Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías
División de Tecnologías para la Integración Ciber-Humana

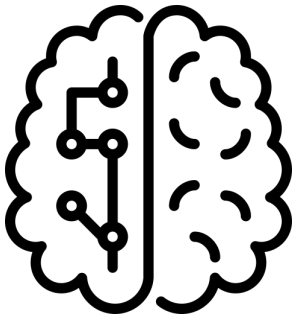
Seminario de Solución de Problemas de Inteligencia Artificial II

Ingeniería en Computación
Sección D05

Pre 1.3

Descenso del gradiente

Carrillo Partida Jorge Alberto
216439258
Jorge.carrillo4392@alumnos.udg.mx



Profesores
M. Diego Alberto Oliva Navarro
M. Diego Campos Peña

22 de Marzo de 2024

Descenso del gradiente

Introducción

En este proyecto se puso a prueba el descenso del gradiente para hallar el mínimo de una compleja función matemática, utilizando una metodología que combina la teoría matemática con la programación en Python. A lo largo de este proyecto, investigamos cómo la modificación de los parámetros del modelo puede influir en el rendimiento y la efectividad del algoritmo para encontrar soluciones óptimas, lo que nos permitió entender mejor las dinámicas que operan en los algoritmos de optimización empleados en inteligencia artificial.

Descenso del gradiente

Es un algoritmo de optimización muy utilizado para encontrar los valores mínimos de una función, generalmente con el objetivo de minimizar una función de costo o pérdida en problemas de aprendizaje automático. Es fundamental en el entrenamiento de muchos tipos de modelos, incluidas las redes neuronales. El descenso del gradiente se basa en la observación de que si la función multivariable $F(x)$ es definida y diferenciable en una región, entonces$F(x)$ disminuye más rápido si uno se mueve desde xx en la dirección del gradiente negativo de F en x, $-\nabla F(x)$. En otras palabras, para minimizar F, se debe mover en la dirección opuesta al gradiente.

El algoritmo toma pasos iterativos proporcionales al negativo del gradiente (o del vector de derivadas parciales) de la función en el punto actual. Si ηη es la tasa de aprendizaje, que determina el tamaño del paso, el algoritmo se actualiza según la fórmula:

$$x_{nuevo} = x_{viejo} - \eta \nabla F(x_{viejo})$$

La tasa de aprendizaje η es un hiperparámetro crucial que determina qué tan grande será el paso en cada iteración. Si η es muy grande, el algoritmo puede sobrepasar el mínimo y diverger. Si η es demasiado pequeño, el algoritmo se vuelve lento y puede atascarse antes de alcanzar el mínimo.

Desarrollo

Código en Python:

```python
# Universidad de Guadalajara
# Centro Universitario de Ciencias Exactas e Ingenierías
# División de Tecnologías para la Integración Ciber-Humana
#
# Seminario de Solución de Problemas de Inteligencia Artificial II
# Ingenieria en Computación
# Sección D05
#
# Pre 1.3 - Descenso del gradiente
#
# Jorge Alberto Carrillo Partida / 216439258 / jorge.carrillo4392@alumnos.udg.mx
#
# M. Diego Campos - 22 de Marzo de 2024

# ==============================================================================

# Librerías necesarias para el funcionamiento del programa
import numpy as np
import matplotlib.pyplot as plt

# Definimos la función objetivo que queremos minimizar
def f(x1, x2):
    return 10 - np.exp(-(x1**2 + 3*x2**2))

# Calculamos el gradiente de la función en un punto dado
def gradient_f(x1, x2):
    # Derivadas parciales respecto a x1 y x2
    df_dx1 = 2 * x1 * np.exp(-(x1**2 + 3*x2**2))
    df_dx2 = 6 * x2 * np.exp(-(x1**2 + 3*x2**2))
    return np.array([df_dx1, df_dx2])

# Implementa el algoritmo de descenso del gradiente para encontrar el mínimo de la función
def gradient_descent(starting_point, learning_rate, iterations):
    x = starting_point
    trajectory = [x]
    values = [f(x[0], x[1])]

    for _ in range(iterations):
        grad = gradient_f(x[0], x[1])
        x = x - learning_rate * grad
        trajectory.append(x)
        values.append(f(x[0], x[1]))

    return x, values, trajectory

# Parámetros iniciales
```

```
learning_rate = 0.01
iterations = 100
starting_point = np.random.uniform(-1, 1, 2)

# Ejecutando el descenso del gradiente
optimal_x, values, trajectory = gradient_descent(starting_point, learning_rate, iterations)

# Mostrando los resultados
print(f"Valores óptimos de x1, x2: {optimal_x}")
print(f"Valor final de la función: {values[-1]}")

# Graficando la convergencia del valor de la función
plt.figure(figsize=(10, 6))
plt.plot(values)
plt.title("Convergencia del valor de la función")
plt.xlabel("Iteración")
plt.ylabel("Valor de la función")
plt.grid(True)
plt.show()
```

Código en partes:

1. Librerías

```
# Librerías necesarias para el funcionamiento del programa
import numpy as np
import matplotlib.pyplot as plt
```

Este programa se construye mediante el uso de la biblioteca ***numpy*** para cálculos numéricos y ***matplotlib.pyplot*** para lla creación de gráficos en Python, esto nos ayuda a la visualización de manera gráfica..

2. Definición de funciones para el gradiente

```
# Definimos la función objetivo que queremos minimizar
def f(x1, x2):
    return 10 - np.exp(-(x1**2 + 3*x2**2))

# Calculamos el gradiente de la función en un punto dado
def gradient_f(x1, x2):
    # Derivadas parciales respecto a x1 y x2
    df_dx1 = 2 * x1 * np.exp(-(x1**2 + 3*x2**2))
    df_dx2 = 6 * x2 * np.exp(-(x1**2 + 3*x2**2))
    return np.array([df_dx1, df_dx2])

# Implementa el algoritmo de descenso del gradiente para encontrar el mínimo de la función
def gradient_descent(starting_point, learning_rate, iterations):
    x = starting_point
    trajectory = [x]
    values = [f(x[0], x[1])]

    for _ in range(iterations):
        grad = gradient_f(x[0], x[1])
        x = x - learning_rate * grad
        trajectory.append(x)
        values.append(f(x[0], x[1]))

    return x, values, trajectory
```

La función **f**, definida en el código, es la función objetivo que se desea minimizar. Esta función toma dos variables, **x1** y **x2**, y calcula el valor de $10-e-(x12+3x22)10-e-(x12+3x22)$. Esta fórmula está diseñada para tener un mínimo, que el algoritmo de descenso del gradiente intentará encontrar. Para optimizar esta función, es necesario calcular su gradiente, que es lo que hace la función **gradient_f**. Esta función devuelve un array con las derivadas parciales de la función **f** con respecto a **x1** y **x2**. Las expresiones para estas derivadas, **2 * x1 * exp(-(x1**2 + 3*x2**2))** y **6 * x2 * exp(-(x1**2 + 3*x2**2))**, indican cómo cambia la función en relación con pequeños cambios en **x1** y **x2**, respectivamente.

El núcleo del código es la función **gradient_descent**, que implementa el algoritmo de descenso del gradiente. Este algoritmo comienza en un punto inicial y, iterativamente, hace ajustes a este punto en la dirección que más reduce el valor de la función **f**. Estos ajustes se calculan como el producto del gradiente de la función en el punto actual y la tasa de aprendizaje, que determina el tamaño del paso. El proceso se itera un número definido de veces. Dentro del ciclo de descenso del gradiente, se almacenan tanto la trayectoria (cada posición sucesiva de **x**) como los valores correspondientes de la función en la lista **values**. Esto no solo ayuda a monitorear la progresión del algoritmo, sino que también permite visualizar cómo el valor de la función se acerca al mínimo.

3. Definición de parametros y resultados

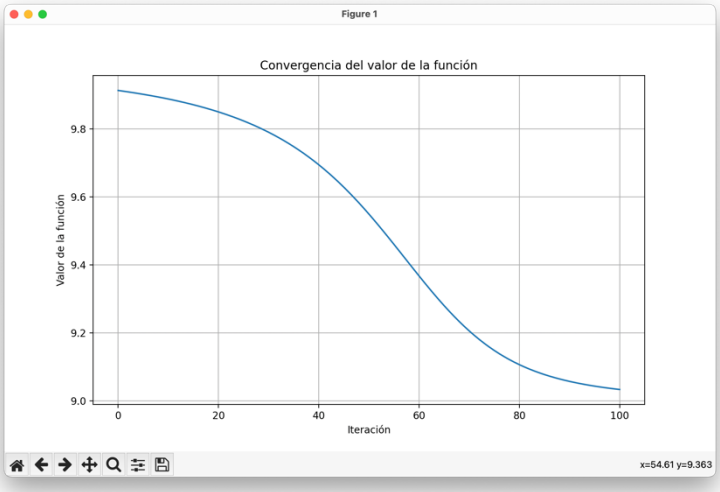
```python
# Parámetros iniciales
learning_rate = 0.01
iterations = 100
starting_point = np.random.uniform(-1, 1, 2)

# Ejecutando el descenso del gradiente
optimal_x, values, trajectory = gradient_descent(starting_point, learning_rate, iterations)

# Mostrando los resultados
print(f"Valores óptimos de x1, x2: {optimal_x}")
print(f"Valor final de la función: {values[-1]}")

# Graficando la convergencia del valor de la función
plt.figure(figsize=(10, 6))
plt.plot(values)
plt.title("Convergencia del valor de la función")
plt.xlabel("Iteración")
plt.ylabel("Valor de la función")
plt.grid(True)
plt.show()
```

Finalmente, el código ejecuta la función **gradient_descent** con valores iniciales y muestra los resultados en la consola. También genera un gráfico que ilustra cómo el valor de la función decrece con cada iteración, visualizando la convergencia del algoritmo hacia el mínimo de la función.

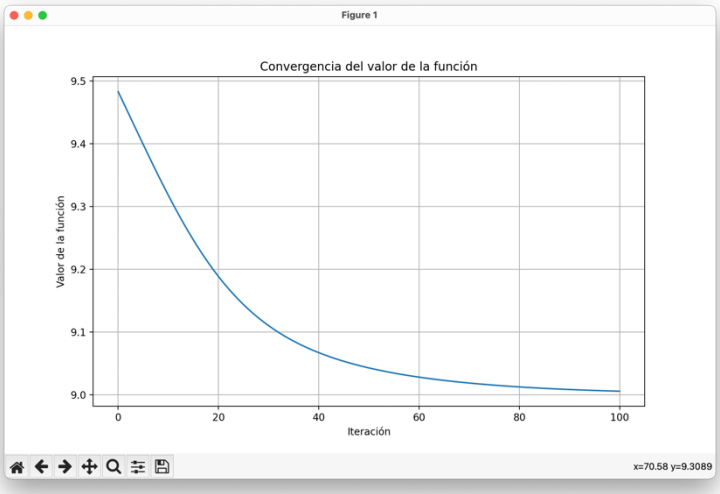
Resultados

Los resultados muestran que el algoritmo de descenso del gradiente ha optimizado eficazmente la función objetivo. Los valores finales de x1*x*1 y x2*x*2 cercanos a cero indican que el algoritmo ha encontrado un punto cerca del mínimo de la función en el rango dado [−1,1][−1,1]. La convergencia de la función se aproxima al valor teórico mínimo de 9 basado en la formulación de la función, confirma que el descenso del gradiente ha funcionado como se esperaba.

```
Pre1.3-Descenso_del_gradiente — -zsh — 89×6
jorgecap@Monster-2 Pre1.3-Descenso_del_gradiente % python Pre1.3-DescensoGradiente.py
Valores óptimos de x1, x2: [0.17065657 0.04089299]
Valor final de la función: 9.033564170834078
jorgecap@Monster-2 Pre1.3-Descenso_del_gradiente %
```




```
Pre1.3-Descenso_del_gradiente — -zsh — 89×5
jorgecap@Monster-2 Pre1.3-Descenso_del_gradiente % python Pre1.3-DescensoGradiente.py
Valores óptimos de x1, x2: [-0.07470879 -0.00154808]
Valor final de la función: 9.005573005914234
jorgecap@Monster-2 Pre1.3-Descenso_del_gradiente %
jorgecap@Monster-2 Pre1.3-Descenso_del_gradiente %
```



Conclusión

Esta actividad ha demostrado claramente la eficiencia del algoritmo de descenso del gradiente para optimizar funciones matemáticas complejas. Gracias a la implementación en Python y al uso de herramientas analíticas como NumPy y Matplotlib, no solo pudimos visualizar sino también medir el proceso de optimización mediante numerosas iteraciones y ajustes de parámetros. Los resultados verifican que el descenso del gradiente es una herramienta sólida y adaptable, ideal para abordar desafíos de optimización en el ámbito de la inteligencia artificial, ofreciendo soluciones tanto eficientes como efectivas. Este ejercicio ha reforzado la relevancia de las técnicas de aprendizaje automático en contextos prácticos.