

Análisis Matemático para Inteligencia Artificial

Martín Errázquin (merrazquin@fi.uba.ar)

Especialización en Inteligencia Artificial

Optimización: solución analítica y Gradient Descent

Caso trivial

Analicemos el caso más simple: se conoce la solución analítica.

Ejemplo: modelo lineal con $\hat{y} = \langle \theta, x \rangle$, 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.

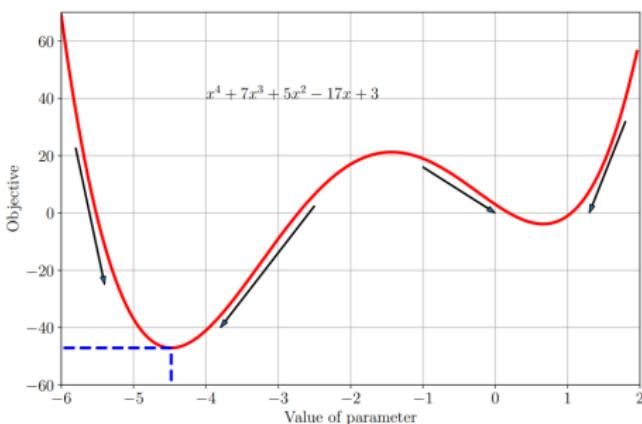
Intuición sobre familia GD

¿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 quiera usar (por ej. ∇_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.

Definición

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

- $\nabla_f(x)$ apunta en la dirección de *máximo crecimiento*.
- $-\nabla_f(x)$ 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* $\rightarrow \gamma$ decrece con t .

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

con diferentes opciones de γ_t decreciente, por ejemplo:

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

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

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

Estimación de ∇_J

En todos estos casos estamos partiendo de la base que conocemos *perfectamente* $\nabla_J(\theta)$, 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 $\nabla_J(\theta)$?

Si recordamos que $\sigma_{\bar{x}} \propto \frac{1}{\sqrt{m}}$, reducir $10x$ el error estándar de la estimación requiere $100x$ más observaciones. → 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/full-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

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)