Gradiente Estocástico

SGD, SGD con momento, Adagrad

Andrea Arguello, David Cuellar

Métodos numéricos

22 de noviembre de 2021

UVG
UNIVERSIDAD
DEL VALLE
DE GUATEMALA

Motivación

Limitaciones del descenso al gradiente tradicional:

- Estar calculando derivadas para *cada punto/observación* en el conjunto de datos consume mucho tiempo y recursos
- La memoria necesaria es proporcional al tamaño del dataset

Este descenso al gradiente también es conocido como **descenso al gradiente con batch**, pues calcula el costo o pérdida *J* de los parámetros θ de todos los datos a la vez. Es decir, donde η es el learning rate:

$$\theta = \theta - \eta \nabla_\theta J(\theta, X, Y)$$

Supóngase el siguiente caso:

- El modelo que se desea optimizar usa 20000 genes para predecir si alguien va a morir.
- Se tiene un conjunto de datos 1,000,000 de muestras
- Mil iteraciones de descenso al gradiente serían

$$1,000,000x20,000x1000 = 2,000,000,000,000$$

de términos a calcular

Solución: Descenso al gradiente estocástico

Y si en vez de utilizar todos los datos, ¿solo se toma una fracción? Esto es lo que hace el **descenso al gradiente estocástico**.

Terminología

Antes de seguir, algunos conceptos importantes:

- ***epoch***: cuando un conjunto de datos completo pasa por una red neuronal (*forward* y *backward*)
- ***mini batch***: si no se va a o no es posible pasar *dataset* completo, se pasa un *mini batch* - una "porción" del conjunto de datos con un tamaño fijo de datos (***batch size***)
- **iteración**: cuando se realiza una actualización de los parámetros. Es equivalente a la cantidad de *batches* necesarios para realizar un *epoch*.

Pseudocódigo

Algorithm 1 Descenso al gradiente estocástico (SGD)

Input: vector inicial θ, taza de aprendizaje η
while no converja y k $\leq$ número máximo de iteraciones **do**
    revolver el conjunto de datos
    for $i \leftarrow 1, 2, ..., n$ **do**
        Seleccione una observación x_i con su respectivo y_i
        $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta, x_i, y_i)$
    end for
end while

Mini-batch gradient descent

La idea del GD por lotes es combinar ambos conceptos: usar un batch o lote de una cantidad *b* de datos para actualizar el parámetro en cuestión, reduciendo así la varianza para llevar a una convergencia más estable; sin embargo, esto hace que el camino para alcanzar los mínimos globales no es tan fluido como el DG.

Pseudocódigo

Algorithm 2 Mini batch SGD

Input: vector inicial θ, taza de aprendizaje η, *batch size b*
while no converja y k $\leq$ número máximo de iteraciones **do**
    revolver el conjunto de datos
    generar $\lceil \frac{n}{b} \rceil$ *batches* de tamaño *b*
    for *batch* in *batches* **do**
        Seleccione las observaciones del *batch* x_{batch} con sus respectivos y_{batch}
        $\theta \leftarrow \theta - \eta \nabla_\theta J(\theta, x_{batch}, y_{batch})$
    end for
end while

Comparación

Algoritmo	batch size	batches
GD	n	1
SGD	1	n
Mini-batch SGD	b	$\lceil \frac{n}{b} \rceil$

Table 1: Comparación de batches para los tres algoritmos, suponiendo un conjunto de n datos

Comparación de convergencia

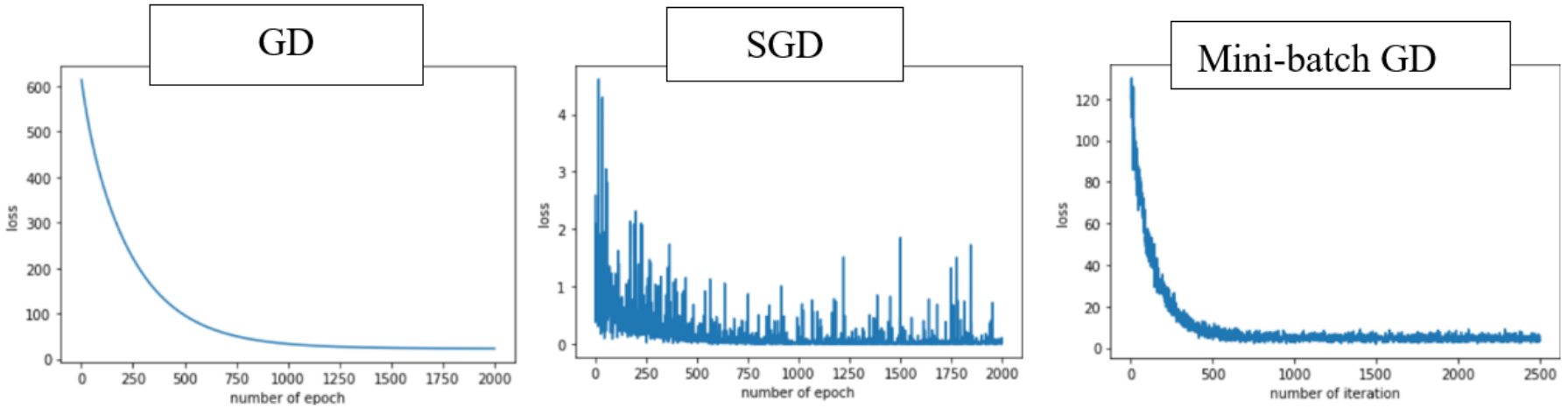


Figure 1: Comparación de pérdida vs epoch del DG y sus variaciones. A mayor batch size, menor variación. Fuente: Ruder, 2016.

Algoritmos de optimización

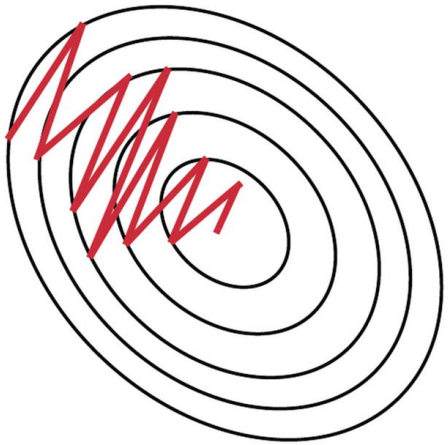
Existen varios algoritmos de optimización que utilizan el concepto del descenso al gradiente estocástico. En esta presentación, presentaremos dos:

- SGD con momento
- AdaGrad

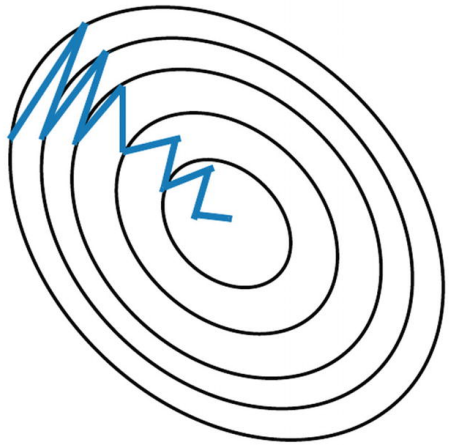
SGD con momento

SGD tiene problemas para navegar por áreas donde la superficie se curva mucho más abruptamente en una dimensión que en otra. El objetivo del SGD con momento es acelerar el proceso de descenso, amortiguando las oscilaciones, agregando un vector de velocidad a la ecuación del SGD. La idea básica es usar la fracción de la actualización anterior para la actual. El algoritmo introduce un nuevo hiperparámetro β, llamado momento.

Comparación: SGD sin y con momento



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

Figure 2: Comparación de trayectoria de SGD sin y con momento. Fuente: Bisong, 2019

Pseudocódigo

Algorithm 3 Descenso de gradiente estocástico con momento

Input: vector inicial θ, vel. inicial m, taza de aprendizaje η, momento β, *batch size b*
while no se cumpla el criterio de paro **do**
    Tómese un minibatch de las $x^{(i)}$ del dataset con sus objetivos $y^{(i)}$
    Calcular aprox. de gradiente $g \leftarrow \frac{1}{n}\nabla_\theta J(\theta, x^{(t)}, y^{(t)})$
    $m \leftarrow \beta m - \eta g$
    $\theta \leftarrow \theta + m$
end while

AdaGrad

En los optimizadores que hemos visto hasta ahora, la taza de aprendizaje η es constante. Sin embargo, esta se puede actualizar automáticamente para usar un tamaño de paso adaptativo para cada variable de entrada en la función objetivo. Estos se llaman Algoritmos de Gradiente Adaptativo (abreviados **AdaGrad**, por su nombre en inglés)

Pseudocódigo

Algorithm 4 AdaGrad

Input: vector inicial θ, taza de aprendizaje η, *batch size b*, error ϵ
while no se cumpla el criterio de convergencia **do**
    revolver el conjunto de datos
    generar $\lceil \frac{n}{b} \rceil$ *batches* de tamaño *b*
    for *batch* in *batches* **do**
        calcular $g_t = \nabla_{\theta_t} J(\theta_t, x_{batch}, y_{batch})$
        $\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{\epsilon + \sum_{j=1}^{t} g_j^2}} g_t$
    end for
end while

Desventajas

Ya que se está dividiendo por la suma de todos los gradientes al cuadrado hasta el momento *t*, la tasa de aprendizaje es cada vez más pequeña, y a partir de ese momento, el algoritmo ya no puede adquirir conocimientos adicionales.
Específicamente a raíz de este problema, soluciones como Adadelta y RMSProp han sido propuestas.

Implementación

Se implementaron los algoritmos de Mini batch SGD, SGD con momento y AdaGrad sobre un dataset de 20640 datos para regresión lineal múltiple para predicción de precios de casas. Originalmente con 10 variables dependientes, el modelo trabajado utiliza siete (se excluyó una variable categórica y latitud y longitud), las cuales fueron normalizadas (incluyendo la variable de respuesta). Para todos los algoritmos, se utilizó como función de pérdida la mitad del error cuadrático medio (MSE).

Resultados

Método	Iters.	Error	Loss
GD	22	10^{-3}	0.7893
SGD	1313	$4 \cdot 10^{-4}$	0.0925
Momento	1204	$2 \cdot 10^{-4}$	0.0571
AdaGrad	11650	$3 \cdot 10^{-4}$	0.0298

Table 2: Resultados de los distintos algoritmos

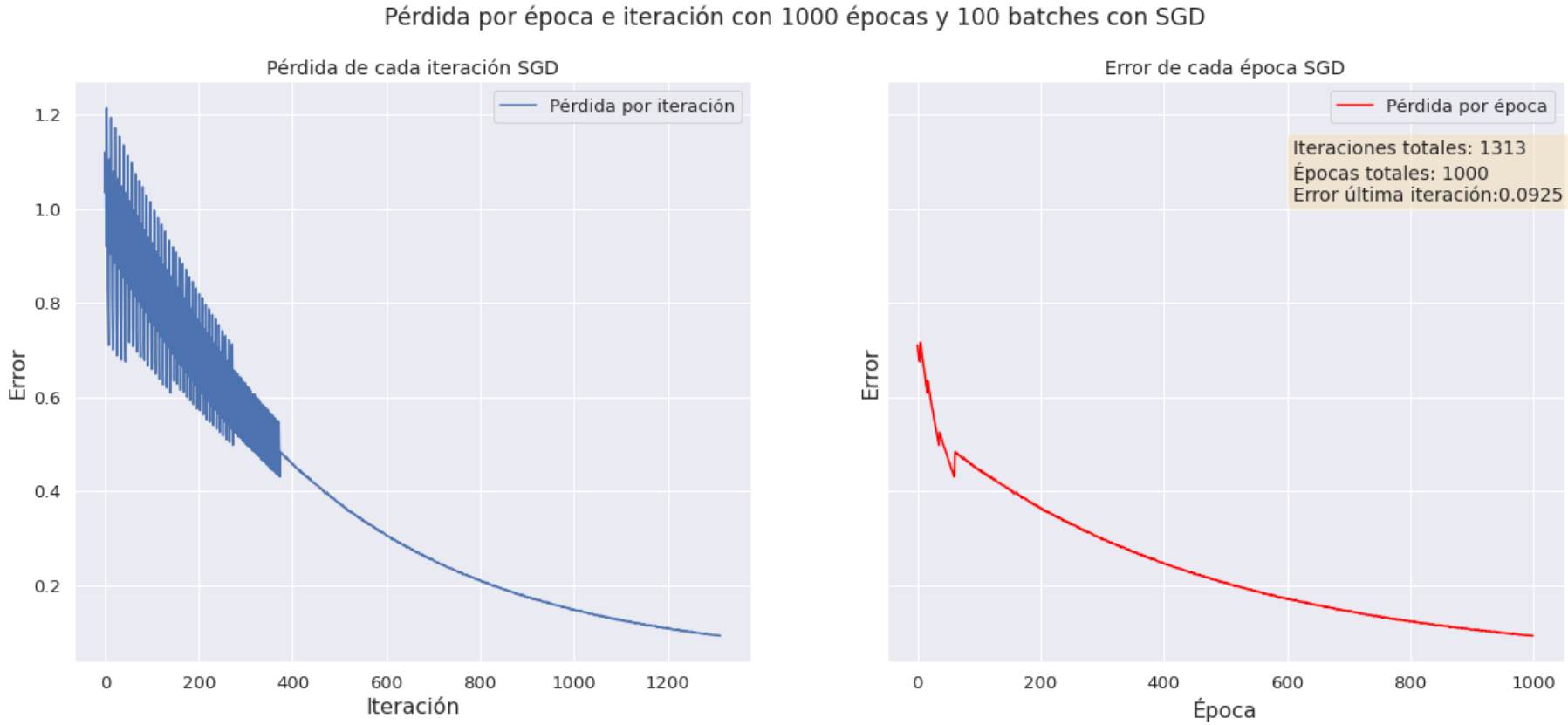


Figure 3: Resultados de loss vs epoch y error vs epoch para SGD con 1000 epochs y 100 batches

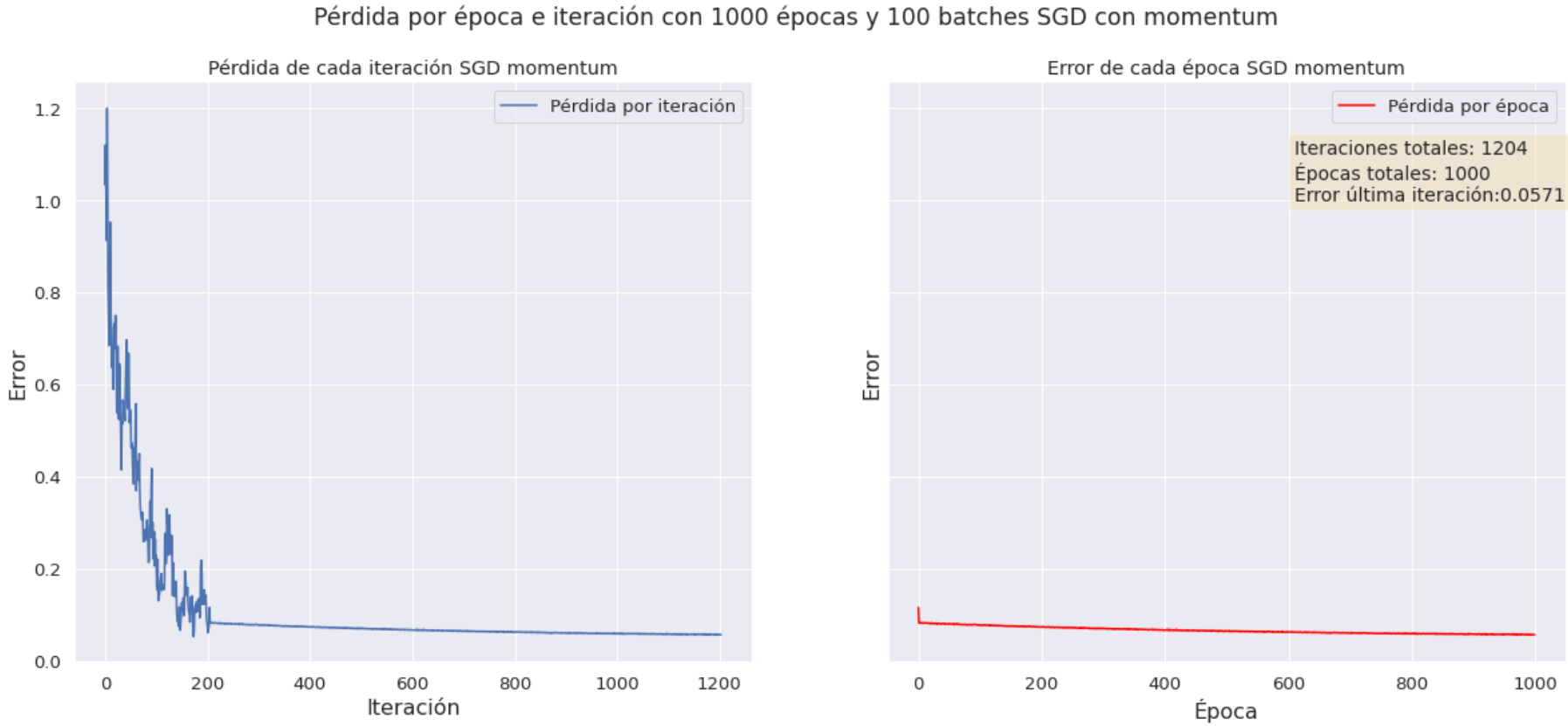


Figure 4: Resultados de loss vs epoch y error vs epoch para SGD con momento con 1000 epochs y 100 batches

UVG
UNIVERSIDAD
DEL VALLE
DE GUATEMALA

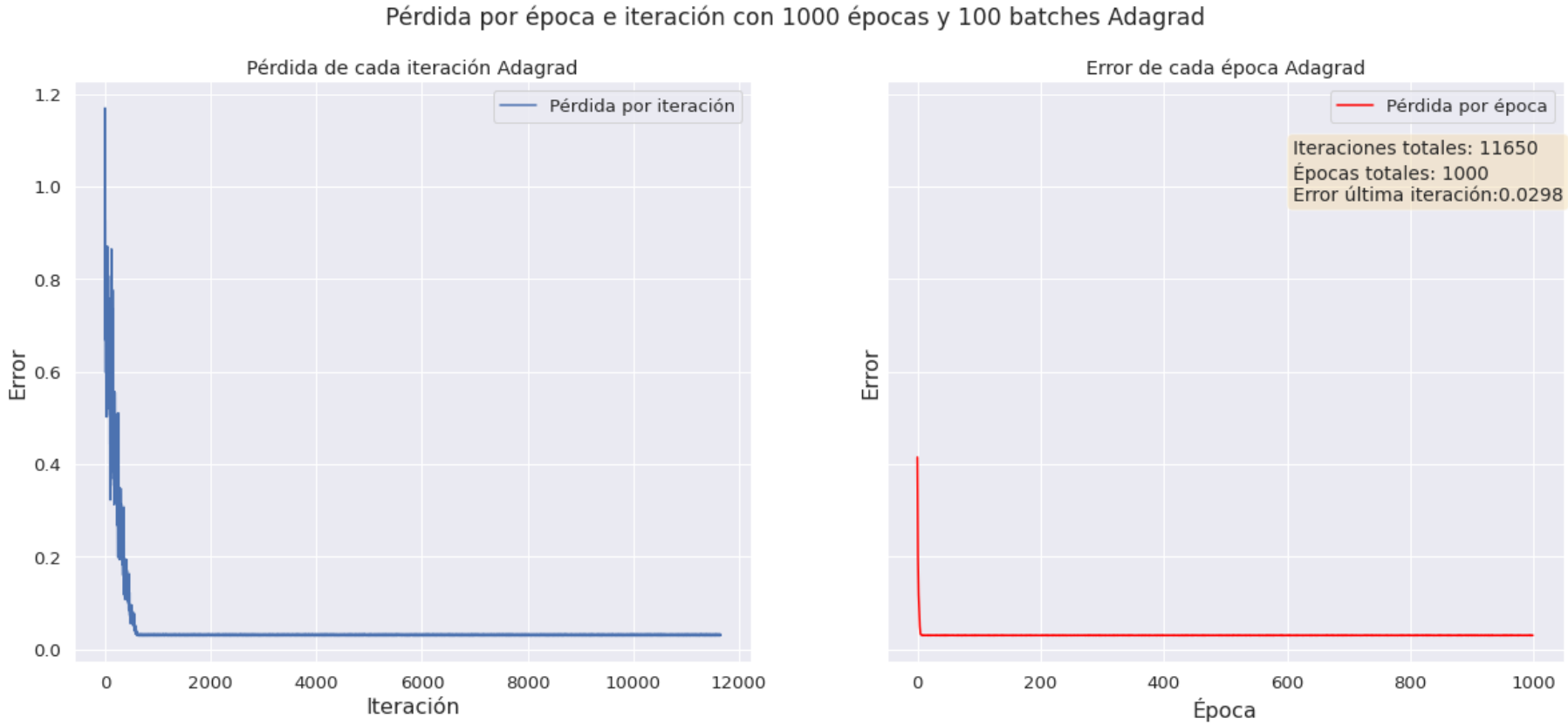


Figure 5: Resultados de loss vs epoch y error vs epoch para AdaGrad con 1000 epochs y 100 batches

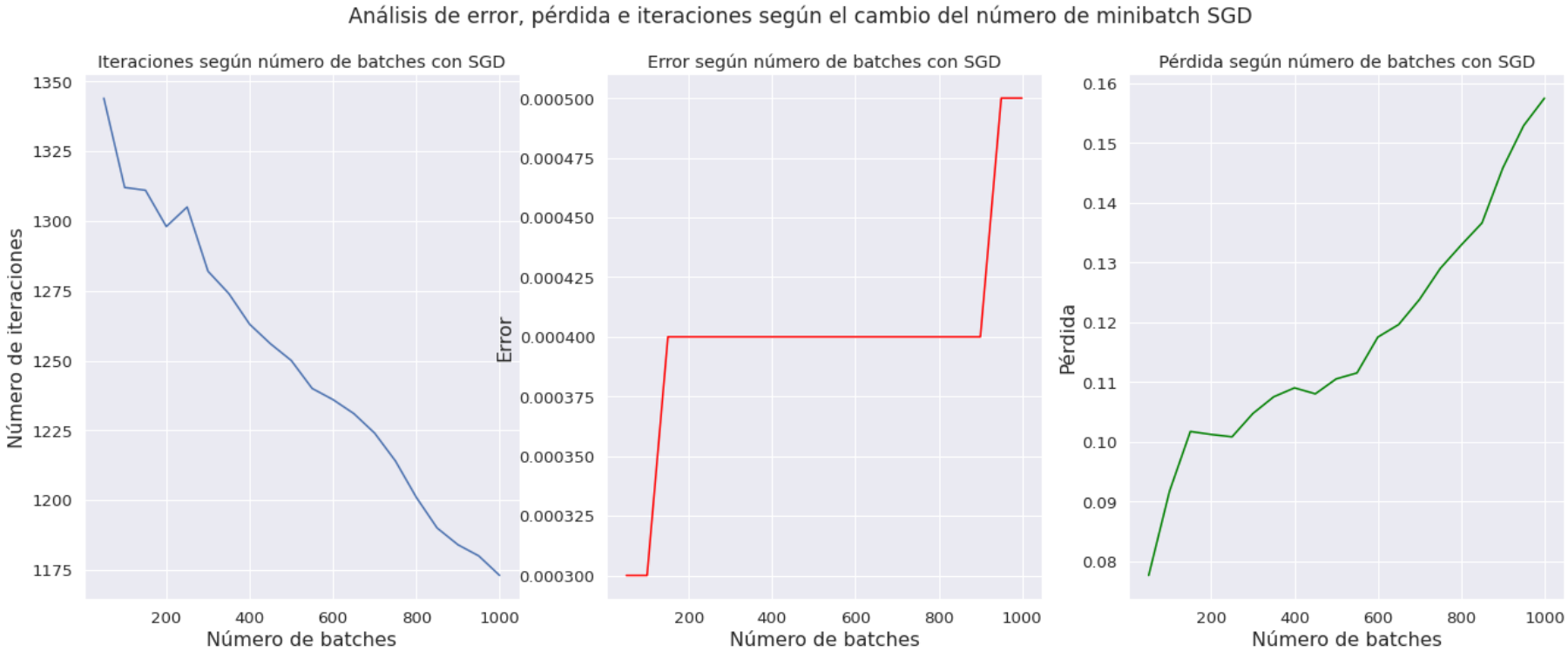


Figure 6: Resultados de SGD variando el numero de batches, con tolerancia de 0.001 y 1000 epochs

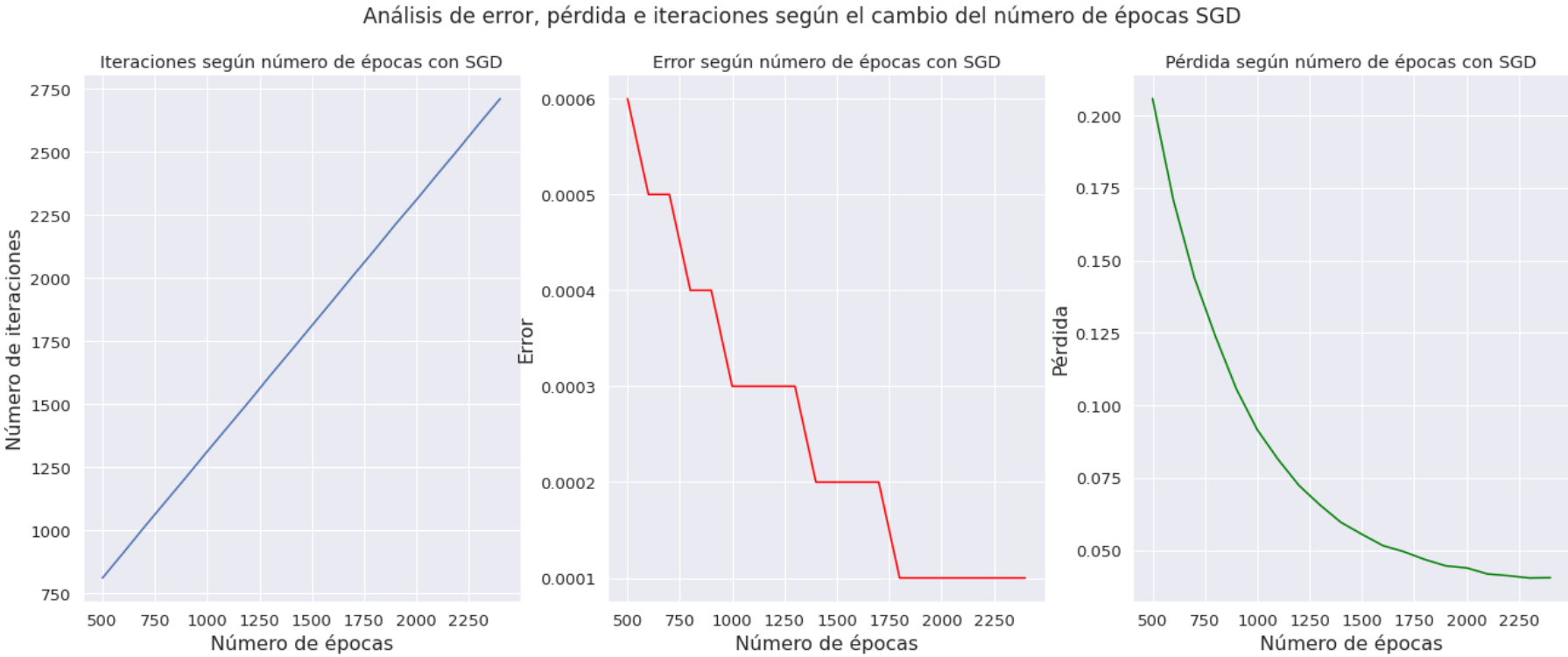


Figure 7: Resultados de SGD variando el número de epochs, con tolerancia de 0.001 y batch size de 100

Referencias

1. Bisong, E. (2019). *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Apress, Berkeley, CA.
2. Mustapha, A., Mohamed, L. y Ali, K.. (2021). "Comparative study of optimization techniques in deep learning: Application in the ophthalmology field". *Journal of Physics: Conference Series*, 1743(012002).
3. Ruder, S. (19 de enero de 2016). "An overview of gradient descent optimization algorithms". *Sebastian Ruder*. `https://ruder.io/optimizing-gradient-descent/`