# Clase 070 — Gradient Descent: batch, stochastic, mini-batch

Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4.  Duración estimada: 60 min.

# Clase 070 — Gradient Descent: batch, stochastic, mini-batch

*Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4. Duración estimada: 60 min.*

## Objetivo

Que el alumno entienda gradient descent como motor de optimización para entrenar modelos lineales cuando la ecuación normal no escala, y sepa elegir entre batch (BGD), stochastic (SGD) y mini-batch GD según tamaño del dataset, ruido tolerable y costo por iteración. Que además dimensione el rol del learning rate y del feature scaling, y use SGDRegressor de scikit-learn.

## Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Explicar el gradiente de la MSE en regresión lineal y por qué moverse en - minimiza el costo.
2. Diferenciar BGD vs SGD vs mini-batch en términos de costo por iteración, varianza del paso y memoria.
3. Diagnosticar la curva de costo (divergente, oscilante, lenta, suave) e inferir si el learning_rate está mal seteado.
4. Aplicar feature scaling (StandardScaler) antes de cualquier GD y justificar por qué sin escalado SGD diverge o tarda 100×.
5. Entrenar SGDRegressor con learning_rate='invscaling' y comparar coeficientes contra LinearRegression (ecuación normal).

## Temas

| # | Tema | Por qué importa |
|---|---|---|
| 1 | Gradiente de la MSE y regla de update θ := | Es el núcleo de todo ML moderno (incluye r |
| 2 | Batch GD: usa todo el dataset por step | Convergencia suave, pero O(m) por iteració |
| 3 | Stochastic GD: 1 muestra por step | Muy rápido y escala, pero ruidoso; nunca c |
| 4 | Mini-batch GD: lotes de 32–256 | Equilibrio práctico, aprovecha vectorizaci |
| 5 | Learning rate η y learning schedule | Demasiado alto diverge, demasiado bajo no |
| 6 | Feature scaling como prerrequisito | Sin escalar, las curvas de nivel son elips |
| 7 | SGDRegressor de sklearn | API estándar, soporta partial_fit (online |

## Definiciones y características

Gradiente MSE(θ)

: Vector de derivadas parciales del costo respecto a cada parámetro. Apunta en la dirección de máximo crecimiento del error → restarlo (-η·) reduce el costo. Para MSE: $= (2/m)\cdot X^T(X\theta - y)$.

Learning rate η (eta)

: Tamaño del paso. Hiperparámetro crítico. Muy alto: el costo diverge o oscila. Muy bajo: tarda eternidades. Valores típicos: 0.001 a 0.1.

Batch Gradient Descent (BGD)

: Cada update usa todas las m muestras. Costo por iteración O(m·n). Determinista, converge suave al mínimo global (en problemas convexos como MSE). Inviable con $m > 10^6$.

Stochastic Gradient Descent (SGD)

: Cada update usa una sola muestra (elegida al azar). Costo O(n) por iteración. Camino ruidoso, "rebota" cerca del mínimo sin asentarse → requiere learning_schedule decreciente para que el ruido baje con el tiempo.

Mini-batch Gradient Descent

: Usa lotes de tamaño b (típicamente 32, 64, 128, 256). Combina lo mejor: vectorizable (rápido en GPU), menos ruidoso que SGD, mucho más liviano que BGD. Es el estándar de facto en deep learning.

Epoch

: Una pasada completa sobre el dataset. En SGD, una epoch = m updates. En mini-batch con batch_size b, una epoch = m/b updates.

Learning schedule

: Función que baja η con el tiempo. Patrón común: $\eta_t = \eta / (1 + \text{decay} \cdot t)$. En sklearn: learning_rate='invscaling' con eta0 y power_t.

Feature scaling

: Llevar las features a escala comparable (StandardScaler: media 0, std 1). Sin esto, una feature con valores $[0, 10^6]$ domina el gradiente y otra con [0, 1] es invisible → GD zigzaguea por un valle alargado.

## Dataset / recursos

sklearn.datasets.fetch_california_housing (20.640 filas, 8 features, escalas muy distintas → caso ideal para mostrar la necesidad de scaling).

## Ejercicios

1. BGD a mano. Implementá BGD en NumPy para regresión lineal sobre un dataset sintético (y = 4 + 3x + ruido). Loopeá 1000 iteraciones con η=0.1. Graficá la trayectoria de θ, θ y la curva de costo.

2. SGD a mano. Mismo dataset. Implementá SGD con learning_schedule $\eta_t = 5 / (50 + t)$. Compará la trayectoria contra BGD: tendría que ser visiblemente más ruidosa pero más rápida en wall-clock.

3. Efecto del learning rate. Corré BGD con η ∈ {0.001, 0.01, 0.1, 0.5, 1.0}. Graficá las 5 curvas de costo en un mismo plot. Identificá cuál diverge y cuál es absurdamente lenta.

4. Scaling sí/no. Sobre California Housing, entrená SGDRegressor (a) sin escalar y (b) con StandardScaler. Reportá n_iter_ y score en cada caso. Tiene que haber un orden de magnitud de diferencia.

5. SGDRegressor vs LinearRegression. Entrená ambos sobre California Housing escalado. Compará coeficientes y $R^2$. Tienen que dar muy parecidos (SGD es aproximación estocástica de la solución cerrada).

## Homework verificable

Notebook con California Housing: (a) train/test split 80/20; (b) pipeline StandardScaler + SGDRegressor(max_iter=1000, tol=1e-3, learning_rate='invscaling', eta0=0.01); (c) reportá $R^2$ en test y

n_iter_ real; (d) repetí sin scaler y mostrá que n_iter_ se dispara o que $R^2$ cae; (e) graficá la curva de loss vs epoch usando partial_fit en loop manual.

Criterio de aceptación: $R^2$ test ≥ 0.55 con scaler. Sin scaler, $R^2$ baja al menos 0.1 puntos o aparece ConvergenceWarning.

## Errores comunes

| Síntoma / mensaje | Causa y cómo arreglar |
|---|---|
| ConvergenceWarning: Maximum number of iter | Olvidaste el StandardScaler. Fix: meté SGD |
| El costo diverge (sube en lugar de bajar) | learning_rate demasiado alto. Fix: bajalo |
| SGD nunca "se asienta", el costo oscila pa | Falta learning_schedule que baje η con el |
| Coeficientes de SGDRegressor distintos a L | Si entrenaste con datos no escalados, los |
| partial_fit parece arrancar de cero en cad | Si reinstanciás el modelo dentro del loop, |

## Preguntas frecuentes

¿BGD, SGD o mini-batch?

Mini-batch en el 90% de los casos. BGD solo si el dataset entra cómodo en RAM y m < ~$10^4$. SGD puro (batch_size=1) casi nunca en la práctica — sirve para entender el concepto y para online learning donde llegan muestras una por una. SGDRegressor de sklearn internamente puede comportarse como mini-batch dependiendo del solver.

¿Por qué LinearRegression no usa GD?

Porque para MSE existe solución cerrada (ecuación normal: $\theta = (X^TX)^{-1}X^Ty$). Es exacta y no tiene hiperparámetros. Pero invertir $X^TX$ es $O(n^3)$ en features y $O(m \cdot n^2)$ en muestras → con muchas features ($n > 10^4$) o $X^TX$ mal condicionada, GD gana.

¿Cómo elijo η?

Empezá con 0.01. Si diverge, dividilo por 10. Si converge pero lento, multiplicalo por 3. Mejor aún: grid search sobre eta0 con cross-validation. En deep learning hay schedulers más sofisticados (Adam, cosine annealing), pero para ML clásico invscaling alcanza.

¿Qué tamaño de mini-batch uso?

32, 64, 128 o 256. Potencias de 2 porque la GPU las prefiere. Más chico → más ruido, más updates por epoch. Más grande → más estable, pero menos pasos. 32 es el default histórico (Bengio); 256 es común con datasets grandes.

¿StandardScaler o MinMaxScaler antes de GD?

StandardScaler por default — centra en 0 (importante para que el gradiente no tenga sesgo direccional) y normaliza varianza. MinMaxScaler lo usás si necesitás un rango acotado (ej. imágenes a [0,1]).

## Referencias

- Géron, cap. 4 — Training Models, sección "Gradient Descent".
- scikit-learn — SGDRegressor

- scikit-learn — Stochastic Gradient Descent user guide
- Sebastian Ruder — An overview of gradient descent optimization algorithms

## Siguiente clase

Clase 071 — Regresión polinomial

## Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

## Archivos complementarios

- notebook.ipynb