automática: ejemplo

Sean $e(x, y) = xy$, $f(x) = 3x$, $g(x) = x^2$, $h(x) = \sin(x)$

$$\frac{\partial e}{\partial x} = y, \frac{\partial e}{\partial y} = x, \frac{\partial f}{\partial x} = 3, \frac{\partial g}{\partial x} = 2x, \frac{\partial h}{\partial x} = \cos(x)$$



$$\frac{\partial s}{\partial x}, \frac{\partial s}{\partial y} \quad (3, 4)$$

$$\frac{\partial s}{\partial e_3} = \frac{\partial e}{\partial x}(e_3, e_4) = e_4 = -0.5966$$

$$\frac{\partial s}{\partial e_4} = \frac{\partial e}{\partial y}(e_3, e_4) = e_3 = 108$$

$$\frac{\partial e_3}{\partial a_1} = \frac{\partial e}{\partial x}(a_1, a_2) = e_2 = 12$$

$$\frac{\partial e_3}{\partial a_1} = \dots = 9$$

$$\frac{\partial e_4}{\partial a_2} = \frac{\partial h}{\partial x}(e_2) = \cos(1) \approx 0.9778$$

$$\frac{\partial e_1}{\partial x} = \frac{\partial g}{\partial x}(x) = 2 \cdot 3 = 6$$

$$\frac{\partial e_1}{\partial y} = \frac{\partial f}{\partial y}(y) = 3$$

$$\frac{\partial s}{\partial a_1} = \frac{\partial s}{\partial e_3} \cdot \frac{\partial e_3}{\partial a_1} = -0.5966 \cdot 12 = -7.1599$$

$$\frac{\partial s}{\partial a_2} = \frac{\partial s}{\partial e_4} \cdot \frac{\partial e_4}{\partial a_2} + \frac{\partial s}{\partial e_3} \cdot \frac{\partial e_3}{\partial a_2} = 108 \cdot 0.9778 + 12 \cdot 0 = 105.6024$$

$$\frac{\partial s}{\partial x} = \frac{\partial s}{\partial a_1} \cdot \frac{\partial a_1}{\partial x} = -7.1599 \cdot 6 = -42.9594$$

$$\frac{\partial s}{\partial y} = \frac{\partial s}{\partial a_2} \cdot \frac{\partial a_2}{\partial y} = 105.6024 \cdot 3 = 316.8072$$

$$x=3$$

$$y=4$$

$$e_1 = g(x)$$

$$e_2 = f(y)$$

$$e_3 = e(e_1, e_2)$$

$$e_4 = h(e_3)$$

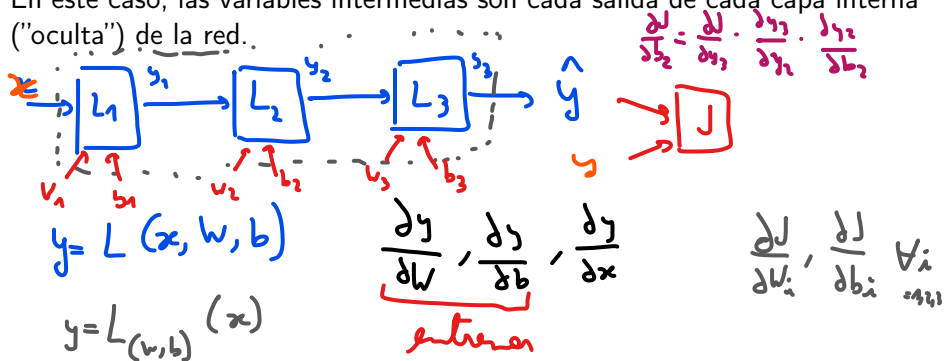
$$s = e(e_3, e_4)$$

Backpropagation

¿Dónde se aplica la diferenciación automática? En **Backpropagation** (o simplemente Backprop), el algoritmo utilizado para entrenar redes neuronales.

¿Qué función cumple? La de computar las derivadas de la función de error/costo respecto de *cada* parámetro de la red neuronal.

En este caso, las variables intermedias son cada salida de cada capa interna ("oculta") de la red.



Redes neuronales (I)

Un perceptrón/neurona es un estimador de la forma:

$$\hat{y} = g(w \cdot x + b)$$

donde en su forma más simple $x, y, w, b \in \mathbb{R}$ y $g : \mathbb{R} \rightarrow \mathbb{R}$ es una función no lineal como puede ser la sigmoidea $\sigma(z) = \frac{1}{1+e^{-z}}$.

Si se define la función $J(W, b)$ de error respecto de los parámetros W y b se puede comprobar que, definiendo $z = w \cdot x + b$ y suponiendo conocido $\frac{dJ}{d\hat{y}} = dY \in \mathbb{R}$:

$$\frac{\partial \hat{y}}{\partial z} = g'(z)$$

$$\frac{\partial J}{\partial W} = \frac{dJ}{d\hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial W} = dY \cdot g'(z) \cdot x$$

$$\frac{\partial J}{\partial b} = \frac{dJ}{d\hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} = dY \cdot g'(z) \cdot 1$$

Redes neuronales (II)

Si ahora consideramos múltiples entradas, es decir $x \in \mathbb{R}^n$, $W \in \mathbb{R}^{1 \times n}$:

$$\hat{y} = g(W \cdot x + b)$$

Entonces ahora para cada elemento de $W = (w_1, \dots, w_n)$ vale lo anterior, y por tanto se puede comprobar que

$$\frac{\partial J}{\partial W} = \nabla_J(W) = (dY \cdot g'(z) \cdot x_1, \dots, dY \cdot g'(z) \cdot x_n) = dY \cdot g'(z) \cdot x^T$$

$$\frac{\partial J}{\partial b} = dY \cdot g'(z)$$

Redes neuronales (III)

Una capa en una red neuronal se define como un vector de k neuronas en paralelo. Una propiedad atractiva de este formato es que se puede considerar a la salida de una capa $y \in \mathbb{R}^k$ como simplemente el x de la capa siguiente. Por convención (y eficiencia computacional) se suele utilizar la misma no-linealidad g para todas las neuronas de la capa.

Nuevamente tenemos:

$$\hat{y} = g(W \cdot x + b)$$

donde $x \in \mathbb{R}^n$, $W \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^k$ y se conviene $g(z) = \begin{pmatrix} g(z_1) \\ \vdots \\ g(z_k) \end{pmatrix}$

¿Y ahora cómo se calculan las derivadas para W y b ?

Redes neuronales (IV)

En el caso de b es simple:

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} = \begin{pmatrix} dY_1 \\ \vdots \\ dY_k \end{pmatrix} \odot \begin{pmatrix} g'(z_1) \\ \vdots \\ g'(z_k) \end{pmatrix} \odot \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = dY \odot g'(z)$$

Ahora para cada elemento de W tenemos:

$$\begin{aligned} \frac{\partial J}{\partial W} &= \begin{pmatrix} \frac{\partial J}{\partial W_{1,1}} & \cdots & \frac{\partial J}{\partial W_{1,n}} \\ \frac{\partial J}{\partial W_{2,1}} & \cdots & \frac{\partial J}{\partial W_{2,n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial W_{k,1}} & \cdots & \frac{\partial J}{\partial W_{k,n}} \end{pmatrix} = \begin{pmatrix} \nabla_J(W_{1,:}) \\ \vdots \\ \nabla_J(W_{k,:}) \end{pmatrix} = \begin{pmatrix} dY_1 \cdot g'(z_1) \cdot x^T \\ \vdots \\ dY_k \cdot g'(z_k) \cdot x^T \end{pmatrix} = \\ &= \begin{pmatrix} dY_1 \\ \vdots \\ dY_k \end{pmatrix} \odot \begin{pmatrix} g'(z_1) \\ \vdots \\ g'(z_k) \end{pmatrix} \cdot x^T = dY \odot g'(z) \cdot x^T \end{aligned}$$

Redes neuronales (V): Cerrando el ciclo

¿Cómo se encadena esto? Nosotros estamos dando por conocida la derivada del error respecto de la salida de la capa, $dY = \frac{dJ}{d\hat{y}}$, pero en realidad no tenemos idea si estamos en una capa intermedia o no.

$$\begin{aligned}\frac{\partial \hat{y}}{\partial x_i} &= \sum_{j=1}^k dY_j \cdot g'(z_j) \cdot W_{j,i} = \left\langle \begin{pmatrix} dY_1 \cdot g'(z_1) \\ \vdots \\ dY_k \cdot g'(z_k) \end{pmatrix}, \begin{pmatrix} W_{1,i} \\ \vdots \\ W_{k,i} \end{pmatrix} \right\rangle = \\ &= \langle dY \odot g'(z), W_{:,i} \rangle = W_{i,:}^T \cdot dY \odot g'(z)\end{aligned}$$

En forma vectorizada:

$$dX = \frac{\partial J}{\partial x} = \begin{pmatrix} \frac{\partial J}{\partial x_1} \\ \vdots \\ \frac{\partial J}{\partial x_n} \end{pmatrix} = \begin{pmatrix} W_{1,:}^T \cdot dY \odot g'(z) \\ \vdots \\ W_{n,:}^T \cdot dY \odot g'(z) \end{pmatrix} = W^T \cdot dY \odot g'(z)$$

Y ese dX no es otra cosa que el dY de la capa anterior!

Redes neuronales (VI): Bosquejo de código

```
class Layer:
    ...
    def forward(self, X):
        self.last_x = X
        self.last_z = self.W @ X + self.b
        self.last_y = self.g.f(self.last_z)
        return self.last_y

    def backwards(self, dY):
        db = dY * self.g.df(self.last_z)
        dW = db @ self.last_x.T
        dX = self.W.T @ db

        self.W, self.b = self.optimizer.step(self.W, self.b, dW, db)

        return dX
    ...
```