SEMINARIO DE SOLUCION DE PROBLEMAS DE INTELIGENCIA ARTIFICIAL 2

Practica 1. Pre Ejercicio 3

Diego Campos Peña

26/02/2024

Portugal Pacheco, Juan Carlos
219747433
ING. COMPUTACION

Introducción:

El gradiente descendente es uno de los algoritmos fundamentales en el campo del aprendizaje automático y la optimización numérica. Se utiliza ampliamente para encontrar los mínimos (o máximos) de una función objetivo, lo que lo convierte en una herramienta esencial en problemas como la optimización de modelos de aprendizaje automático, la regresión, la clasificación y más. Su concepto central se basa en la idea de ajustar iterativamente los parámetros de un modelo o función para minimizar el error o la pérdida

Desarrollo:

En esencia, el gradiente descendente opera calculando el gradiente (derivada) de la función objetivo con respecto a los parámetros del modelo y luego ajustando estos parámetros en la dirección opuesta al gradiente. Esto significa que el algoritmo "desciende" por la pendiente de la función objetivo en busca del mínimo. El proceso se repite en ciclos, y la tasa de aprendizaje (learning rate) controla el tamaño de los pasos que el algoritmo toma en cada iteración. Para el desarrollo visual de este algoritmo lo implementaremos en base a la siguiente función para que nuestro algoritmo sea capaz de optimizar esta función:

$$f(x_1, x_2) = 10 - e^{-(x_1^2 + 3x_2^2)}$$

Código:

```python
import numpy as np
import matplotlib.pyplot as plt

def F(x1, x2):
 return 10- np.exp(-((x1**2) + (x2**2)))

def G(x, y):
 return np.array([(1- 2*(x**2))*np.exp(-x**2-y**2),-2*x*y*np.exp(-x**2-y**2)])

def gradient_descent(lr, max_iterations):
 x1 = np.random.uniform(-1, 1)
 x2 = np.random.uniform(-1, 1)
 error1 = []
 for i in range(max_iterations):
 grad = G(x1, x2)
 x1-= lr * grad[0]
 x2-= lr * grad[1]
 x1 = max(-1, min(x1, 1))
 x2 = max(-1, min(x2, 1))
 error = F(x1, x2)
 error1.append(error)
 if i % 100 == 0:
     print(f"Iteracion numero: {i}: F({x1}, {x2}) = {error}")
 return x1, x2, errors

lr = 0.01
```

```
max_iterations = 1000

opt_x1, opt_x2, errors = gradient_descent(lr, max_iterations)

plt.plot(range(len(errors)), errors)
plt.xlabel('Numero de iteraciones')
plt.ylabel('Error')
plt.title('Convergencia del error durante el descenso del gradiente')
plt.show()

print("\n\n\n")
print(f"Optimizacion evaluada para: x1 = {opt_x1}, x2 = {opt_x2}")
print(f"Optimizacion para la funcion F(x1, x2) = {F(opt_x1, opt_x2)}")
```

Resultado:

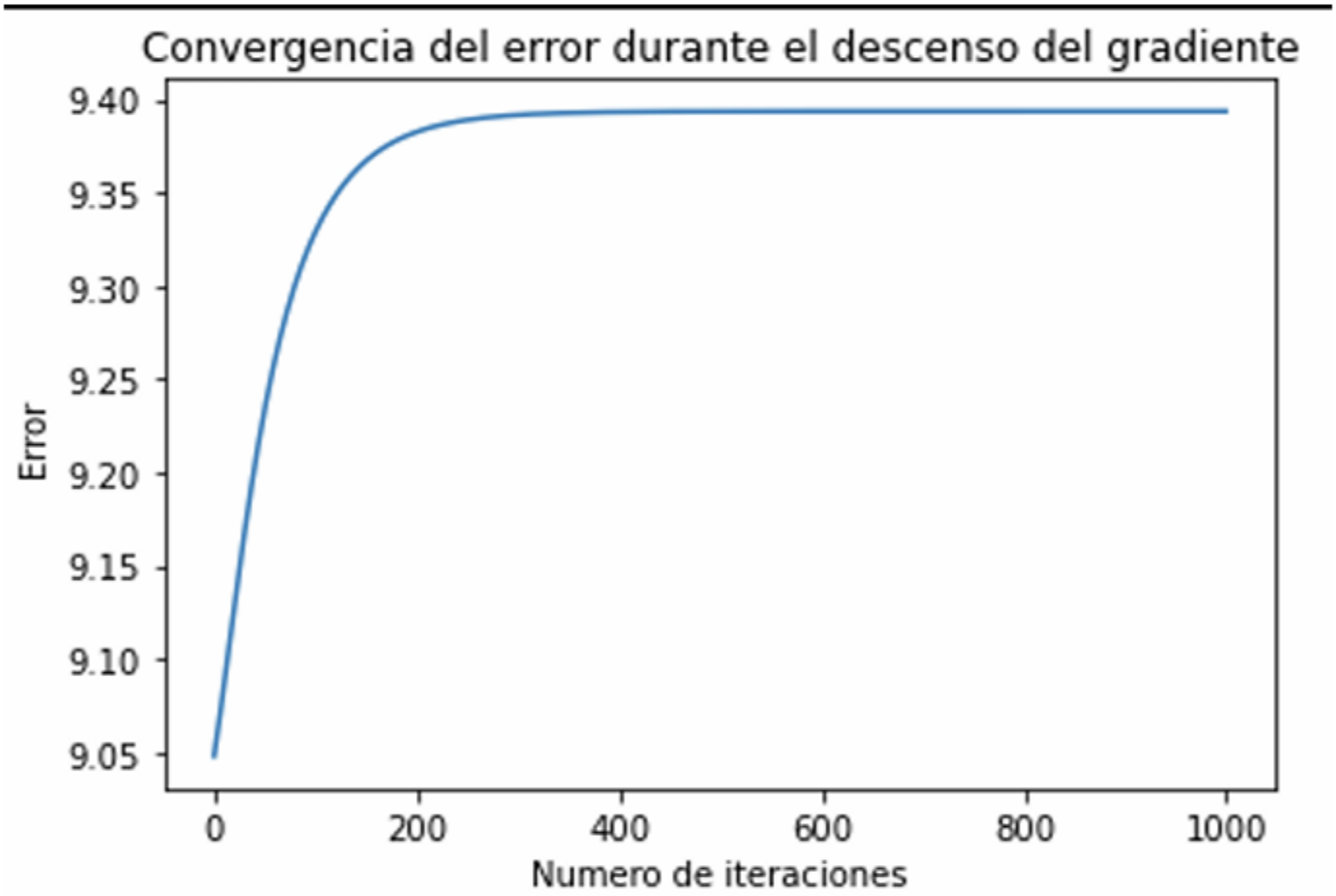
Se evalúa la función en base a N número de iteraciones:

```
In [6]: runfile('C:/Users/emman_pup4rlf/Desktop/prueba1.py',
wdir='C:/Users/emman_pup4rlf/Desktop')
Iteracion numero: 0: F(-0.21624746842544967,
-0.04828965195802658) = 9.047909188097842
Iteracion numero: 100: F(-0.631014373899287,
-0.023033106667262295) = 9.328814457654124
Iteracion numero: 200: F(-0.69416120539854,
-0.009762235800556244) = 9.382425197600927
Iteracion numero: 300: F(-0.7048297574002726,
-0.004125468972005568) = 9.391526554273332
Iteracion numero: 400: F(-0.7067038036053206,
-0.0017432051974096447) = 9.393125524860496
Iteracion numero: 500: F(-0.7070353859175132,
-0.0007365802983717412) = 9.393408429110911
Iteracion numero: 600: F(-0.7070941296792148,
-0.00031123698530805433) = 9.393458547036028
Iteracion numero: 700: F(-0.7071045392152582,
-0.0001315110423663011) = 9.393467427695438
Iteracion numero: 800: F(-0.707106383884806,
-5.556908242517223e-05) = 9.393469001369201
Iteracion numero: 900: F(-0.7071067107802741,
-2.3480331748220155e-05) = 9.393469280229803
```

Mejor solución para x1 y x2:

```
Optimizacion evaluada para: x1 = -0.7071067684919945, x2 = -1.0007290826856647e-05
Optimizacion para la funcion F(x1, x2) = 9.39346932945918
```

Gráfica de la convergencia del error:



Conclusión:

En esta práctica apreciamos el comportamiento del algoritmo descenso del gradiente en base a una función numérica para visualizar su optimización para encontrar los mínimos o máximos de la función, el ver el entrenamiento de este algoritmo en funciones aplicadas como en base a entrenamiento estas funciones obtienen su aprendizaje automático y esto trae como resultado la obtención de búsqueda eficiente.

Bibliografía:

● Repetur, A. E. (2019). Redes neuronales artificiales. Buenos Aires: Universidad Nacional del Centro de la Provincia de Buenos Aires.

● Larranaga, P., Inza, I., & Moujahid, A. (1997). Tema 8. redes neuronales. Redes Neuronales, U. del P. Vasco, 12, 17.

● Munt,A.M.(2018). Introducción a los modelos de redes neuronales artificiales el perceptrón simple y multicapa. Universidad de Zaragoza, España