

# 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  $\theta$  óptimo resulta:

Error cuadrático

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

El tita optimo es el de  
cuadrados minimos

Error cuadrático medio

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

3 cuenta para el  
MÍNIMO GLOBAL

NOTA: el mínimo global solo se logra en el caso trivial.  
Los demás casos con suerte se llega al mínimo local y nunca  
se sabe si estoy en mínimo global o no

# Intuición sobre familia GD

Debería

Proporcional

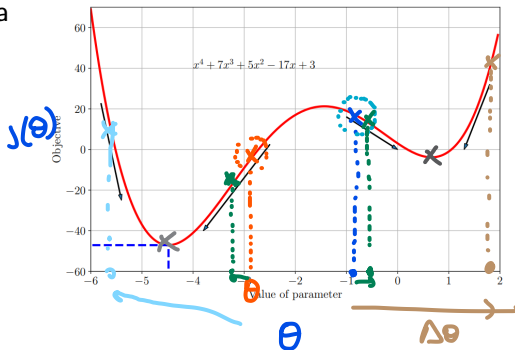
$$\left. \begin{array}{l} \theta_{t+1} = \theta_t + \Delta\theta_t \\ J'(\theta) < 0 \rightarrow \Delta\theta > 0 \\ J'(\theta) > 0 \Rightarrow \Delta\theta < 0 \end{array} \right\} \Delta\theta \propto -J'(\theta)$$

¿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  $\theta$ .
- Como mínimo, informa la *dirección de crecimiento* y (en sentido contrario) la *dirección de decrecimiento*.



Se puede  $J'(\theta)$ !

# 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.  $\nabla_f, H_f$ ) 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. ← aproximaciones

(\*\*) En iteraciones, no en tiempo reloj.

# Definición      Gradiente Descendente

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*.

$\Delta\theta = -\nabla_j(\theta)$  es el  
mejor  $\Delta\theta$   
LOCAL

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

Vanilla GD

$$x_{t+1} = x_t - \underbrace{\gamma}_{\text{learning rate}} \cdot \nabla f(x_t) \quad \Delta\theta = -\gamma \cdot \nabla_j(\theta_t)$$

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

- Para una sucesión  $\gamma_t$  apropiada está demostrado que GD converge a un mínimo *local*.
- Son dos problemas a resolver:
  - Cómo seleccionar el punto inicial  $x_0$   $\rightarrow$  RANDOM
  - Cómo seleccionar  $\gamma$  (o  $\gamma_t$ )  $\rightarrow$ 
    - schedulers
    - extensiones de GD

# LR decay/"Scheduling"

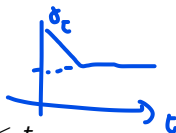
Idea: al principio está bien aprender de forma agresiva, luego hay que ir *refinando*  $\rightarrow \gamma$  decrece con  $t$ .

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

con diferentes opciones de  $\gamma_t$  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  $\gamma$  constante.

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

Si gamma es muy chico, el modelo puede tardar mucho y no converger.

Si gamma es muy grande, puede no aprender.

Entonces es mas conveniente una SUCESION de gamma que uno constante

# Estimación de $\nabla_J$

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  $10\times$  el error estándar de la estimación requiere  $100\times$  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/full-batch:  $m = n$

Handwritten diagram illustrating the relationship between parameters and gradients for different batch sizes. It shows a list of parameters  $\theta_1, \dots, \theta_n$  and their corresponding gradients  $\frac{\partial J}{\partial \theta_1}, \dots, \frac{\partial J}{\partial \theta_n}$ . The diagram is grouped into three sections:  $(\text{Full}) \theta_i$  for the full batch,  $\hat{\theta}_i, \hat{\theta}_j$  for minibatch, and  $\hat{\theta}_i$  for stochastic. A large bracket on the right indicates the overall gradient  $\nabla_J(\theta)$ .

(\*) 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.

Para reducir el ruido (desvío estándar) es conveniente tomar de a minibatch



# Recap

Cerrando todo entonces:  $\theta_t$   $(x_0, y_0) \dots (x_n, y_n)$

- 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  $\theta \rightarrow$  Backprop
- 3 Utilizando esa información se realiza el cálculo del nuevo valor  $\theta_{t+1} = \theta_t + \Delta\theta$  según optimizador elegido  $\theta_{t+1}$  usando GD
- 4 (se repite hasta convergencia o criterio de corte)

✓  
ej.  
100 epoch

$$\theta_t \leftarrow \theta_{t+1}$$

Al tomar  $m$  muestras se procesan  $m$  forward, en paralelo, y luego 1 backward. Finalmente se actualiza el valor de  $\theta$ ; y el ciclo se repite hasta cierta condición.

1 Epoch es una pasada completa sobre todo el dataset