Análisis Matemático para Inteligencia Artificial

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

Especialización en Inteligencia Artificial

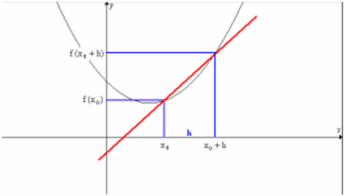
Clase 5

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 $h \to 0$, ...

(x_1, y_1) (x_2, y_2)



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

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

$$\frac{y_2-y_1}{x_2-x_1} = \frac{f(x_2)-f(x_1)}{x_2-x_1} \overset{x_2:=x_1+h}{=} \frac{f(x_1+h)-f(x_1)}{h}$$

$(x,y) \to (x\,\text{sen}(y), x\cos(y))$

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

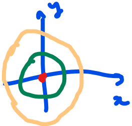
- Si $m = 1$ diremos que es una función
  - escalar, si $n = 1$,
  - campo escalar, $n > 1$.
- Si $m > 1$ diremos que es una función
  - vectorial, si $n = 1$,
  - campo vectorial, $n > 1$.

(in) n \ m (out)	esc (=1)	vec (>1)
esc (=1)	función escalar	función vectorial
vec (>1)	campo escalar	campo vectorial

$g(t) = (5\cos(t), 5\,\text{sen}(t)) \to g'(t) = (5\cos(t)', 5\,\text{sen}(t)')$

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

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



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 $f : D \subset \mathbb{R}^n \to \mathbb{R}$, $(x_1, ..., x_n)^T \mapsto f((x_1, ..., x_n)^T)$, se definen las derivadas parciales como:

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

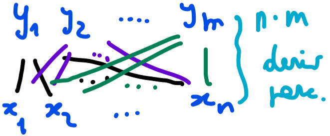
$$\vdots$$

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

Se define el gradiente como: $\nabla f = \left(\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n} \right)$. $\in \mathbb{R}^n$ igual que la entrada de f

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

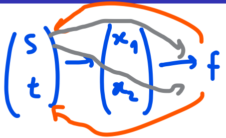
- vectoriales: Sea $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$, $(x_1, ..., x_n)^T \mapsto (f_1((x_1, ..., x_n)^T), ..., f_m((x_1, ..., x_n)^T)$, se define el jacobiano como:



$$J_f = \begin{array}{cc} & x \rightarrow \\ y \downarrow & \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ & \vdots & \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \end{array} = \begin{pmatrix} -\nabla f_1- \\ \vdots \\ -\nabla f_m- \end{pmatrix} \quad J_f(i,j) = \frac{\partial y_i}{\partial x_j}$$

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

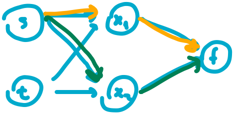
Sea $f(x_1(s,t), x_2(s,t))$



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

$\nabla f(s,t)$

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



Y luego

$\nabla f(x_1,x_2)$ $\quad J_x$

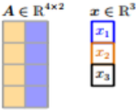
$$\frac{df}{d(s,t)} = \frac{df}{dx}\frac{dx}{d(s,t)} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t}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

$1\times 2 \qquad 2\times 2$

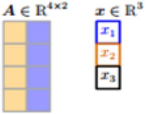
Recordemos reglas de derivación:

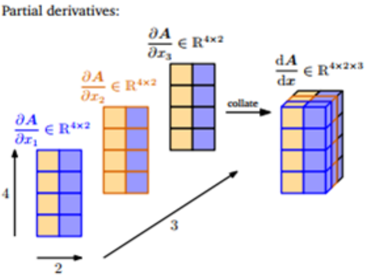
- $\frac{\partial (f+g)(s)}{\partial s} = \frac{\partial f}{\partial s} + \frac{\partial g}{\partial s}$
- $\frac{\partial (fg)(s)}{\partial s} = \frac{\partial f}{\partial s} g(s) + f(s)\frac{\partial g}{\partial s}$

A.shape=(4,2)   x.shape=(3,)



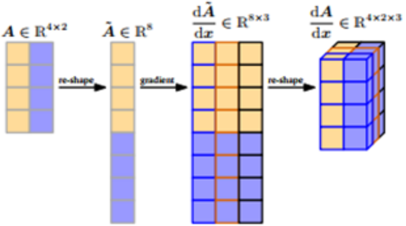
$$\frac{dA}{dx}(i,j,k) = \frac{\partial A_{ij}}{\partial x_k}$$





(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.

- descompongo el in a escalares
- recompongo (apilo)



(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.

- aplano A (flatten)
- recompongo en el orden inverso a aplanar

Diferenciación Automática

Sean, para una función f:

- $x_1, \ldots, x_d$ las variables de entrada
- $x_{d+1}, \ldots, x_{D-1}$ 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

$$\frac{\partial f}{\partial s} = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \cdot \frac{\partial x_i}{\partial s}$$

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:

```
class Square
  def f(x): x**2
def df(x): 2*x
```

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

- 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

$y = x^{\cos^2(x)}$

- $\dfrac{\partial y}{\partial y} = 1$
- $\dfrac{\partial y}{\partial b} = \dfrac{\partial y}{\partial y} \cdot \dfrac{\partial y}{\partial b} =$ $\quad x^b \log(x)$
- $\dfrac{\partial y}{\partial a} = \dfrac{\partial y}{\partial b} \cdot \dfrac{\partial b}{\partial a} =$ $\quad 2a$
- $\boxed{\dfrac{\partial y}{\partial x}} = \overbrace{\dfrac{\partial y}{\partial a} \cdot \dfrac{\partial a}{\partial x}} + \overbrace{\text{“}\dfrac{\partial y}{\partial x}\text{”}}$ $\quad -\operatorname{sen}(x)$

$= \boxed{\text{[illegible]}}$ ✓

$\dfrac{d}{dx} x^b = b \cdot x^{b-1}$

x

$a = \cos(x)$

$b = a^2$

$y = x^b$

$$\begin{tikzcd} x \arrow[r] \arrow[rrr, bend right] & a \arrow[r] & b \arrow[r] & y \end{tikzcd}$$

$x = 2$

$a = \cos(2)$

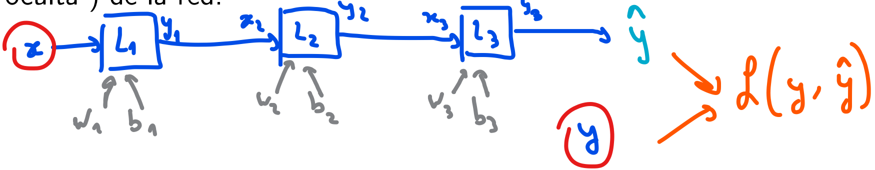
$b =$

$y =$

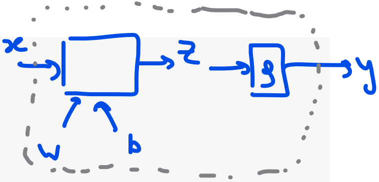
¿Dónde se aplica la diferenciación automática? En Backpropagation (o simplemente Backprop), el algoritmo utilizado para entrenar redes neuronales.

¿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.



$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial y_3}{\partial x_3} \cdot \frac{\partial y_2}{\partial x_2} \cdot \frac{\partial y_1}{\partial w_1}$$




```
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
  ...
```

accum-df

$z = W \cdot x + b$

$y = g(z)$

$$\frac{d\ell}{dy_1} = \frac{d\ell}{dy_2} \cdot \frac{dy_2}{dz_2}$$

dy_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 $g : \mathbb{R} \to \mathbb{R}$ 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 $W = (w_1, \ldots, w_n)$ vale lo anterior, y por tanto se puede comprobar que

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

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

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?

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:

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

$$= \begin{pmatrix} dY_1 \\ \vdots \\ dY_k \end{pmatrix} \odot \begin{pmatrix} g'(z_1) \\ \vdots \\ g'(z_k) \end{pmatrix} \cdot x^T = dY \odot g'(z) \cdot x^T$$

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*.

$$\frac{\partial \hat{y}}{\partial x_i} = \sum_{j=1}^{k} dY_j \cdot g'(z_j) \cdot W_{j,i} = \langle \begin{pmatrix} dY_1 \cdot g'(z_1) \\ \vdots \\ dY_k \cdot g'(z_k) \end{pmatrix}, \begin{pmatrix} W_{1,i} \\ \vdots \\ W_{k,i} \end{pmatrix} \rangle =$$

$$= \langle dY \odot g'(z), W_{:,i} \rangle = W_{i,:}^T \cdot dY \odot g'(z)$$

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!