Análisis Matemático para Inteligencia Artificial

Martín Errázquin (merrazquin@fi.uba.ar)

Especialización en Inteligencia Artificial

Clase 5

Clasificación de funciones

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

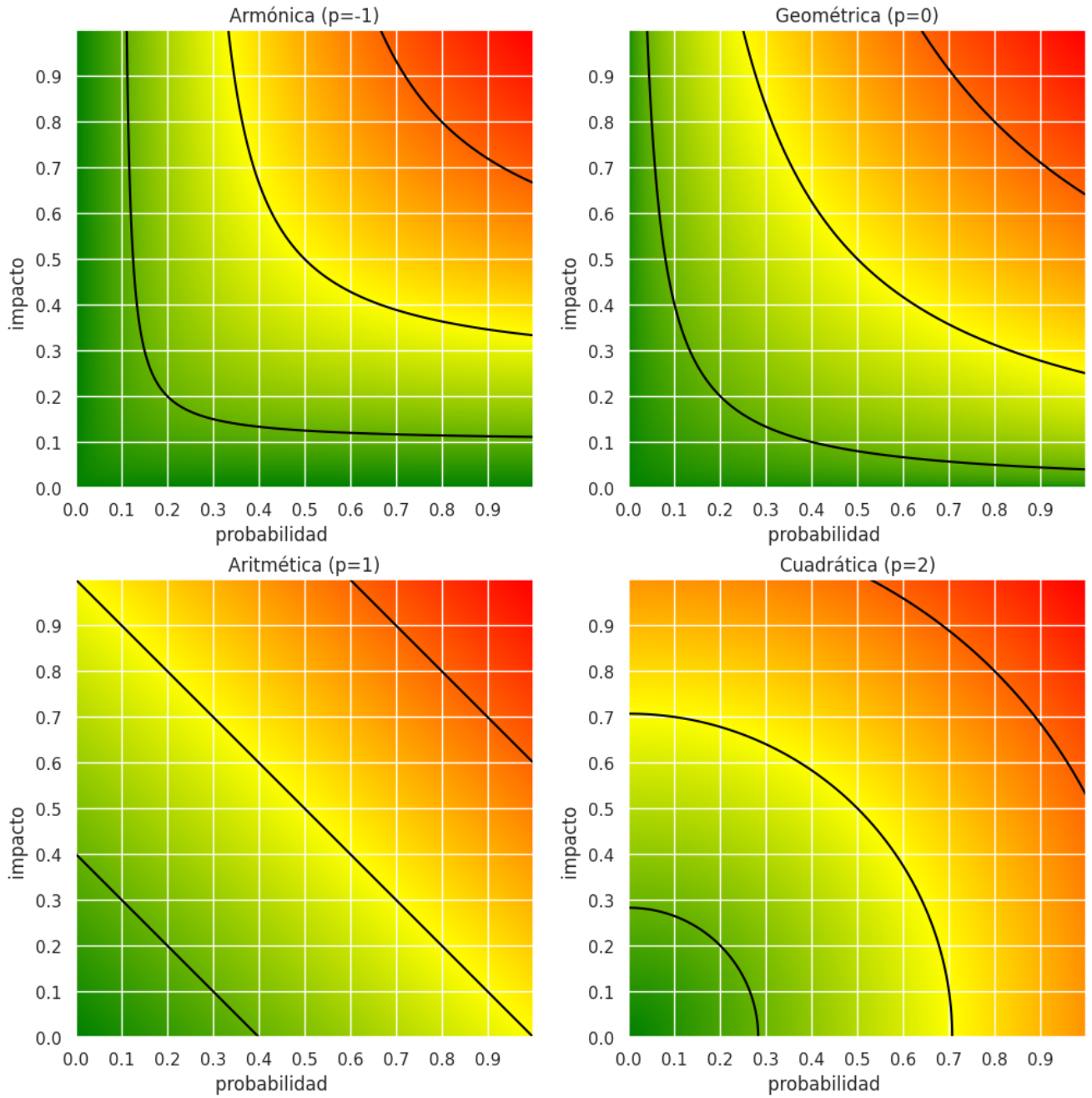
- Si $m = 1$ diremos que es una función
  - escalar, si $n = 1$,
  - campo escalar, $n > 1$.
- Si $m > 1$ diremos que es una función
  - vectorial, si $n = 1$,
  - campo vectorial, $n > 1$.

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:

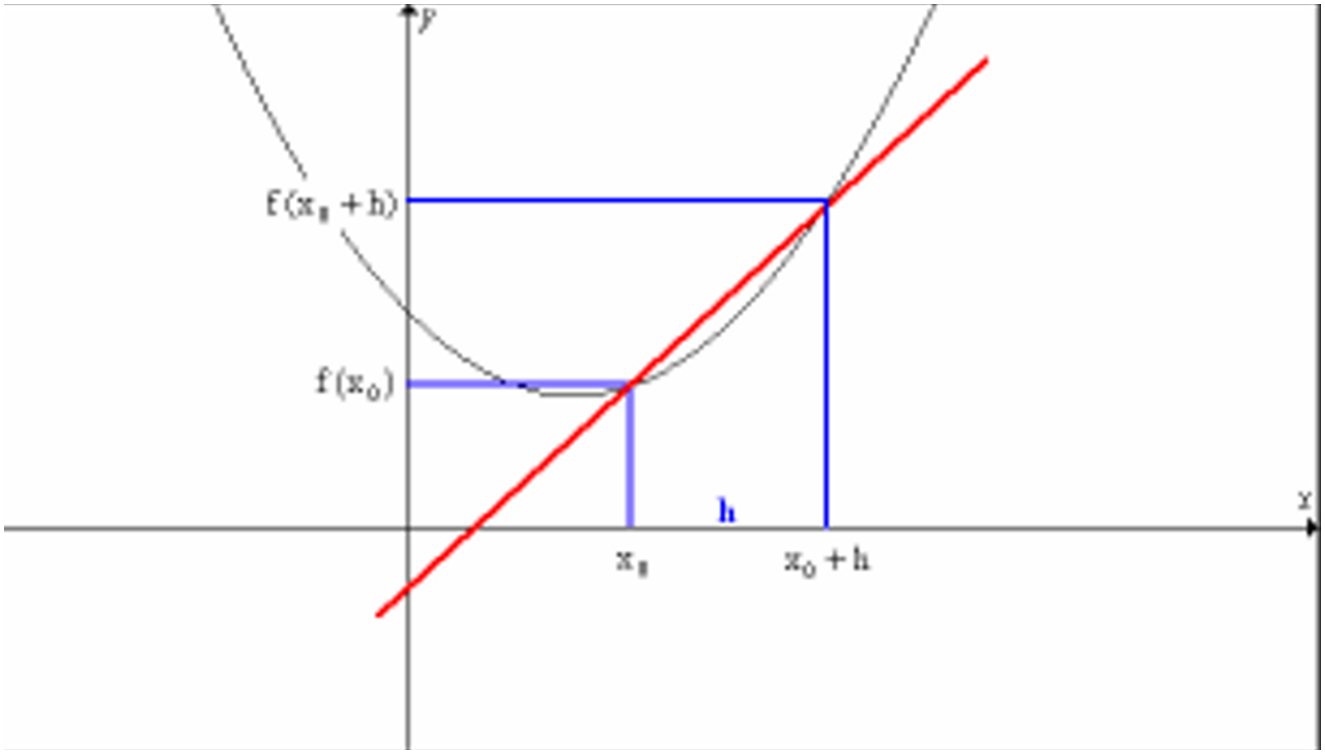
$$L_k = \{x \in \mathbb{R}^n \ / \ x \in D \wedge f(x) = k\}$$

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.

Visualizando los conjuntos de nivel



Repaso: derivando escalares



Recordemos siempre que la derivada busca calcular $\Delta y/\Delta x$ para una función dada $y = f(x)$ en un punto x.

$$f'(x) \doteq \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

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: $\nabla f = \left(\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n} \right)$.

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:

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

Matriz Hessiana y Polinomio de Taylor

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

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

Una aplicación común del Hessiano es el polinomio de Taylor de orden 2 para campos escalares. Sea f un campo escalar $f : \mathbb{R}^n \to \mathbb{R}$, 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:

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

Ejemplo

Sea $f(x_1, x_2) = cos(x_1) + sin(x_2) + 3x_1^2$

Derivadas vectoriales y matriciales

Algunas derivadas útiles de operaciones a nivel de vector/matriz son:

- $\partial X^H = (\partial X)^H$
- $\frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a$
- $\frac{\partial a^T X b}{\partial X} = ab^T$
- $\frac{\partial a^T X^T b}{\partial X} = ba^T$
- $\frac{\partial x^T A x}{\partial x} = (A + A^T)x$
- $\frac{\partial ||x-a||_2}{\partial x} = \frac{x-a}{||x-a||_2}$
- $\frac{\partial}{\partial x}\frac{x-a}{||x-a||_2} = \frac{1}{||x-a||_2} I - \frac{1}{||x-a||_2^3}(x-a)(x-a)^T$
- $\frac{\partial ||x||_2^2}{\partial x} = 2x$
- $\frac{\partial ||X||_F^2}{\partial X} = 2X$

Sea W simétrica:

- $\frac{\partial (x-Ay)^T W (x-Ay)}{\partial x} = 2W(x - Ay)$
- $\frac{\partial (x-Ay)^T W (x-Ay)}{\partial y} = -2A^T W(x - Ay)$
- $\frac{\partial (x-Ay)^T W (x-Ay)}{\partial A} = -2W(x - Ay)y^T$

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

$$\frac{df}{d(s,t)} = \frac{df}{dx}\frac{dx}{d(s,t)} = \begin{bmatrix}\frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2}\end{bmatrix} \begin{bmatrix}\frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t}\end{bmatrix}$$

Nota: Sean $A = f(X) \in \mathbb{R}^{m\times n}, X = g(y) \in \mathbb{R}^{p\times q}, y \in \mathbb{R}$. Si nos interesa $\frac{\partial A}{\partial y} \in \mathbb{R}^{m\times n}$ NO queremos pasar (en forma explícita) por la derivada intermedia $\frac{\partial A}{\partial X} \in \mathbb{R}^{m\times n\times p\times q}$.

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

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:

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

- 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

Colab

Veamos ahora un ejemplo de cómo se implementa un grafo de cómputo con diferenciación automática.

Importante: para fines didácticos está implementado con escalares, pero la lógica es la misma con vectores/matrices, como vimos antes.

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: Resumen

Sea $X \in \mathbb{R}^n, W \in \mathbb{R}^{k \times n}, b \in \mathbb{R}^k$ y $g : \mathbb{R} \to \mathbb{R}$ no lineal, si se recibe $\frac{dJ}{dy} \in \mathbb{R}^k$ entonces:

$$\frac{dJ}{db} = dY \odot g'(z)$$

$$\frac{dJ}{dW} = dY \odot g'(z) \cdot X^T$$

$$\frac{dJ}{dX} = W^T \cdot dY \odot g'(z)$$

donde $z = W \cdot X + b \in \mathbb{R}^k$ e $y = g(z)$ con g aplicada elemento a elemento.
Luego como $y^{(m)} = x^{(m+1)}$, entonces $\frac{dJ}{dX^{(m+1)}} = \frac{dJ}{dy^{(m)}}$ que es lo que le pasamos a la capa anterior.

Redes neuronales: 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
  ...
```

Redes neuronales: Desarrollo

Para no ocupar demasiado tiempo de clase con la derivación del desarrollo que lleva a las fórmulas que vimos antes, dejamos una playlist con videos de youtube en el campus en la sección de esta semana.

¡Aprovecharla!

Recordatorio: Encuesta de Clase

Recordar que está disponible la encuesta de clase! Completarla es cortito y sirve para ir monitoreando el estado del curso.

¿Dónde encontrarla? En la hoja de notas (e.g. "Notas CEIA 10Co2024"), abajo de todo, junto al link de las grabaciones de las clases.