CMM
Center for
Mathematical
Modeling

# ¿Cómo optimizar redes neuronales?

## Desde el SGD hasta Adam

**Diego Olguín**

DIM-CMM
Universidad de Chile

4 de septiembre, 2025

# Contenidos

CMM
Center for
Mathematical
Modeling

1. El problema de optimizar una red neuronal

2. El autogradiente

3. Mejorando el SGD: el momentum

4. Hablando de momentos: Adam

CMM
Center for
Mathematical
Modeling

# El problema de optimizar una red neuronal

DIM-CMM Universidad de Chile

# Problema de aprendizaje

CMM
Center for
Mathematical
Modeling

Supongamos que tenemos una variable aleatoria, que representa los datos de *input*, digamos $X$, y otra que representa el *output*, digamos $Y$. En un problema de aprendizaje nos gustaría, valga la redundancia, **aprender** una función $f$ tal que

$$Y = f(X)$$

En términos de un problema de optimización, esto es equivalente a minimizar según un criterio $\mathcal{L}$ sobre un espacio de funciones $\mathcal{F}$, esto es

$$\min_{f \in \mathcal{F}} \mathbb{E}[\mathcal{L}(f(X), Y)]$$

# Primeros problemas del problema

CMM
Center for Mathematical Modeling

Aquí ya vemos dos problemas:

- Calcular esa esperanza en general es realmente difícil al no tener una expresión cerrada.
- El espacio de decisión de infinito dimensional.

# Aproximar la solución con datos

CMM
Center for
Mathematical
Modeling

Entonces, solucionemos primero lo de la esperanza, supongamos que tenemos $N$ realizaciones del par aleatorio $(X, Y)$, $\{(x_i, y_i\}_{i=1}^N$ tal que

$$y_i = f(x_i)$$

luego, podemos decir que

$$\mathbb{E}[\mathcal{L}(f(X), Y)] \approx \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i)$$

que es lo que llamamos el **riesgo empírico**.

# Ahora sí, redes neuronales

CMM
Center for Mathematical Modeling

- Hasta el momento, no hemos introducido redes neuronales, y cualquier otra forma de poder aproximar esa función $f$ parece válida.
- Lo que haremos es reemplazar $f$, que es un elemento infinito dimensional, por una versión finito dimensional, es decir una **parametrización**, que sea lo más parecida posible.
- Esta parametrización será una **red neuronal**.

# Redes neuronales feedforward

CMM Center for Mathematical Modeling

## Definición (Red neuronal feedforward)

Una red neuronal $\Phi : \mathbb{R}^{n_{in}} \to \mathbb{R}^{n_{out}}$ de $L-1 \in \mathbb{N}$ capas ocultas de anchos $\{d^{(\ell)}\}_{\ell=0}^{L}$, con pesos $\{W^{(\ell)}\}_{\ell=1}^{L}$, con $W^{(\ell)} \in \mathbb{R}^{d^{(\ell)} \times d^{(\ell-1)}}$, sesgos $\{b^{(\ell)}\}_{\ell=1}^{L}$ con $b^{(\ell)} \in \mathbb{R}^{d^{(\ell)}}$ y funciones de pérdida $\{\phi^{(\ell)}\}_{\ell=1}^{L}$ con $\phi^{(\ell)} : \mathbb{R} \to \mathbb{R}$, se define mediante su output $\Phi(x)$ calculado según la recurrencia

$$x^{(0)} = x$$

$$x_j^{(\ell)} = \phi^{(\ell)} \left( \sum_{i=1}^{d^{(\ell-1)}} w_{ij}^{(\ell)} x_i^{(\ell-1)} + b_j^{(\ell)} \right), \quad 1 \leq j \leq d^{(\ell)}, \quad 1 \leq \ell \leq L$$

$$\Phi(x) = x^{(L)}$$

# Nuevo problema de optimización

CMM
Center for
Mathematical
Modeling

Llamemos a todos los parámetros de la red $\theta$ que están en un espacio de parámetros $\Theta$, que es finito dimensional, aunque puede ser enorme en la práctica. Entonces, cambiamos el problema de minimizar el riesgo empírico en un espacio de funciones

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i)$$

por uno en un espacio finito dimensional, que es el espacio de parámetros

$$\min_{\theta \in \Theta} \sum_{i=1}^{N} \mathcal{L}(\Phi_\theta(x_i), y_i),$$

donde $\Phi_\theta$ es la realización de una red neuronal de parámetros $\theta$.

# Problema para resolver en un computador

CMM
Center for Mathematical Modeling

- Ahora, este nuevo problema, sí es un problema que puede decidirse en un computador al ser completamente finito dimensional, aunque no deja de ser muy difícil, en el sentido de que, en general, es NP-hard.
- La pregunta natural es, ¿por qué redes neuronales para reemplazar a la $f$? (Esto también lo llamamos modelo subrogado).
- Y la respuesta está en que las redes neuronales tienen una gran capacidad de aproximar funciones, gracias al **teorema de aproximación universal**, que no veremos ahora.
- Además, tienen una gran capacidad de **generalizar** a datos nuevos, que es algo que tampoco veremos ahora en teoría.

# ¿Cómo intentamos resolver esto en un computador?

CMM
Center for Mathematical Modeling

Vale la pena notar que el problema de optimizar el riesgo empírico en el espacio de parámetros de una red neuronal es un problema **no convexo**, gracias a la no linealidad de la función de activación $\phi$ (que es importantísimo para que la red aproxime bien), con lo que se vuelve aún más difícil.
En la literatura de problema de optimización, existen dos gran familias de algoritmos:

- Algoritmos tipo descenso de gradiente.
- Algoritmos de búsqueda global o (en general llamados evolutivos).

# Algoritmos tipo descenso de gradiente

CMM
Center for Mathematical Modeling

Ventajas:

- Son baratos en términos computacionales.
- En general, sus hiperparámetros son pocos y sencillos de elegir.

Desventajas:

- Es muy fácil que caigan en un mínimo local y no puedan escapar de ahí, lo que obliga a ejecutar el algoritmo demasiadas veces desde distintos puntos iniciales.
- En ciertos casos puede ser sensible a la tasa de aprendizaje.

# Algoritmos evolutivos

CMM
Center for
Mathematical
Modeling

Estos algoritmos se llaman evolutivos porque hacen que ciertos puntos vayan 'evolucionando' para adaptarse a minimizar el criterio de optimización. Siguen ideas similares a **simulated annealing** y todos son naturalmente **estocásticos**.
Ventajas:

- En muchos contextos tienen la capacidad de alcanzar todos los mínimos globales de una función.

Desventajas:

- Sufren de **maldición de la dimensionalidad**, es decir, el tiempo de ejecución sube exponencialmente con la dimensión del espacio de búsqueda.

# Lo mejor de los dos mundos

CMM
Center for
Mathematical
Modeling

La idea es hacer un algoritmo barato, aprovechándose de técnicas tipo descenso de gradiente, con la estocasticidad como los algoritmos evolutivos que logra vencer la no convexidad del problema. De ahí nace la idea hacer estocástico el descenso de gradiente.
Primero, un algoritmo de descenso de gradiente clásico en este contexto se basaría en actualizar los parámetros de la forma

$$\theta_{k+1} = \theta_k - \alpha_k \left( \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta [\mathcal{L}(\Phi_\theta(x_i), y_i)] \right)$$

Que, para efectos de nuestro problema, ¡no tiene nada de estocástico!

# Optimizando por batches

CMM
Center for Mathematical Modeling

Podemos volverlo estocástico cambiando en cada iteración la función de pérdida, en vez de promediar todos los datos, solo promediar algunos e ir cambiándolos, lo que llamamos **batches**. Entonces, particionamos el conjunto de entrenamiento en $N_b$ grupos, de manera que estos se vean de la forma

$$\text{batch}_j = \{(x_i, y_i) : i \in \sigma_j \subseteq \{1, \dots, N\}\}, j \in \{1, \dots, N_b\},$$

tal que los $\sigma_j$ son partición de $\{1, \dots, N\}$. Así, en cada iteración se elige aleatoriamente un $j$ entre $1$ y $N_b$ de manera que el parámetro se actualiza como

$$\theta_{k+1} = \theta_k - \alpha_k \left( \frac{1}{|\sigma_j|} \sum_{i \in \sigma_j} \nabla_\theta [\mathcal{L}(\Phi_\theta(x_i), y_i)] \right).$$

# Comparemos

CMM
Center for Mathematical Modeling

- Pasemos al código y comparemos los dos algoritmos. También se puede complementar con el siguiente video.
- **Disclaimer:** en la sesión de hoy trabajaremos todo a mano, es decir, no ocuparemos ningún **backend** de *deep learning*. La idea es que se entienda qué hace un *backend* por dentro, cómo arma la red y cómo la optimiza. En la próxima sesión veremos otras arquitecturas y aplicaciones más complejas con PyTorch.

CMM
Center for
Mathematical
Modeling

# El autogradiente

DIM-CMM Universidad de Chile

# ¿Por qué esto funciona bien en NN?

CMM
Center for
Mathematical
Modeling

Algo importante del algoritmo del descenso de gradiente es, justamente, el cálculo del gradiente de la función de pérdida. Según lo anterior, la iteración del algoritmo sería

$$\begin{aligned}\theta_{k+1} &= \theta_k - \alpha_k \left( \frac{1}{|\sigma_j|} \sum_{i \in \sigma_j} \nabla_\theta [\mathcal{L}(\Phi_{\theta_k}(x_i), y_i)] \right) \\ &= \theta_k - \alpha_k \left( \frac{1}{|\sigma_j|} \sum_{i \in \sigma_j} \langle \nabla_1 \mathcal{L}(\Phi_{\theta_k}(x_i), y_i), \nabla_\theta \Phi_{\theta_k}(x_i) \rangle \right),\end{aligned}$$

es decir, ¡necesitamos la expresión del gradiente de $\Phi_\theta$ en función de sus parámetros!

# Una opción a diferencias finitas

CMM
Center for Mathematical Modeling

- Alguien más clásico podría decir que es bueno calcular ese gradiente vía diferencias finitas, pero en la práctica es caro y comete un error que crece con la dimensión del problema, así que no es una buena idea.
- Pero, recordar que las redes neuronales son súper simples: son composición de funciones afines con una función no lineal conocida, por tanto, si tuviésemos tiempo podríamos calcular ese gradiente de manera analítica y **exacta** con todas las reglas de derivación que conocemos.

# Grafo de derivación

CMM
Center for
Mathematical
Modeling

- En la práctica, dado que las redes neuronales son enormes, hacer el cálculo de ese gradiente sigue siendo muy exhaustivo, pero se puede hacer algoritmicamente, con lo que se denomina el algoritmo del autogradiente, basado en un **grafo de derivación**.
- Veamos esto en un computador, en el clásico caso en el que la función de pérdida $\mathcal{L}$ es la pérdida cuadrática

$$\mathcal{L}(f(X), Y) = (f(X) - Y)^2.$$

En este caso, llamamos al problema un problema de **regresión**.

CMM
Center for
Mathematical
Modeling

# Mejorando el SGD: el momentum

DIM-CMM Universidad de Chile

# Ser estocástico no basta

CMM
Center for
Mathematical
Modeling

El descenso de gradiente aunque tenga buenas propiedades, puede fallar en mínimos locales. Es por ello que se propone añadirle a la iteración actual un término relacionado con el parámetro de la iteración anterior, dándole más **inercia** al parámetro, haciéndolo más reacio al cambio. Esto se llama **momentum**.

# Un momentum

Abusaremos un poco de la notación, definiendo

$$g_k := \frac{1}{|\sigma_j|} \sum_{i \in \sigma_j} \langle \nabla_1 \mathcal{L}(\Phi_{\theta_k}(x_i), y_i), \nabla_\theta \Phi_{\theta_k}(x_i) \rangle$$

Sea $\beta_k \in [0, 1)$, luego se define el momentum como

$$m_{k+1} = \beta_k m_k + (1 - \beta_k) g_k,$$

con $m_0 = 0$, luego la iteración del algoritmo es

$$\theta_{k+1} = \theta_k - \alpha_k m_k.$$

# Interpretación física del momentum

CMM
Center for Mathematical Modeling

Tomemos la ecuación del momentum y restemos por $m_k$, es decir

$$m_{k+1} - m_k = (\beta_k - 1)m_k + (1 - \beta_k)g_k = (1 - \beta_k)(g_k - m_k),$$

llevando esta ecuación al continuo:

$$\frac{dm(t)}{dt} = (1 - \beta(t))(g(t) - m(t)).$$

Por otro lado, haciendo lo mismo para $\theta$

$$\frac{d\theta(t)}{dt} = -\alpha(t)m(t)$$

# Interpretación física del momentum

CMM
Center for Mathematical Modeling

Así, tomando la segunda derivada

$$\begin{aligned}\frac{d^2\theta(t)}{dt^2} &= -\alpha'(t)m(t) - \alpha(t)m'(t) \\ &= -\alpha'(t)m(t) - \alpha(t)(1-\beta(t))(g(t)-m(t)) \\ &= (\alpha(t)(1-\beta(t)) - \alpha'(t))m(t) - \alpha(t)(1-\beta(t))g(t) \\ &= \gamma_1(t)m(t) - \gamma_2(t)g(t)\end{aligned}$$

Donde $\gamma_1(t) = \alpha(t)(1-\beta(t)) - \alpha'(t))$ y $\gamma_2(t) = \alpha(t)(1-\beta(t))$ son positivos.

# Interpretación física del momentum

CMM
Center for Mathematical Modeling

- El término $m(t)$ aumenta la aceleración, que representa la inercia del sistema y el término $g(t)$ desacelera, representando una fuerza externa.
- Es decir, el parámetro acelera en función de una fuerza que es un equilibrio entre inercia y ”gravedad”.

CMM
Center for
Mathematical
Modeling

# Hablando de momentos: Adam

DIM-CMM Universidad de Chile

# Primer momento

CMM
Center for
Mathematical
Modeling

- Notar que la función de pérdida del problema es estocástica, dado que nos estamos basando en una cantidad finita de datos y por tanto, aunque el riesgo empírico aproxima la esperanza a minimizar, no deja de tener un componente aleatorio.
- Esto provoca que la cantidad dada por el gradiente del riesgo empírico tenga un sesgo y varianza, que si no corregimos puede dar errores muy grande. Por ello, miremos la iteración de SGD con momentum

$$m_{k+1} = \beta_k m_k + (1 - \beta_k) g_k,$$

$$\theta_{k+1} = \theta_k - \alpha_k m_k$$

# Corrección del primer momento

CMM
Center for Mathematical Modeling

Consideremos el caso en que los parámetros son constantes, esto es, $\beta_k = \beta_1$ y $\alpha_k = \alpha$, es más, tomaremos $\beta_1 \approx 1$ con lo que el primer momento se escribe como

$$m_k = (1 - \beta_1) \sum_{i=1}^{k} \beta_1^{k-i} g_i.$$

Si tenemos que $\mathbb{E}[g_i] \approx \mathbb{E}[g_k]$, se tiene que

$$\mathbb{E}[m_k] \approx \left( (1 - \beta_1) \sum_{i=1}^{k} \beta_1^{k-i} \right) \mathbb{E}[g_k] = (1 - \beta_1) \frac{(1 - \beta_1^k)}{(1 - \beta_1)} \mathbb{E}[g_k] = (1 - \beta_1^k) \mathbb{E}[g_k].$$

# Corrección del primer momento

CMM
Center for Mathematical Modeling

Con lo que el sesgo de $m_k$ se debe corregir, quedando un primer momento corregido

$$\hat{m}_k = \frac{m_k}{(1-\beta_1^k)}.$$

Entonces, esta cantidad actúa como un estimador insesgado del gradiente de la función de pérdida (el riesgo) original. Ya hemos arreglado el problema del sesgo, aún así, la varianza se puede disparar, por lo que trabajaremos eso.

# Introduciendo el segundo momento

CMM
Center for
Mathematical
Modeling

Para aproximar la varianza, introduciremos el siguiente término en las iteraciones

$$v_{k+1} = \beta_2 v_k + (1-\beta_2) g_k \odot g_k,$$

en donde $g_k \odot g_k$ es el producto Hadamard de vectores, o *element-wise*, denoratemos $g_k^2 = g_k \odot g_k$. Con las mismas hipótesis de antes

$$\mathbb{E}[v_k] = (1-\beta_2^k)\mathbb{E}[g_k^2],$$

en donde el término $\mathbb{E}[g_k^2]$ es la varianza no centrada.

# Corrección del segundo momento

Center for Mathematical Modeling

Con esto, el segundo momento no centrado y corregido viene dado por

$$\hat{v}_k = \frac{v_k}{1 - \beta_2^k}.$$

Con ello, el término

$$\frac{\hat{m}_k}{\sqrt{\hat{v}_k}},$$

es un estimador insesgado para el gradiente de la función de pérdida y con varianza ajustada.

# Actualización del parámetro

CMM
Center for Mathematical Modeling

Luego, el parámetro se ajusta como

$$\theta_{k+1} = \theta_k - \alpha \frac{\hat{m}_k}{(\sqrt{\hat{v}_k} + \varepsilon)}$$

en donde $\varepsilon \approx 0$ es para evitar divisiones por cero o divergencias numéricas.

# ¿Teorema central del límite?

CMM
Center for Mathematical Modeling

- El término

$$\frac{\hat{m}_k}{\sqrt{\hat{v}_k}},$$

  es un símil al ajuste del teorema central del límite, en donde queda un término centrado en el parámetro a estimar (el gradiente de la pérdida en este caso) y con varianza unitaria.
- Este término es lo que se denomina **signal-to-noise** que cuantifica que tan importante es el estimador del parámetros en comparación al ruido del problema, lo que es clave en muchas aplicaciones, en particular en optimización estocástica.

CMM
Center for
Mathematical
Modeling

# ¡Gracias por su atención!

**Diego Olguín**

DIM-CMM
Universidad de Chile

4 de septiembre, 2025