

# Análisis Matemático para Inteligencia Artificial

Martín Errázquin (merrazquin@fi.uba.ar)

Especialización en Inteligencia Artificial

Clase 6

- 1 Optimización en ML
- 2 Opt. sin restricciones
  - Gradient Descent
  - Extensiones
  - Batch size

## Optimización en ML

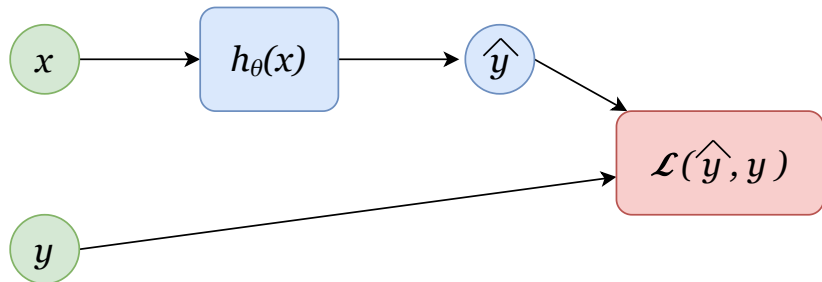
**Convención:** Todos los casos van a asumirse de minimización, sin pérdida de generalidad ya que maximizar  $f$  equivale a minimizar  $f' = -f$ .

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  $\theta$  del modelo, y por lo tanto también la pérdida/error.

Para un dataset  $(x_1, y_1), \dots, (x_n, y_n)$  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  $J_{test}(\theta)$

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

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

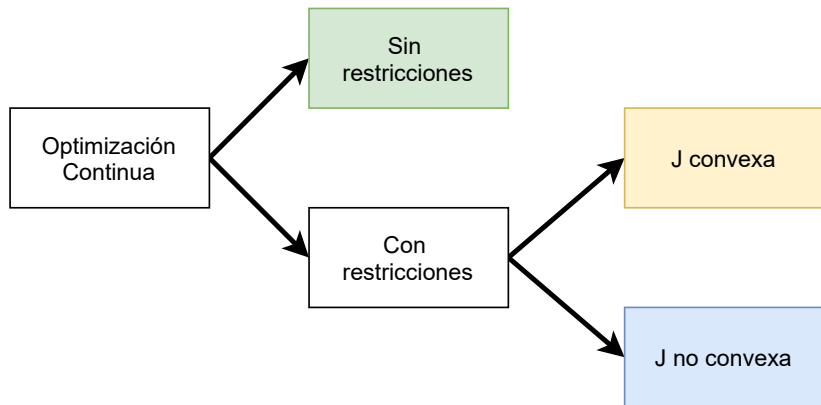
Como podemos ver, esta función de pérdida es *muy mala* para minimizar.

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  $\hat{y}$** .

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





Opt. sin restricciones

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:

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

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

## Gradient Descent

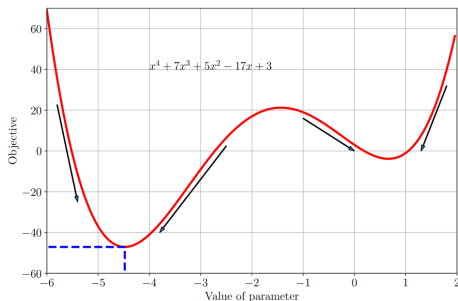
# Intuición

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



# Definición

Sea  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  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  $\gamma_t$  apropiada está demostrado que GD converge a un mínimo *local*.
- Son dos problemas a resolver:
  - Cómo seleccionar el punto inicial  $x_0$
  - Cómo seleccionar  $\gamma$  (o  $\gamma_t$ )

¿Pausa! ¿Por qué Gradient Descent?

- GD pide muy poco, que  $f$  sea diferenciable (y recordemos que nosotros la construimos...)
- GD es una aproximación *lineal*

¿Podemos hacer algo mejor que lineal?

Recordemos el polinomio de Taylor de grado 2 de una función  $f(x)$  alrededor de un punto  $x_t$  evaluada en un punto  $\tilde{x} = x_t + \Delta$  con  $\Delta$  pequeño:

$$f(\tilde{x}) \approx f(x_t) + f'(x_t)(\tilde{x} - x_t) + \frac{1}{2}f''(x_t)(\tilde{x} - x_t)^2$$

$$f(x_t + \Delta) \approx f(x_t) + f'(x_t)\Delta + \frac{1}{2}f''(x_t)\Delta^2$$

# Método de Newton

Para el caso anterior (desarrollo de Taylor de orden 2) el máximo ocurre en  $f'(x_t + \Delta^*) = 0$  para  $\Delta^* = -\frac{f'(x_t)}{f''(x_t)}$ . Luego, el método de Newton plantea:

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$$

O en su versión multivariada:

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

Pros:

- Incorpora curvatura para corrección  $\rightarrow$  mayor velocidad de convergencia

Cons:

- Newton en particular no converge a mínimo, sino a punto crítico: surgen los *saddle points* como peligro.
- La estimación de Hessiano requiere **muchas** observaciones. Goodfellow compara  $10^2$  obs. para  $\nabla_f(x)$  vs  $10^4$  para  $H^{-1} \nabla_f(x)^T$ .

## Extensiones



# EWMA/EMA/Exponential Smoothing

Idea: quiero ir promediando (o algo parecido) pero darle más peso a lo reciente para que lo muy viejo no condicione tanto.

Dada una sucesión de valores a medir  $x_1, \dots, x_t$  el  $\lambda$ -*Exponentially Weighted Moving Average* es:

$$a_t = \lambda \cdot a_{t-1} + (1 - \lambda) \cdot x_t$$

Podemos ver que, aproximadamente,  $a_n$  toma la forma de  $\sum_{i=1}^n \lambda^{n-i} x_i = x_n + \lambda x_{n-1} + \lambda^2 x_{n-2} + \lambda^3 x_{n-3} + \dots$ . Usando  $\lambda \in (0, 1)$  es un promedio ponderado que favorece valores recientes.

Y algo muy importante: es  $\mathcal{O}(1)$  en memoria y  $\mathcal{O}(1)$  para actualizar!

Nota: El valor inicial  $a_0$  es un hiperparámetro, aunque es bastante común usar 0, la media global de  $x$  ( $\bar{x}$ ) o directamente el primer valor  $x_1$ .

# EWMA vs Windowed-Average (SMA)



A. Kofman <http://spotware.ctrader.com/c/XKB5n>, <https://commons.wikimedia.org/w/index.php?curid=27263134>

Queremos seguir utilizando gradient descent (GD/VGD), la idea es proponer adaptaciones del mismo que ataquen los problemas del original, a saber:

- elección de  $\theta_0$
- elección de  $\gamma_t$
- convergencia lenta

Recordemos la expresión de *(Vanilla) Gradient Descent*:

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

con  $g = \nabla_f(\theta_t)$ .

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, entre ellas:

- 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.

# Momentum

Idea: adaptar el  $\gamma$  según consistencia (tener en cuenta steps anteriores)  $\rightarrow$  agregar memoria.

$$\begin{cases} v_t = \alpha v_{t-1} - \gamma \cdot g \\ \theta_{t+1} = \theta_t + v_t \end{cases}$$

- $\alpha \in (0, 1)$  es la *viscosidad* (en términos físicos) o retención de memoria de valores anteriores.

Observar que

$$\theta_{t+1} = \theta_t - \gamma(g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \dots) = \theta_t - \gamma \sum_{i=0}^t \alpha^i g_{t-i}$$

Idea: "reescalar" el gradiente para tener más estabilidad. El reescalamiento se hace a nivel de *feature* para que variaciones grandes sobre un feature no anulen a otros que aún no variaron.

$$\begin{cases} s_t = \lambda s_{t-1} + (1 - \lambda)g^2 \\ \theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{s_t + \epsilon}} \odot g \end{cases}$$

con  $^2$  y  $\sqrt{\phantom{x}}$  aplicados *element-wise*, e.g.  $g^2 = g \odot g = (g_1^2, g_2^2, \dots, g_n^2)$ .

- $\lambda \in (0, 1)$  es la retención de memoria de valores anteriores.
- $0 < \epsilon \ll 1$  es una constante para estabilidad numérica. Valores típicos rondan  $10^{-6}$ .

Idea: Momentum y RMSProp hacen cosas distintas y ambas están buenas  
¡Mezclemos!

$$\begin{cases} v_t = \beta_1 v_{t-1} + (1 - \beta_1)g \\ s_t = \beta_2 s_{t-1} + (1 - \beta_2)g^2 \\ v'_t = \frac{v_t}{1 - \beta_1^t} \\ s'_t = \frac{s_t}{1 - \beta_2^t} \\ \theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{s'_t + \epsilon}} \odot v'_t \end{cases}$$

- $\beta_1, \beta_2 \in (0, 1)$  son la retención de memoria de valores anteriores de media y variabilidad del gradiente. Valores default son  $\beta_1 = 0.99, \beta_2 = 0.999$ .
- $0 < \epsilon \ll 1$  es una constante para estabilidad numérica. Valor default es  $10^{-8}$ .

# De dónde sale el rescaling

Formato de promedio común es como ponderar todo con  $\frac{1}{n}$ :

$$\frac{\sum_{i=1}^n x_i}{n} = \frac{1}{n} \sum_{i=1}^n x_i = \sum_{i=1}^n \frac{1}{n} x_i. \text{ Importante: } \sum_{i=1}^n \frac{1}{n} = 1.$$

¿Qué pasa con el EWMA? Para una sucesión que inicia en  $t = 0$  con  $a_{-1} = 0$ , tenemos  $a_n = \sum_{i=0}^n \lambda^{n-i}(1-\lambda)x_i$ . Veamos que:

$$\lambda^0 + \lambda^1 + \dots + \lambda^n = \sum_{i=0}^n \lambda^i = \frac{1 - \lambda^{n+1}}{1 - \lambda}$$

Entonces,

$$\sum_{i=0}^n \lambda^{n-i}(1-\lambda) = \sum_{i=0}^n \lambda^i(1-\lambda) = \frac{1 - \lambda^{n+1}}{1 - \lambda}(1-\lambda) = 1 - \lambda^{n+1}$$

La suma de los pesos no da 1! Si  $n \gg 0$  entonces  $1 - \lambda^{n+1} \approx 1$ , pero en  $n$  chicos se nota mucho. Por ej.  $\lambda = 0.99, n = 10 \Rightarrow 1 - \lambda^{n+1} \approx 0.0956$ .

La solución? Reescalar dividiendo por  $1 - \lambda^{n+1}$ .



Batch size

## Estimación de $\nabla_f$

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  $10\times$  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:  $m = n$

(\*) 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.