# Análisis Matemático para Inteligencia Artificial

Martín Errázquin (merrazquin@fi.uba.ar)

Especialización en Inteligencia Artificial

Clase 7

# Temario

1. Gradient Descent (cont.)
   - Extensiones
2. Optimización sin restricciones (métodos de segundo orden)
   - Newton y asociados
   - BFGS
3. Comentarios Extra
   - Cuándo usar cuál
   - *State of the Art*

# Gradient Descent (cont.)

# 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 $x_1, \ldots, x_t$ 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 $\sum_{i=1}^{n} \lambda^{n-i} x_i = x_n + \lambda x_{n-1} + \lambda^2 x_{n-2} + \lambda^3 x_{n-3} + \ldots$. 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 ($\overline{x}$) 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

# 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

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

# RMSProp

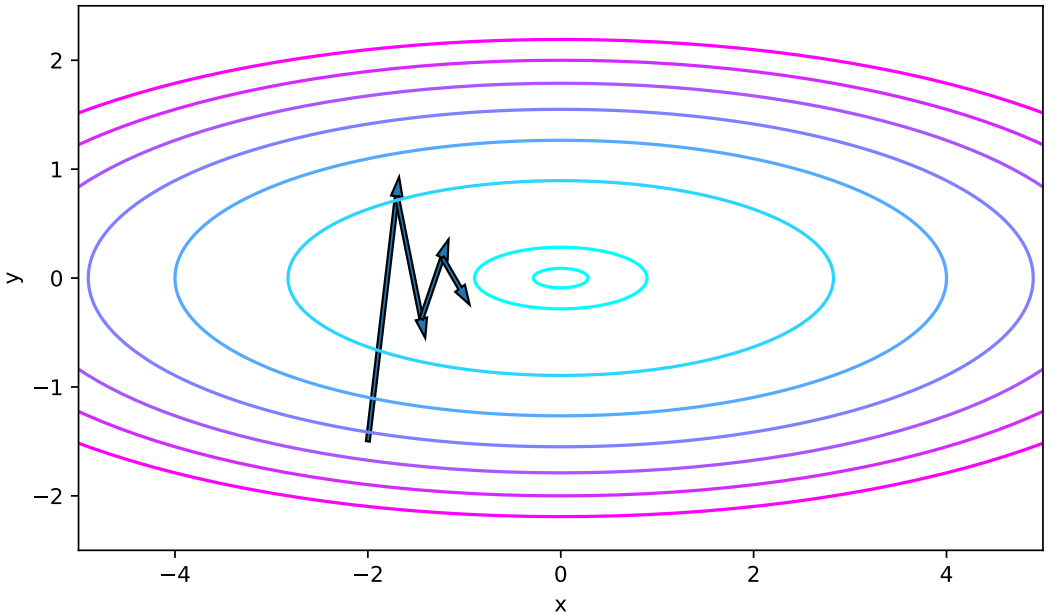
Idea: "reescalar" el gradiente para tener más estabilidad. El reescalamiento se hace a nivel de *feature* para que variaciones grandes sobre un feature no anulen a otros que aún no variaron.

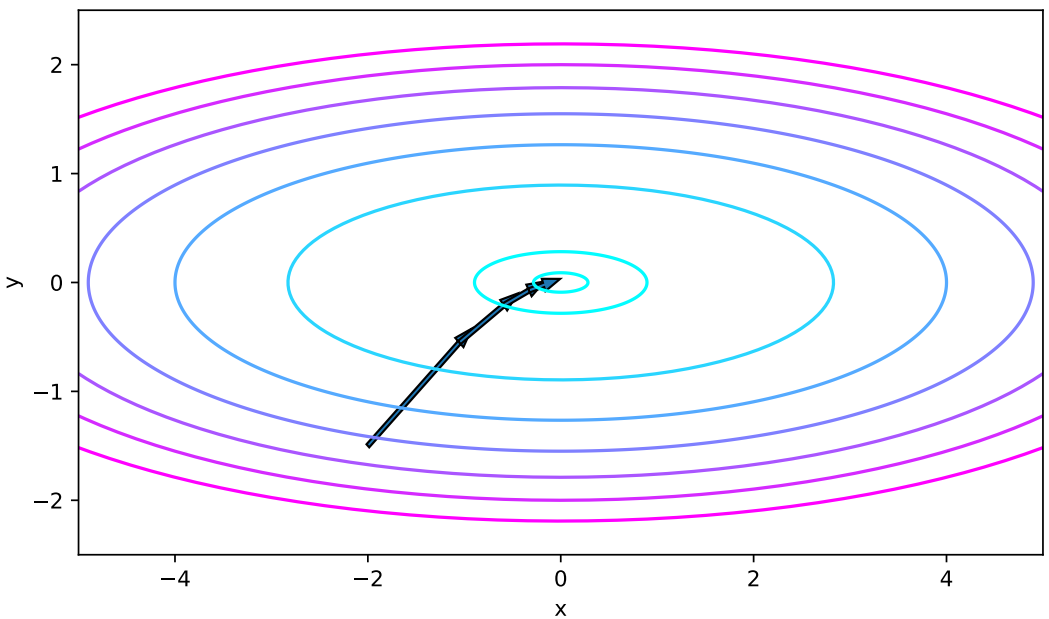
$$\begin{cases} s_t = \lambda s_{t-1} + (1-\lambda) g^2 \\ \theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{s_t + \epsilon}} \odot g \end{cases}$$

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

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

# Visualización GD vs RMSProp



# Adam

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 $\frac{1}{n}$:

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

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

Entonces,

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

La suma de los pesos no da 1! Si $n >> 0$ entonces $1-\lambda^n \approx 1$, pero en $n$ chicos se nota mucho. Por ej. $\lambda = 0.99, n = 10 \Rightarrow 1-\lambda^n \approx 0.0956$.
La solución? Reescalar dividiendo por $1-\lambda^n$.

# Optimización sin restricciones (métodos de segundo orden)

# Método de Newton

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:

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

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

Si derivamos e igualamos a 0, obtenemos el mínimo en $\Delta^* = -\frac{f'(x_t)}{f''(x_t)}$. En versión multivariada, esto es $\Delta^* = H^{-1} \nabla_f(x_t)^T$.
El método de Newton es eso:

$$\theta_{t+1} = \theta_t - H^{-1} \nabla_J(\theta_t)^T$$

Pro: Tiene convergencia local cuadrática.
Con: Es *caro* estimar $H^{-1} \nabla_J(\theta_t)^T$ (Según Goodfellow $10^4$ vs $10^2$ para $\nabla_J(\theta_t)^T$.

# BFGS

Como calcular $H^{-1}\nabla_J(\theta_t)^T$ en cada iteración es muy caro, se plantea aproximar $H$ usando $H_t$ iterable que sea simple. Se pide que $H_t$ sea simétrica y definida positiva, y que además cumpla la *ecuación secante*:

$$H_t y_t = s_t$$

donde $y_t = \Delta\nabla_J(\theta) = \nabla_J(\theta_t) - \nabla_J(\theta_{t-1})$ y $s_t = \Delta\theta = \theta_t - \theta_{t-1}$.

La regla de update resulta:

$$H_{t+1} = V_t^T H_t V_t + \frac{s_t s_t^T}{y_t^T s_t}$$

donde $V_t = I - \frac{s_t y_t^T}{y_t^T s_t}$. Observar que el método, si bien es más eficiente, es sub-cuadrático ("Quasi-Newton") en iteraciones.

Nota: ¿Cómo inicializar $H_0$? No hay fórmula, suele usarse $H_0 = I$. Por ejemplo, Sklearn/SciPy lo hacen.

# L-BFGS

Un problema inherente a BFGS (y cualquier método similar) es que es $\mathcal{O}(n^2)$ en memoria. L-BFGS plantea usar sólo información de las últimas $m$ iteraciones para aproximar $H$, específicamente los pares $(s_k, y_k)$ con $k = t-1, \ldots, t-m$. Luego se preestablece un $H_t^0 = \frac{s_{t-1}^T y_{t-1}}{y_{t-1}^T y_{t-1}} I$ y se aplican los $m$ pasos de BFGS hasta llegar a $H_t$. Si bien la cuenta es engorrosa, es eficiente de computar $H_t \nabla_J(\theta_t)$:

```
q = grad_t
for i in k-1, ..., k-m:
  alpha[i] = s[i].T @ q / (y[i].T @ s[i])
  q -= alpha[i] * y[i]
res = H_k0 @ q
for i in k-m, ..., k-1:
  res += s[i] * (alpha[i] - y[i].T @ r)
```

Luego de obtener el resultado, se elimina del buffer el par $(s, y)$ más viejo y se reemplaza por el último, siendo entonces $\mathcal{O}(mn)$.

# Comentarios Extra

# ¿Cuándo conviene usar qué?

En términos generales, los métodos de 1º orden son mucho más rápidos. Además, si bien existen variantes que lo solventan, en general los métodos de 2º orden requieren entrenar *en batch*.

Sin embargo, hay casos donde esto está bien. Por ejemplo, las librerías de árboles boosteados (GBDT) como LightGBM o XGBoost usan métodos de 2º orden *entre árboles*.

- Para datasets "chicos" y/o modelos tradicionales de pocos parámetros es más que aceptable usar métodos de 2º orden.
- Para datasets o modelos muy grandes resulta prohibitivo entrenar en batch, y suelen premiarse *updates rápidos*.
- Como siempre, el método correcto *depende* de la situación.

# SOTA: AdamW (1/2)

¿Qué pasa si le agregamos regularización $L^2$? Se ve así:

$$
\begin{aligned}
g_t &\leftarrow \nabla_J(\theta_{t-1}) + w\theta_{t-1} \\
v_t &\leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t \\
s_t &\leftarrow \beta_2 s_{t-1} + (1-\beta_2) g_t^2 \\
v_t' &\leftarrow v_t / (1-\beta_1^t) \\
s_t' &\leftarrow s_t / (1-\beta_2^t) \\
\theta_t &\leftarrow \theta_{t-1} - \frac{\gamma}{\sqrt{s_t'} + \epsilon} \odot v_t'
\end{aligned}
$$

¿Cuál es el problema? Que al inyectarse la penalización en $g_t$ se deforma la información del gradiente, y $v_t, s_t$ se ven afectados, al mismo tiempo que no se "achican" los parámetros tal como uno esperaría.

¿Pero esto no pasaba antes? TL;DR con Momentum no, con RMSProp sí pero Adam llegó muy rápido y es muy popular.

# SOTA: AdamW (2/2)

¿Cómo se arregla? Más fácil de lo que uno cree:

$$
\begin{aligned}
g_t &\leftarrow \nabla J(\theta_{t-1}) \\
v_t &\leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t \\
s_t &\leftarrow \beta_2 s_{t-1} + (1-\beta_2) g_t^2 \\
v'_t &\leftarrow v_t / (1-\beta_1^t) \\
s'_t &\leftarrow s_t / (1-\beta_2^t) \\
\theta_t &\leftarrow \theta_{t-1} - \frac{\gamma}{\sqrt{s'_t}+\epsilon} \odot v'_t - w\theta_{t-1}
\end{aligned}
$$

¿Por qué nos importa algo tan simple? Porque esto era EL algoritmo al menos hasta inicios de 2025. Salvo indicación contraria, todos los LLM de la era ChatGPT+ se asume que fueron entrenados con alguna variante para entrenamiento distribuido de AdamW.

# *SOTA++: Muon*

Uno de los últimos avances (fines 2024/inicios 2025), específicamente para matrices (2d) en capas no extremas en redes neuronales.

Es Momentum + ortogonalización equivalente a $UV^T$ si $X = U\Sigma V^T$... sí, rescaling de valores singulares vía DVS.

Sea $\theta \in \mathbb{R}^{m \times n}$, el algoritmo es:

$$
\begin{aligned}
G_t &\leftarrow \nabla_J(\theta_{t-1}) \\
B_t &\leftarrow \mu B_{t-1} + G_t \\
O_t &\leftarrow \mathit{Orthogonalize}(B_t) \\
\theta_t &\leftarrow \theta_{t-1} - \gamma O_t
\end{aligned}
$$

- Se usan (pocas) iteraciones de un algoritmo llamado *Newton Schulz* para esquivar la DVS propiamente dicha.
- Sólo mantiene 1 elemento extra ($B_t$), no 2 ($v_t, s_t$)
- No soporta parámetros no-2d, recomienda AdamW

Se recomienda leer el blog y el paper de Moonlight (aunque sea el abstract).

# 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.