

Análisis Matemático para Inteligencia Artificial

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

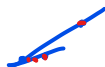
Especialización en Inteligencia Artificial

Clase 5

Análisis Matemático

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 $h \rightarrow 0$, ...



$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{y_2 - y_1}{h}$$

$x_2 = x_1 + h$

... 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

$$(x, y, z) \rightarrow (x \cos(y), y \sin(z))$$

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**, $n > 1$.

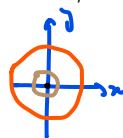
$m \backslash n$	$= 1$	> 1
$= 1$	función escalar	función vectorial
> 1	campo escalar	campo vectorial

$\mathbb{R}^n \rightarrow \mathbb{R}$

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

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

$$f: \mathbb{R}^2 \rightarrow \mathbb{R}$$



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.

conjunto nivel $f: (h, \theta, \phi) \rightarrow h$

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:



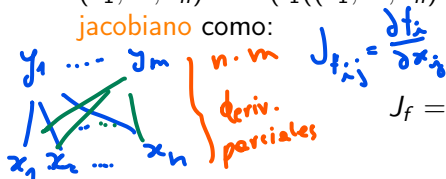
$$\frac{\partial f}{\partial x_1} = \lim_{h \rightarrow 0} \frac{f(x_1 + h, x_2, \dots, x_n) - f(x_1, x_2, \dots, x_n)}{h} \in \mathbb{R}$$

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

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

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:



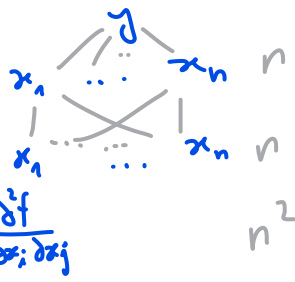
$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} = \begin{pmatrix} -\nabla f_1- \\ \vdots \\ -\nabla f_m- \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Matriz Hessiana

La **matriz Hessiana** es aquella cuyas derivadas de orden 2 de f respecto a $x \in \mathbb{R}^n$ se ubican:

$$f: \mathbb{R}^n \rightarrow \mathbb{R} \leftarrow \begin{matrix} \text{campo} \\ \text{escalar} \end{matrix}$$
$$\nabla f: \mathbb{R}^n \rightarrow \mathbb{R}^n \leftarrow \begin{matrix} \text{campo} \\ \text{vectorial} \end{matrix}$$

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$



A diagram showing a symmetric matrix structure with indices x_1, \dots, x_n on both the top and bottom rows. A handwritten γ is placed above the matrix. To the right, the dimension n^2 is indicated.

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Una aplicación común del Hessiano es el polinomio de Taylor de orden 2 para campos escalares. Sea f un campo escalar $f: \mathbb{R}^n \rightarrow \mathbb{R}$, asumiendo que posee derivadas parciales de todo orden en un entorno de un punto $a \in \mathbb{R}^n$, se define el **polinomio de Taylor** de orden 2:

$$P_2(x) = f(a) + \nabla f(a)^T (x - a) + \frac{1}{2} (x - a)^T H_f(a) (x - a) \in \mathbb{R}$$

Regla de la Cadena en forma matricial

Sea $f(x_1(s, t), x_2(s, t))$

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

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

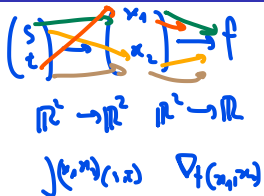
Y luego

$$\frac{df}{d(s, t)} = \frac{df}{dx} \frac{dx}{d(s, t)} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t} \end{bmatrix} \in \mathbb{R}^{1 \times 2}$$

1×2 2×2

Recordemos reglas de derivación:

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

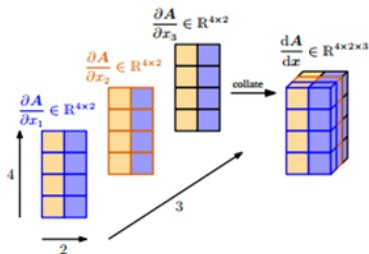


Derivada de matrices

$$f: \mathbb{R}^{a \times b \times c} \rightarrow \mathbb{R}^{m \times n} \Rightarrow \frac{\partial f}{\partial x} \in \mathbb{R}^{(a+b+c \times m \times n)}$$

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


Partial derivatives:

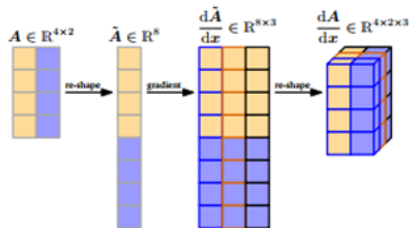


(a) Approach 1: We compute the partial derivative $\frac{\partial A}{\partial x_1}, \frac{\partial A}{\partial x_2}, \frac{\partial A}{\partial x_3}$, each of which is a 4×2 matrix, and collate them in a $4 \times 2 \times 3$ tensor.

plantear $A(x_i)$ $\forall i$, calcular $\frac{\partial A}{\partial x_i}$

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


convertir vector



(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.

1) flatten A
2) gr. matrices
3) recomponer el tensor

Diferenciación Automática



Sean, para una función f :

- x_1, \dots, x_d las variables de entrada
- x_{d+1}, \dots, x_{D-1} las variables intermedias
- x_D la variable de salida
- g_i funciones elementales
- $Hij(x_i)$ el conjunto de nodos hijos de cada x_i

```
class Square:
    def fw(x):
        return x**2
    def bw(x):
        return 2*x
```

Así queda definido un **grafo de cómputo**. Recordando que $f = D$, tenemos que $\frac{\partial f}{\partial x_D} = 1$. Para las otras variables x_i aplicamos la regla de la cadena:

$$\frac{\partial f}{\partial x_i} = \sum_{x_j \in Hij(x_i)} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i} = \sum_{x_j \in Hij(x_i)} \frac{\partial f}{\partial g_j} \frac{\partial g_j}{\partial x_i}$$

- La diferenciación automática se puede utilizar siempre que la función pueda representarse como un grafo de cómputo.
- La gran ganancia de este mecanismo está en que cada función sólo precisa saber cómo derivarse a sí misma, permitiendo OOP.

Diferenciación automática: idea gráfica

$$y = x^{a^b(x)}$$

$$\frac{\partial y}{\partial b} = x^b \log(x)$$

in al fw
no media
evalua b

$$\frac{\partial y}{\partial y} = 1$$

$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial y} \cdot \frac{\partial y}{\partial b} = 1 \cdot 2^{0.9999} \cdot \log(2) = 0.6931$$

$$\frac{\partial y}{\partial a} = \frac{\partial y}{\partial b} \cdot \frac{\partial b}{\partial a} = 0.6931 \cdot 2 = 1.3862$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial a} \cdot \frac{\partial a}{\partial x} + \frac{\partial y}{\partial b} \cdot \frac{\partial b}{\partial x} =$$

$$= 0.6931 \cdot \ln(2) + 1 \cdot 0.9999 \cdot 2^{-0.9999}$$

^ ...

$$\begin{aligned} x &= 2 \\ a &= \ln(x) \\ b &= a^2 \\ y &= x^b \end{aligned}$$



$$x = 2$$

$$a = \ln(2) = 0.6931$$

$$b = 0.6931^2 = 0.4803$$

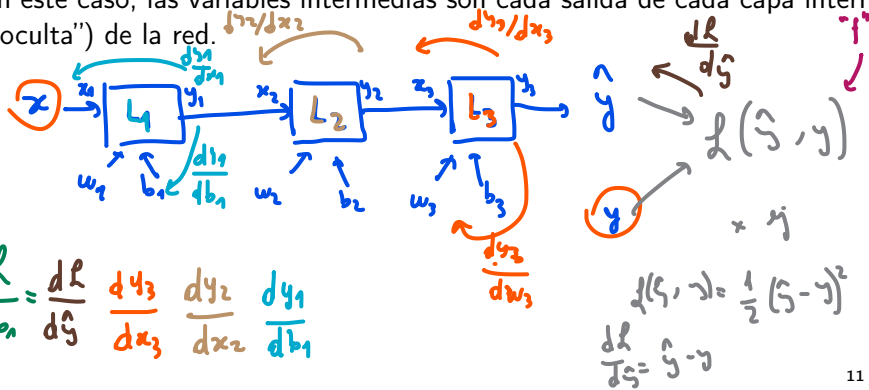
$$y = 2^{0.4803} = 1.3862$$

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 (0): 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)  $\frac{dh}{dy_n}$ 
```

```
    return self.last_y
```

```
def backwards(self, dY):
```

```
    db = dY * self.g.df(self.last_z)
```

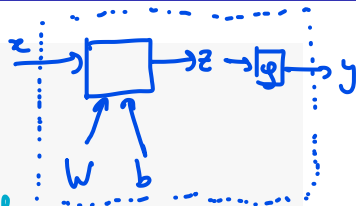
```
    dW = db @ self.last_x.T
```

```
    dX = self.W.T @ db
```

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

```
    return dX
```

```
...
```



$$z = W \cdot x + b$$

$$y = g(z)$$

$$db = dY \odot g'(z)$$

$$dW = db \cdot X^T$$

$$dX = W^T \cdot db$$

para de optim

devuelve el accum-df $\frac{dh}{dy_{n-1}}$

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!