Análisis Matemático para Inteligencia Artificial

Martín Errázquin (merrazquin@fi.uba.ar)

Especialización en Inteligencia Artificial

Clase 6

Temario

Optimización

Optimización en ML

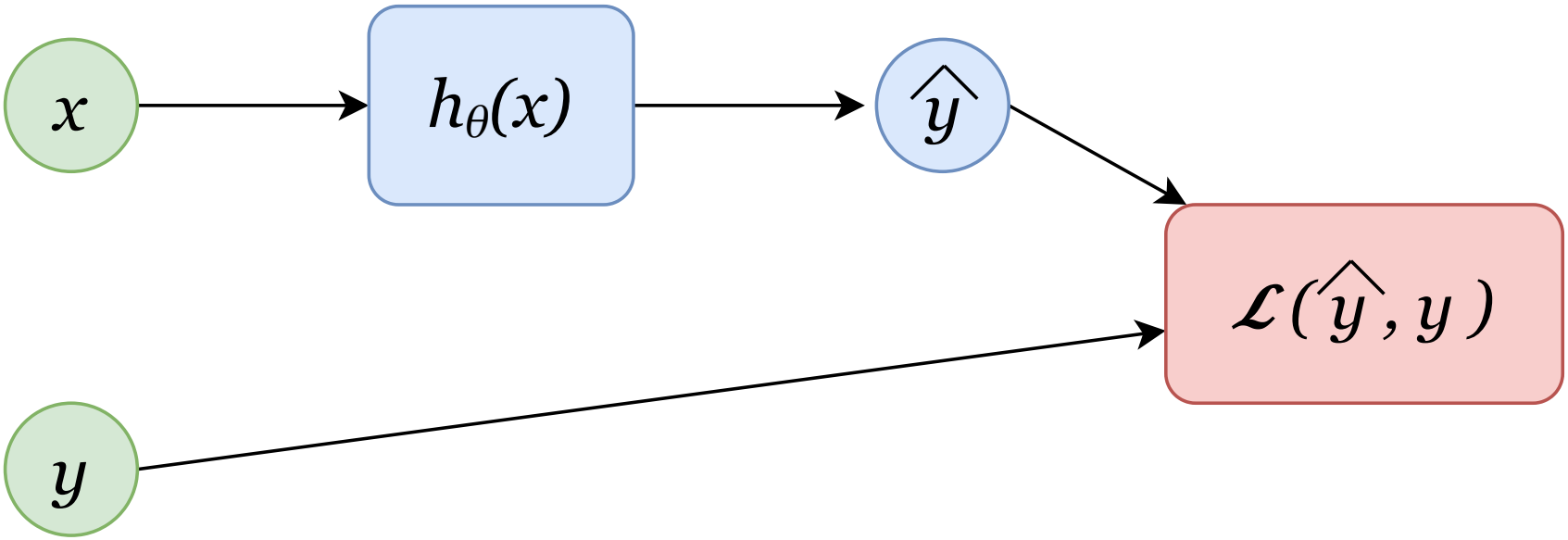
Convención: Todos los casos van a asumirse de minimización, sin pérdida de generalidad ya que maximizar f equivale a minimizar $f' = -f$.

Optimización en general: buscamos minimizar $J(\theta)$, tenemos toda la información necesaria disponible.

Optimización en ML: buscamos minimizar $J(\theta)$, sólo disponemos de un $\hat{J}(\theta)$ basado en el dataset disponible.

Conclusión: **no** son el mismo problema.

Aprendizaje supervisado: esquema



Dada una observación (x, y) fija, entonces la predicción $\hat{y} = h_\theta(x)$ depende puramente de los parámetros θ del modelo, y por lo tanto también la pérdida/error.

Para un dataset $(x_1, y_1), \ldots, (x_n, y_n)$ fijo, definimos entonces una función de costo $J(\theta)$ que sólo depende de los parámetros del modelo, y queremos minimizarla.

Proxy target/surrogate loss

Importante: Definida una función de pérdida por observación $\mathcal{L}(\hat{y}, y)$, la función de costo típicamente se define como

$$J(\theta) = \mathbb{E}[\mathcal{L}(\hat{y}, y)]$$

de donde

$$\hat{J}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\hat{y}_i, y_i)$$

Denominamos *proxy* o *surrogate* a una función f' que queremos minimizar como *medio* para minimizar otra función f que es la que verdaderamente nos interesa.
El esquema entonces resulta:

- *aprendemos* vía train set $\rightarrow$ *necesitamos* minimizar $J_{train}(\theta)$
- *predecimos* vía test set $\rightarrow$ *queremos* minimizar $J_{test}(\theta)$

Ejemplo

Supongamos un caso de clasificación binaria donde definimos la función de pérdida como el *accuracy*, definido como

$$\mathcal{L}(\hat{y}, y) = 1\{\hat{y} \neq y\}$$

Como podemos ver, esta función de pérdida es *muy mala* para minimizar.

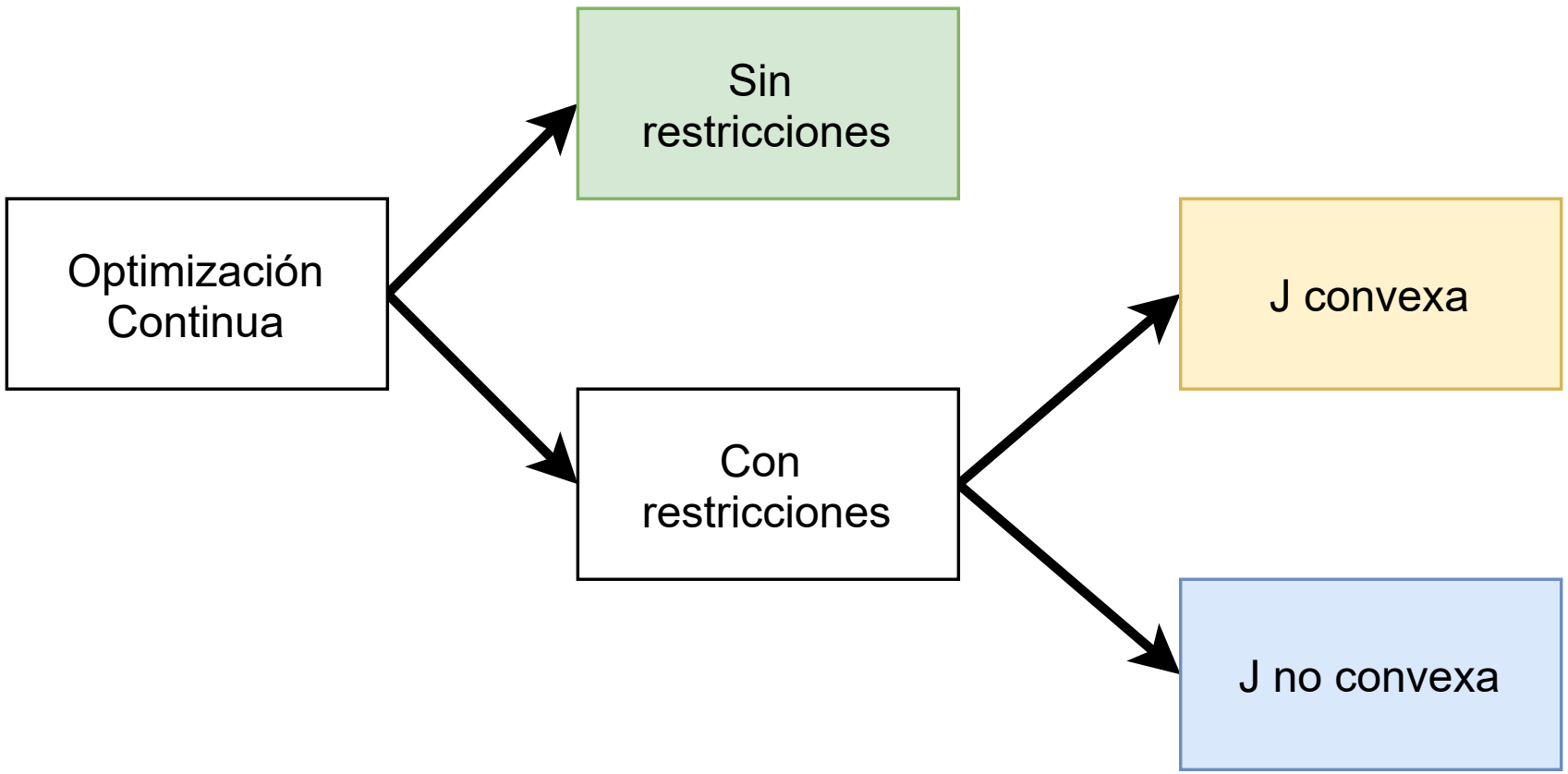
Planteamos entonces entrenar sobre la *cross-entropy loss*

$$\mathcal{L}_{train}(\hat{y}, y) = -y \cdot log(\hat{y}) - (1 - y) \cdot log(1 - \hat{y})$$

que nos permite ya no sólo trabajar con $\hat{y} \in \{0, 1\}$ sino todo el rango continuo $[0, 1]$ de probabilidades, además de, especialmente, ser **derivable respecto de** $\hat{y}$.

Taxonomía

Ahora que nuestro problema es minimizar $J_{train}(\theta)$, podemos separarlo en varios casos:



Spoilers

A grandes rasgos, estos son los temas que vamos a cubrir en las próximas clases:

- Caso Trivial (solución analítica)
- Sin Restricciones:
  - Métodos de primer orden (Gradient-based)
  - Métodos de segundo orden (Hessian-based)
- Con Restricciones:
  - Esquema general (Lagrangiano)
  - Opt. convexa (LP, QP)

Introducción a Optimización

Caso trivial

Analicemos el caso más simple: se conoce la solución analítica.
Ejemplo: modelo lineal con $\hat{y} =< \theta, x >$, matriz de diseño X, vector de targets Y, $\mathcal{L}(\hat{y}, y) = (\hat{y} - y)^2$, entonces el θ óptimo resulta:

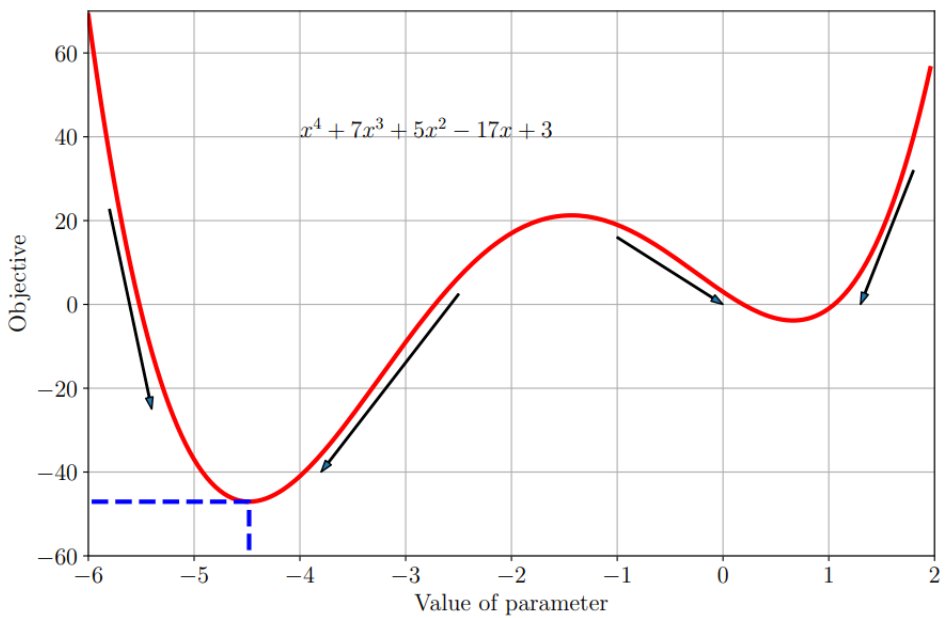
$$\theta^* = \arg\min_{\theta} J(\theta) = (X^T X)^{-1} X^T Y$$

Importante: si ese cálculo nosotros lo realizamos mediante cierto método iterativo en vez de calcularlo directamente es *decisión de implementación* nuestra, la expresión de θ^* ya la tenemos.

Intuición

¿Qué ocurre si no existe solución analítica? En términos generales, la única estrategia posible es *prueba y error* en forma *iterativa*.

Planteemos el caso de $J(\theta)$, $\theta \in \mathbb{R}$.
En cada punto ¿Cómo saber hacia donde moverme?
- Si J es derivable, J' informa la inclinación de J para cada θ.
- Como mínimo, informa la *dirección de crecimiento* y (en sentido contrario) la *dirección de decrecimiento*



Métodos de primer y segundo orden

Los métodos más populares se dividen en dos grandes grupos, aquellos de primer orden (usan gradiente) y de segundo orden (usan gradiente y Hessiano).

Para que un método nos resulte viable debe proveer un resultado *suficientemente bueno* y debe llegar al mismo *suficientemente rápido*.

En forma **muy resumida**, se considera lo siguiente para $\theta \in \mathbb{R}^n$:

- El consumo de memoria (*) de los métodos de primer orden es $\mathcal{O}(n)$ mientras que de segundo orden es $\mathcal{O}(n^2)$
- Todo lo que se quiere usar (∇_f, H_f) se debe estimar, estimar algo más complejo requiere medir más puntos!
- La tasa de convergencia (**) de los métodos de primer orden es $\mathcal{O}(t)$ mientras que de segundo orden es $\mathcal{O}(t^2)$.

(*) Hay formas de hacerlos más eficientes, pero no mucho.
(**) En iteraciones, no en tiempo reloj.

Gradient Descent

Definición

Sea $f : \mathbb{R}^n \to \mathbb{R}$ diferenciable, entonces:

- $\nabla_f(x)^T$ apunta en la dirección de *máximo crecimiento*.
- $-\nabla_f(x)^T$ apunta en la dirección de *máximo decrecimiento*.

Se define entonces el algoritmo de minimización de *descenso por gradiente* (GD) como:

$$x_{t+1} = x_t - \gamma \cdot \nabla_f(x)^T$$

donde $\gamma > 0$ es el *learning rate*, un valor pequeño que controla *cuánto* moverse por paso.

- Para una sucesión γ_t apropiada está demostrado que GD converge a un mínimo *local*.
- Son dos problemas a resolver:
  - Cómo seleccionar el punto inicial x_0
  - Cómo seleccionar γ (o γ_t)

LR decay/"Scheduling"

Idea: al principio está bien aprender de forma agresiva, luego hay que ir *refinando* $\rightarrow$ γ decrece con t.

$$\theta_{t+1} = \theta_t - \gamma_t \cdot g$$

con diferentes opciones de γ_t decreciente, entre ellas:

- polinomial: $\gamma_t = \gamma_0(\frac{1}{t})^k = \gamma_0 \cdot t^{-k}$
- exponencial: $\gamma_t = \gamma_0(\frac{1}{k})^t = \gamma_0 \cdot k^{-t}$
- restringida: $\gamma_t = \begin{cases} (1 - \frac{t}{t_{max}})\gamma_0 + \frac{t}{t_{max}}\gamma_{min} & \text{si } 0 \leq t < t_{max} \\ \gamma_{min} & \text{si } t \geq t_{max} \end{cases}$

con hiperparámetros $k, \gamma_0, \gamma_{min}$ menos sensibles que γ constante.

Detalle de notación: llamamos g al gradiente $\nabla_J(x_t)$ y θ_t al parámetro genérico a optimizar en iteración t.

Estimación de ∇_f

En todos estos casos estamos partiendo de la base que conocemos *perfectamente* $\nabla_f(\theta)$, pero la realidad es que no. En el mejor de los casos, podemos calcular el promedio sobre las n observaciones del dataset.

El problema: ¿cuántas m observaciones utilizamos para estimar $\nabla_f(\theta)$?

Si recordamos que $\sigma_{\bar{x}} \propto \frac{1}{\sqrt{m}}$, reducir *10x* el error estándar de la estimación requiere *100x* más observaciones. $\rightarrow$ no rinde. Al mismo tiempo, hardware tipo GPU/TPU nos permite procesar múltiples entradas en paralelo.

Se definen 3 enfoques generales:

- stochastic (*): $m = 1$
- **minibatch**: $1 < m \ll n$ según hardware
- batch: $m = n$

(*) Hay un conflicto en la literatura, donde a cualquier $m < n$ se le llama *stochastic*, especialmente dada la preponderancia del esquema de minibatch por sobre los demás.

Recap & Colab

Cerrando todo entonces:

1. Para una cantidad m de observaciones realizamos las predicciones
2. En base a esos m puntos se estiman $\nabla J(\theta_t)$ (y potencialmente otros) para cada parámetro relevante θ
3. Utilizando esa información se realiza el cálculo del nuevo valor $\theta_{t+1} = \theta_t - \Delta\theta$ según optimizador elegido
4. (se repite hasta convergencia o criterio de corte)

Ejemplificamos esto con un Colab de Regresión Logística, un modelo de la familiza de GLMs que sirve para problemas de clasificación binaria.

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.