Universidad de Sevilla
Escuela Superior de Ingeniería

Trabajo Final:

Explorando el Descenso del Gradiente Estocástico. Desde la Teoría Matemática hasta su aplicación en el Aprendizaje Automático

Autor:
- David Pineda Peña

Tutor:
- Pedro Almagro Blanco

Enero, 2024

Índice general

1 Introducción

El campo del aprendizaje automático ha experimentado un crecimiento exponencial en la última década, impulsando avances significativos en diversas áreas como la visión por ordenador, el procesamiento del lenguaje natural y la inteligencia artificial. Uno de los componentes clave en estos avances es la optimización de algoritmos, teniendo como puntos centrales, el desarrollo del *hardware* y el aumento en la capacidad computacional, junto con el desarrollo de técnicas como retropropagación [1], o *backpropagation* y el Descenso del Gradiente, o *Gradient Descent* (GD).

En este contexto, el GD, y su variante, el Descenso del Gradiente Estocástico, o *Stochastic Gradient Descent* (SGD), emergen como técnicas fundamentales. El GD ajusta iterativamente los parámetros de una función para minimizarla, siendo esencial en el entrenamiento de modelos de aprendizaje automático, especialmente en contextos con gran cantidad de datos y funciones de pérdida complejas. El SGD introduce un elemento de aleatoriedad y exploración al seleccionar muestras, oaleatoriamente del conjunto de datos en cada iteración, lo que puede llevar a una mejor generalización y a evitar mínimos locales subóptimos.

Este trabajo tiene como objetivo explorar de manera integral el SGD, haciendo un desarrollo completo como es inevitable del GD fundacional. Comenzaremos estableciendo una base teórica sólida de ñas matemáticas que sustentan tanto al GD como al SGD. Luego, nos adentraremos en los aspectos algorítmicos y prácticos, proporcionando una guía detallada para su implementación. Además, aplicaremos el SGD a un conjunto de datos real, analizando los resultados y destacando las lecciones aprendidas. Por último, discutiremos los desafíos actuales, limitaciones y posibles direcciones para futuras investigaciones en este campo dinámico y en constante evolución.

2 Fundamentos Teóricos

En el siguiente capítulo se procede a dar las definiciones y demostrar las bases matemáticas sobre las que se desarrolla el algoritmo del Descenso del Gradiente, y por consiguiente, su variante Estocástica.

2.1. Matemáticas del Descenso del Gradiente

Enfoque de Regresión Lineal

Ecuación de la Recta

La forma más simple de una ecuación lineal se representa por medio de la ecuación de la recta en un plano bidimensional:

$$y = mx + c \tag{2.1}$$

donde:

- y, es la variable dependiente (lo que intentamos predecir)
- x, es la variable independiente (entrada o característica)
- m, es la pendiente de la línea (indica cuánto cambia y por cada unidad de cambio en x)

- c, es la ordenada al origen y (el valor de y cuando $x = 0$, es decir, cuando x corta el plano)

Ejemplo: Si $m = 2$ y $c = 3$, la ecuación se convierte en $y = 2x + 3$. Esto significa que por cada unidad que aumenta x, y aumenta en 2 unidades, y cuando $x = 0$, y es 3. Representada dicha recta en el plano, se vería tal que:

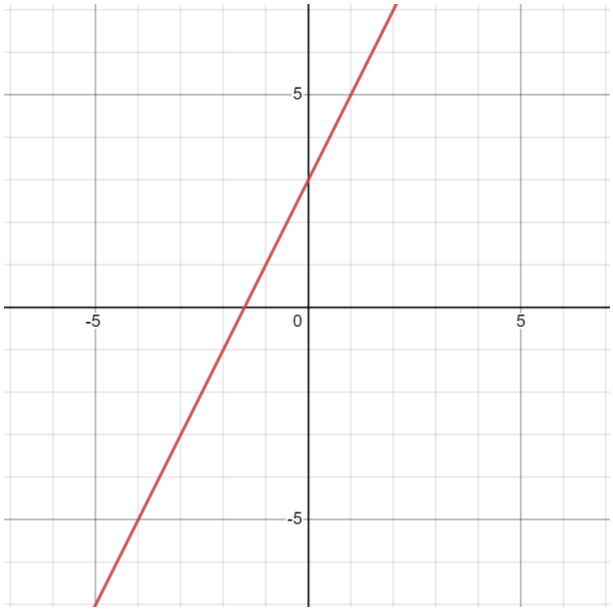

Figura 2.1: Ecuación de la recta $y = 2x + 3$

Función de Coste

En la regresión lineal, nuestro objetivo es encontrar la línea que mejor se ajuste a una nube de puntos, o más concretamente, nuestros datos. La función de coste ayuda a medir la precisión de nuestro modelo. Una función de coste comúnmente utilizada en problemas de regresión es el Error Cuadrático Medio, o *Mean Squared Error* (MSE), definido como:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + c))^2 \tag{2.2}$$

donde:

- N, es el número total de observaciones
- y_i, es el valor real del punto i en la nube de puntos o conjunto de datos
- mx_i+c, es el valor predicho para el punto i-ésimo usando nuestro modelo de regresión lineal.

Ejemplo: Considera dos puntos de datos $(1, 3)$ y $(2, 7)$ con un modelo inicial $y = 2x+3$ (es decir, $m = 2$, $c = 3$).

$$\text{Predicción para } (1,3): \quad 2 \times 1 + 3 = 5 \tag{2.3}$$

$$\text{Predicción para } (2,7): \quad 2 \times 2 + 3 = 7 \tag{2.4}$$

$$\text{Cálculo de MSE:} \quad MSE = \frac{1}{2}\left[(3-5)^2 + (7-7)^2\right] = 2 \tag{2.5}$$

En el ejemplo 2.5, el valor del error según el MSE entre el valor real y el valor predicho para cada punto de datos es:

- Para el punto (1, 3): Error $= (3-5)^2 = 4$
- Para el punto (2, 7): Error $= (7-7)^2 = 0$

Así, el modelo tiene un error total de $4/2 = 2$, ya que el MSE calcula el error promedio de todas las instancias de puntos, tal y como se observa en la siguiente imagen:

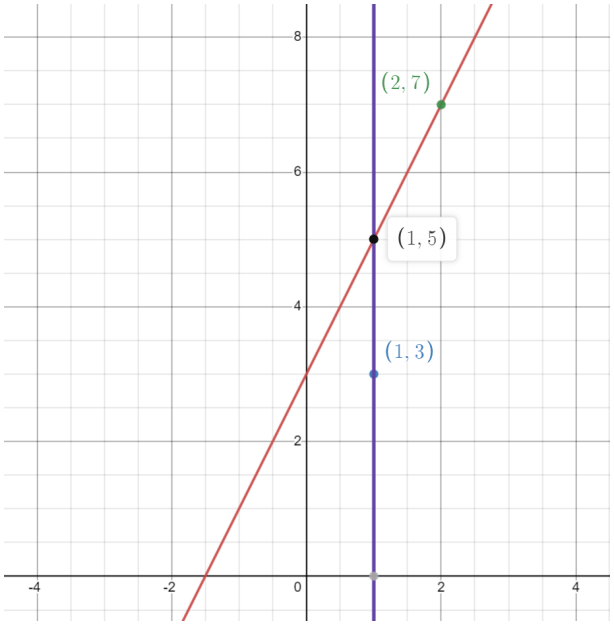

Figura 2.2: Ecuación de la recta $y = 2x + 3$ con los puntos $1, 3)$ y $2, 7)$

Si se busca minimizar el error, se debe ajustar los valores de m y c en el modelo de regresión lineal $y = mx + c$, para que las predicciones del modelo sean lo más cercanas posibles a los valores reales. Es en este punto, donde se aplican algoritmos de optimización como el GD.

Descenso del Gradiente

El Descenso del Gradiente es un algoritmo de optimización iterativo utilizado para minimizar la función de coste. Involucra calcular el gradiente de la función de coste y actualizar los parámetros en la dirección que más rápido disminuya dicha función de coste.

Derivadas Parciales

Tomando el MSE como función de coste, las derivadas parciales del MSE con respecto a m y c se calculan de la siguiente manera.

Con respecto a **m**, se utiliza la regla de potencia,

$$\frac{d}{dx}[u^n] = n \cdot u^{n-1} \cdot \frac{du}{dx} \tag{2.6}$$

y la regla de la cadena,

$$\frac{d}{dx}[f(g(x))] = f'(g(x)) \cdot g'(x) \tag{2.7}$$

Para cada término en la sumatoria, diferenciamos $(y_i - (mx_i + c))^2$ con respecto a m:

1. Sea $u = y_i - (mx_i + c)$. Entonces, $u^2 = (y_i - (mx_i + c))^2$.
2. Aplicamos la regla de potencia: $\frac{d}{dm}[u^2] = 2u \cdot \frac{du}{dm}$.
3. Como $u = y_i - (mx_i + c)$, $\frac{du}{dm} = -x_i$.

Sustituyendo, obtenemos:

$$\frac{d}{dm}[(y_i - (mx_i + c))^2] = 2(y_i - (mx_i + c)) \cdot (-x_i)$$

Finalmente, promediamos esta derivada sobre todos los puntos de datos:

$$\frac{\partial MSE}{\partial m} = \frac{-2}{N} \sum_{i=1}^{N} (y_i - (mx_i + c))x_i \tag{2.8}$$

Con respecto a **c**, si se aplican las mismas reglas 2.6 2.7 se obtiene que para cada término en la sumatoria, si se diferencia $(y_i - (mx_i + c))^2$ con respecto a c:

1. Sea $u = y_i - (mx_i + c)$. Entonces, $u^2 = (y_i - (mx_i + c))^2$.

2. Aplicando la regla de potencia: $\frac{d}{dc}[u^2] = 2u \cdot \frac{du}{dc}$.

3. Como $u = y_i - (mx_i + c)$, $\frac{du}{dc} = -1$.

Sustituyendo, se obtiene:

$$\frac{d}{dc}[(y_i - (mx_i + c))^2] = 2(y_i - (mx_i + c)) \cdot (-1)$$

Finalmente, se hace el promedio de esta derivada sobre todos los puntos de datos:

$$\frac{\partial MSE}{\partial c} = \frac{-2}{N} \sum_{i=1}^{N} (y_i - (mx_i + c)) \tag{2.9}$$

Regla de Actualización

Los parámetros se actualizan de la siguiente manera:

$$m := m - \alpha \frac{\partial MSE}{\partial m}$$
$$c := c - \alpha \frac{\partial MSE}{\partial c}$$

donde α es la tasa de aprendizaje, la cual se detallará más adelante.

Enfoque del Perceptrón Simple

Modelo de Perceptrón

Un perceptrón simple consiste en múltiples entradas, pesos asociados a cada entrada, un bias, y una función de activación para producir la salida, tal y como se puede apreciar en la figura 2.3. La salida se calcula como:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

donde:

- x_i son las entradas o características
- w_i son los pesos de las conexiones entre las neuronas
- b es el bias

- f es la función de activación
- y es la salida del perceptrón

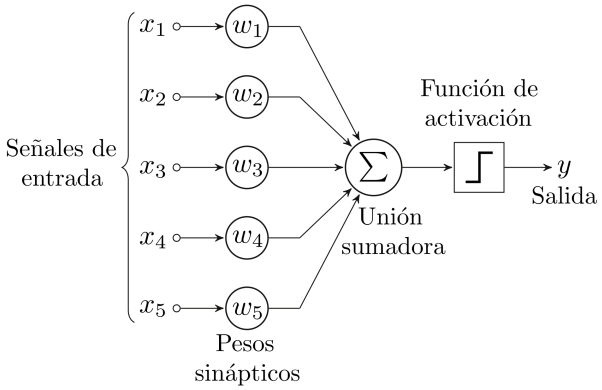

Figura 2.3: Modelo de perceptrón con cinco señales de entrada, FUENTE: [2].

Cálculo de las Derivadas Parciales

Para actualizar los pesos y el bias en dicho perceptrón simple, es necesario calcular las derivadas parciales de la función de coste con respecto a cada uno de ellos. Por seguir en la misma línea que en el enfoque de regresión lineal y no extender demasiado las demostraciones matemáticas, no se va a tener en cuenta ninguna función de activación en el modelo, simplificando así la aplicación de la regla de la cadena.

Para un peso w_j:

$$\frac{\partial MSE}{\partial w_j} = \frac{\partial MSE}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_j}$$

donde:

- $\frac{\partial MSE}{\partial \hat{y}_i} = \frac{-2}{N} \sum (y_i - \hat{y}_i)$, es la derivada parcial de la función de coste MSE sobre las predicciones del modelo
- $\frac{\partial \hat{y}_i}{\partial w_j} = f' \left(\sum w_j x_{ij} + b \right) \cdot x_{ij}$.

Analogamente, para el bias b:

$$\frac{\partial MSE}{\partial b} = \frac{\partial MSE}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial b}$$

donde $\frac{\partial \hat{y}_i}{\partial b} = f' \left(\sum w_j x_{ij} + b \right)$.

Actualización de Pesos y Bias

De igual manera que se actualizaban m y c en el enfoque de regresión lineal, los pesos y el bias se actualizan en cada iteración del proceso de entrenamiento de la siguiente forma:

$$w_j \leftarrow w_j - \alpha \cdot \frac{\partial MSE}{\partial w_j}$$

$$b \leftarrow b - \alpha \cdot \frac{\partial MSE}{\partial b}$$

2.2. La idea detrás del GD y SGD

En cálculo, la derivada de una función nos muestra cuánto varía un valor dependiendo de cómo variamos sus argumentos. Las derivadas son importantes para la optimización porque las derivadas cero nos indican un mínimo, máximo o punto de equilibrio.

El gradiente de una función C de varias variables independientes $\langle v_1, \ldots, v_r \rangle$ se denota como $\nabla C(v_1, \ldots, v_r)$ y se define como la función vectorial de las derivadas parciales de C con respecto a cada variable independiente $\nabla C = (\partial C/\partial v_1, \ldots, \partial C/\partial v_r)$

El valor no nulo del gradiente de una función C en un punto dado define la dirección y la tasa de mayor incremento de C.

Así que, cuando trabajamos con el descenso del gradiente buscamos la dirección con la disminución más rápida en la función de coste. Esto viene determinado por el gradiente negativo $-\nabla C$.

Es por ello que la idea detrás del GD se basa en, elegir inicialmente de manera arbitraria la posición del punto o vector v y moverlo en la dirección en la que el gradiente disminuya más rápidamente.

Un concepto muy importante a introducir en este momento es el de tasa de aprendizaje, o *learning rate*, denominado por la letra griega η.

Ya que la posición nueva a la que vamos a mover nuestro v es en la dirección del gradiente negativo $v \rightarrow v - \eta \nabla C$, donde η es nuestro valor positivo de "tasa de aprendizaje", el cual determina cómo de grande o de pequeña es la distancia que nos vamos a mover en el espacio.

Es decir, si decimos que η tiene un valor muy bajo, supongamos $\eta = 0{,}000001$, nuestro algoritmo convergerá muy lentamente, mientras que si lo instanciamos con un valor alto

($\eta = 1$, el algoritmo dará saltos muy grande, haciendo que tenga problemas para converger, incluso llegando a escapar de los mínimos que buscamos.

3 Aspectos Algorítmicos

En este capítulo, exploramos el núcleo del Descenso de Gradiente Estocástico, sus variaciones y consideraciones clave para su implementación.

3.1. Pseudocódigo y descripción algorítmica del SGD

El SGD actualiza los parámetros del modelo iterativamente basándose en el cálculo del gradiente para todo el conjunto de entrenamiento en cada paso. El pseudocódigo para el SGD es el siguiente:

Algorithm 1 Descenso de Gradiente Estocástico (SGD)

```
Inicializar el vector de parámetros w y la tasa de aprendizaje η.
repeat
    Mezclar aleatoriamente los ejemplos en el conjunto de entrenamiento.
    for i = 1, 2, ..., n do
        Calcular el gradiente ∇Q_i(w).
        Actualizar w := w − η∇Q_i(w).
    end for
until se obtenga un mínimo aproximado
```

Este enfoque iterativo con aleatoriedad añade ruido en la actualización de los parámetros, permitiendo una mejor exploración en la convergencia hacia un mínimo global, siempre dependiendo de la naturaleza convexa o pseudoconvexa de la función objetivo.

3.2. Variantes del SGD (por ejemplo, mini-lote SGD)

Una variante popular del SGD es el SGD de mini-lote, o *mini-batch*, que equilibra la eficiencia del cálculo del gradiente con la estabilidad en la convergencia. En lugar de seleccionar un único ejemplo del conjunto de entrenamiento, el SGD de mini-lote utiliza un pequeño conjunto (mini-lote) para cada iteración en la actualización de parámetros.

Además, el SGD por mini-lotes reduce la varianza de las actualizaciones de los parámetros, lo que puede llevar a una convergencia más estable y rápida en comparación con el SGD puro. También, permite el aprovechamiento de las capacidades de cálculo en paralelo de las GPU y librerías de vectorización, mejorando aún más la eficiencia del entrenamiento en práctica.

4 Implementación

En el siguiente capítulo se detalla el código que implementa el Descenso del Gradiente Estocástico en el lenguaje de programación Python.

4.1. Descenso del Gradiente aplicado a un punto en una función convexa

En este apartado se va a implementar el algoritmo de GD en el escenario más básico posible, el cual es un punto en una función convexa sobre el plano bidimensional, moviéndose hacia un mínimo de la función.

Para ello, a partir de las demostraciones e intuiciones dadas en el capítulo 2, se procede a implementar la función de *gradient_ descent*, la cual a partir de los siguientes parámetros:

- *gradient*, la pendiente de la recta tangente a la gráfica
- *start*, el valor de inicio en el eje X
- *learn_ rate*, la tasa de aprendizaje η mencionada en el capítulo 2
- *n_ iter*, el número de veces que se va a actualizar la posición calculando el gradiente y aplicando la tasa de aprendizaje
- *tolerance*, una constante que para el proceso de iteración en caso de cumplirse

Para la función $f(x) = x^2$ con $\eta = 0{,}2$ y $x = 10$, la ejecución del código es la siguiente:

```python
import numpy as np

def gradient_descent(gradient, start, learn_rate, n_iter=50, tolerance=1e-06):
    vector = start # starting point
    vector_history = [vector]  # store the history of vectors
    for _ in range(n_iter):
        diff = -learn_rate * gradient(vector)

        # if the difference between the vectors is smaller than the tolerance, stop
iteratinif np.all(np.abs(diff) <= tolerance):
            break
        vector += diff
        vector_history.append(vector)  # append the new vector to the history

    print(f'Starting at x= {start}, found minimum at x= {vector}')
    return vector, vector_history


if __name__ == '__main__':
    gradient_descent(gradient=lambda x: 2 * x, start=10.0, learn_rate=0.2)


>>> Starting at x= 10.0, found minimum at x= 2.210739197207331e-06
```

Figura 4.1: Método gradient_descent para un punto en una función convexa

Dibujo de la gráfica de Función y Camino de Descenso

Para poder visualizar de una mejor forma cómo se aplica el algoritmo de GD y de qué manera actualiza la posición del punto dado, en base a los parámetros, se ha desarrollado una función *plot_gradient_descent*.

El código y la gráfica resultante para el mismo caso de la función $f(x) = x^2$ con $\eta = 0{,}2$ y $x = 10$, se obtiene:

```python
import matplotlib.pyplot as plt


def plot_gradient_descent(gradient_descent_path, function, range_x):
    # generate points for the function plot
    x = np.linspace(range_x[0], range_x[1], 100)
    y = function(x)

    # extract the gradient descent path
    gd_x = [v for v in gradient_descent_path]
    gd_y = [function(v) for v in gradient_descent_path]

    plt.figure(figsize=(10, 6))
    plt.plot(x, y)
    plt.scatter(gd_x, gd_y, color='red')
    plt.plot(gd_x, gd_y, color='red')
    plt.show()

# run gradient descent
optimal_v, gd_path = gradient_descent( gradient=lambda v: 2 * v,
                                       start=10.0,
                                       learn_rate=0.2
)

# plot function and path
plot_gradient_descent(gd_path, function=lambda v: v ** 2, range_x=[-10,
10])
```

Figura 4.2: Método para dibujar la gráfica de la función y camino del GD

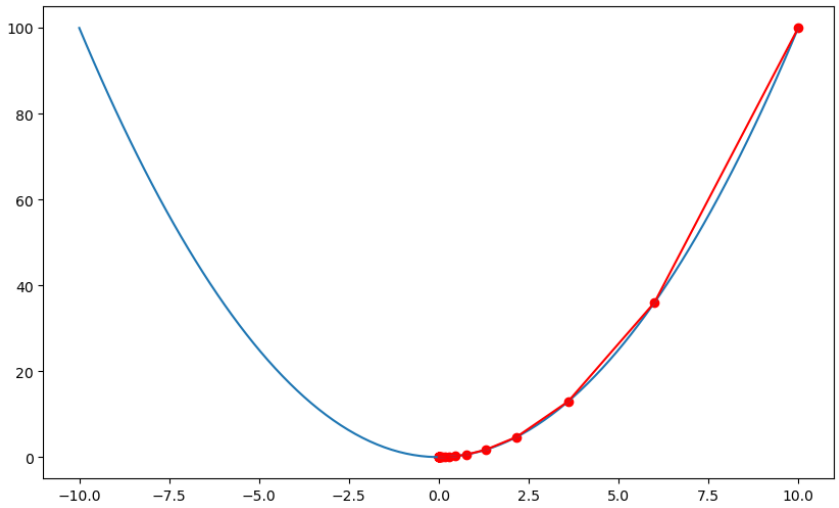

Figura 4.3: GD sobre un punto $x_0 = 10$ en la función $f(x) = x^2$ y $\eta = 0{,}2$

Para confirmar la importancia del parámetro de tasa de aprendizaje η sobre la convergencia del modelo, se ha estudiado diferentes configuraciones del mismo, y probado también en otra función convexa más compleja:

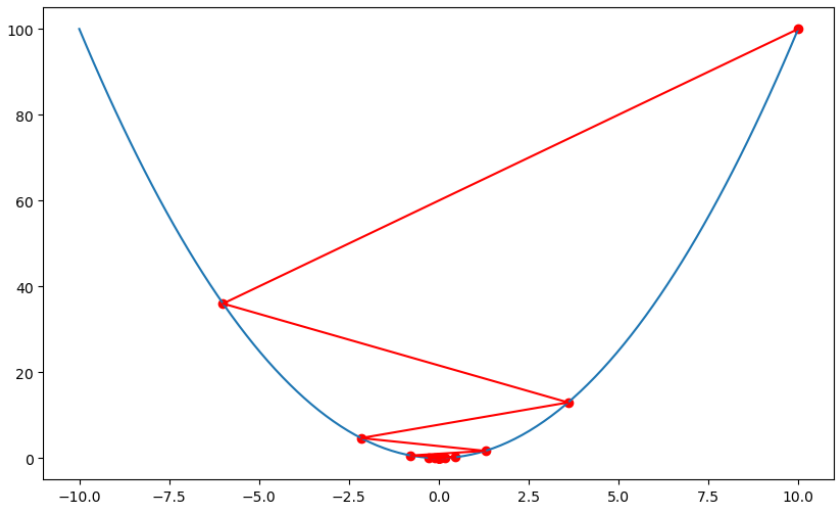

Figura 4.4: $\eta = 0{,}8$

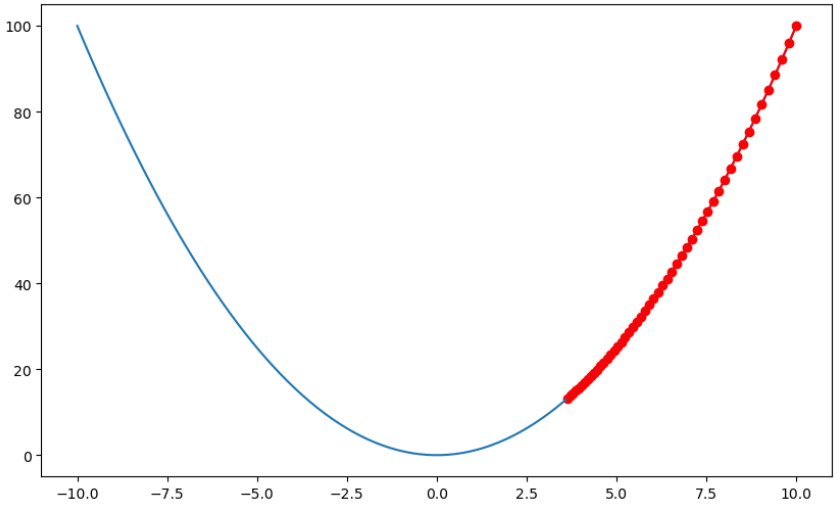

Figura 4.5: $\eta = 0{,}01$

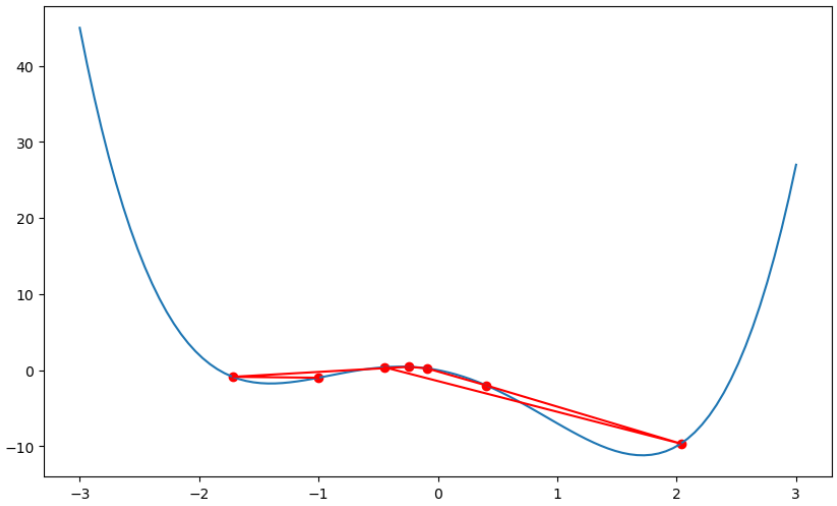

Figura 4.6: GD de 6 iteraciones sobre un punto $x_0 = -1$ en la función $f(x) = x^4 - 5x^2 - 3x$ y $\eta = 0{,}24$

4.2. Descenso del Gradiente aplicado a los pesos y bias de un conjunto de datos

De igual manera que se desarrolló en el capítulo 2, se puede enfocar el problema de optimización a los valores de los pesos y bias en un modelo de red neuronal.

Para ello se ha llevado a cabo una implementación sobre un conjunto de datos simple y de carácter público llamado *Advertising and Sales*[3].

El código implementado se puede encontrar en el repositorio personal de este proyecto, el cual aparece referenciado en el capítulo 8 de este proyecto, y se compone de los siguientes pasos:

1. Primero, se ha realiza la carga de datos por medio del fichero *.csv* y estandarizado los datos.

2. Seguidamente, se han implementado los métodos de inicialización, cálculo de predicción y cómputo del coste del algoritmo.

3. Finalmente, se ha realizado el algoritmo de GD, tanto en un paso, como en un número de iteraciones dada, para estudiar su convergencia.

5 Caso de estudio

En el siguiente capítulo se presenta una comparativa entre la implementación del SGD por mini-lotes en el lenguaje de programación Python y su implementación con una de las bibliotecas de Redes Neuronales más reconocida y popularmente usada, *Keras*.

5.1. Conjunto de datos: *Boston Housing Prices*

El conjunto de datos escogido para el estudio entre implementación manual e implementación en Keras es el de *Boston Housing*[4].

Este conjunto de datos es una versión reducida del *dataset* original de *Boston Housing* el cual es ampliamente utilizado para análisis de regresión y contiene información acerca de las viviendas en el área de Boston.

Dicho conjunto de datos cuenta con 496 muestras y 4 atributos, y es particularmente interesante por las implicaciones sociales y económicas que pueden inferirse de los datos. Las columnas de interés seleccionadas para este estudio son:

- **RM**: número promedio de habitaciones por vivienda.
- **LSTAT**: porcentaje de población considerada de clase baja.
- **PTRATIO**: ratio de alumnos por profesor en la ciudad.
- **MEDV**: valor medio de las viviendas ocupadas por sus propietarios

Siendo esta última **MEDV** la variable objetivo del estudio

5.2. Comparativa resultados manuales frente a Keras

Tras la implementación de ambos casos, y bajo la misma condición de parámetros, se ha obtenido una gráfica de pérdida del modelo a lo largo de las iteraciones prácticamente idénticas en ambas implementaciones:

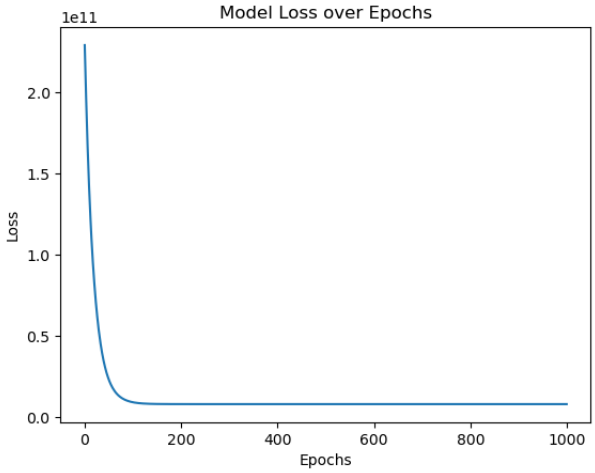

Figura 5.1: Gráfica de pérdida del modelo en la implementación manual

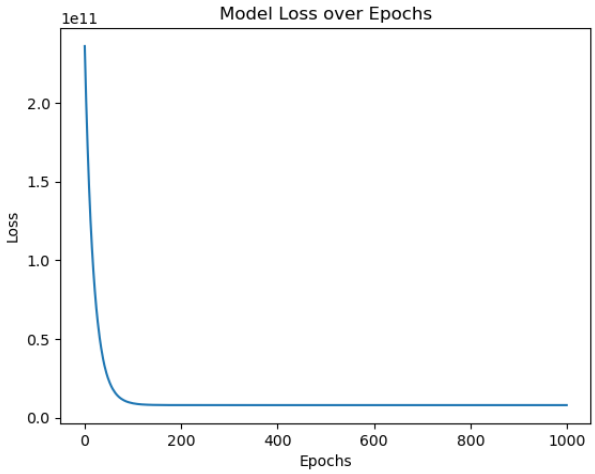

Figura 5.2: Gráfica de pérdida del modelo en la implementación con Keras

Cabe recalcar que existe una única diferencia entre ambas implementaciones, y se debe a que Keras, por defecto, en sus modelos neuronales incluye la función de activación, mientras que en la implementación manual y a lo largo de este trabajo no se ha llevado a cabo, por motivos de tiempo y debido a que este estudio no pretende ser un Trabajo de Fin de Grado (TFG).

De todas formas, se ha ajustado dicha función de activación en el modelo de Keras para que fuese una activación lineal, igual a la identidad, y no añadiese ningún tipo de

no-linealidad al modelo.

En cuanto a tiempos de ejecución, por este motivo comentado de aplicar una función de activación, se obtiene unos resultados de:

- $\approx$ 2.2s, para la implementación manual
- $\approx$ 31.1s, para la implementación en Keras.

6 Desafíos y Limitaciones

En el siguiente capítulo, se abordarán algunos de los principales desafíos y limitaciones asociados con el uso del Descenso de Gradiente Estocástico. El SGD es un pilar en la optimización de algoritmos de aprendizaje automático, pero su aplicación práctica puede encontrarse con varios obstáculos y restricciones que impactan en su eficacia y eficiencia.

6.1. Desafíos y problemas comunes en el uso del SGD

- **Selección de la Tasa de Aprendizaje η:** La eficiencia del SGD depende en gran medida de la elección de la tasa de aprendizaje. Una tasa demasiado alta puede provocar que el algoritmo diverja, mientras que una tasa muy baja puede llevar a una convergencia extremadamente lenta, tal y como se ha demostrado con ejemplos en capítulos anteriores.

- **Dependencia de la Inicialización:** Los resultados del SGD pueden variar significativamente dependiendo de cómo se inicialicen los parámetros, pudiendo conducir al algoritmo a convergencias subóptimas.

- **Minimización de Funciones no Convexas:** En problemas con funciones de costo no convexas, SGD puede quedar atrapado en mínimos locales o puntos de silla, lo que impide llegar hasta el mínimo global.

- **Necesidad de Ajuste Fino:** El ajuste fino de los hiperparámetros (como la tasa de aprendizaje, y el tamaño del lote en sus variantes *batch* y *mini-batch*) es crítico y puede ser un proceso tedioso basado en prueba y error.

6.2. Limitaciones del SGD en ciertos escenarios

- **Datos de Alto Volumen y Alta Dimensión:** El SGD puede enfrentar dificultades en conjuntos de datos con un volumen extremadamente alto o con muchas características (alta dimensionalidad), donde la eficiencia computacional se convierte en un desafío. Es por ello que existen sus variantes enfocadas a lotes del conjunto de datos, buscando reducir dicho tiempo computacional.
- **Dependencia de la Calidad de los Datos:** El rendimiento del SGD es altamente dependiente de la calidad de los datos de entrenamiento. Datos ruidosos, incompletos o sesgados pueden llevar a resultados poco confiables, ya que el SGD, de por sí hace actualizacioens ruidosas de los parámetros.

7 Conclusiones

En el siguiente capítulo se hace un resumen general de los hallazgos y aprendizajes obtenidos a lo largo de este estudio sobre el SGD. Adicionalmente, discuten los aspectos más significativos de la investigación, incluyendo las contribuciones clave, y las perspectivas para futuras investigaciones en este campo.

7.1. Reflexión sobre el estudio

- **Base Matemática y Teórica:** Se estableció una sólida base matemática para el entendimiento del GD y su variante estocástica, abarcando desde los fundamentos teóricos hasta los aspectos algorítmicos detallados.

- **Implementación de Algoritmos:** La correcta implementación de los algoritmos de GD y SGD, con una mención especial en la variante de mini-lotes, demostró la aplicabilidad práctica de estos conceptos teóricos.

- **Cumplimiento de Objetivos:** Se alcanzaron los objetivos propuestos al inicio del estudio, proporcionando una comprensión integral tanto del GD como del SGD y sus aplicaciones en conjuntos de datos reales.

7.2. Reflexión sobre la efectividad del SGD

- **Versatilidad y Eficiencia:** El SGD, especialmente con mini-lotes, se mostró eficiente y competitivo en el caso de estudio realizado en el capítulo 5.
- **Futuras Mejoras:** Como margen de mejora, se propone incluir la integración de funciones de activación en los cálculos de los algoritmos y bases matemáticas, lo cual se asemejaría aún más al verdadero funcionamiento de estos algoritmos.

8 Repositorio

El código de todo este trabajo puede encontrarse en el siguiente repositorio de GitHub del alumno.

Incluida una copia de este documento final del estudio.

Bibliografía

[1] D. E. Rumelhart, G. E. Hinton y R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, n.º 6088, págs. 533-536, 1986.

[2] Wikipedia, *Perceptrón — Wikipedia, The Free Encyclopedia*, http://es.wikipedia.org/w/index.php?title=Perceptr%C3%B3n&oldid=148351059, [Online; accessed 15-January-2023], 2023.

[3] sazid28, *Advertising Dataset*, https://www.kaggle.com/datasets/sazid28/advertising.csv, Accessed: 2024-01-24, 2023.

[4] C. Schirmer, *Boston Housing MLND Dataset*, https://www.kaggle.com/datasets/schirmerchad/bostonhoustingmlnd, Accessed: 2024-01-25, 2023.