UNIVERSIDAD DE GUADALAJARA

Centro Universitario de Ciencias Exactas

Inteligencia Artificial 2

Sección D04

Tarea 3:

Neurona Lineal

219747824 - Jesús Eduardo Quintero Gómez
Ingeniería en Computación

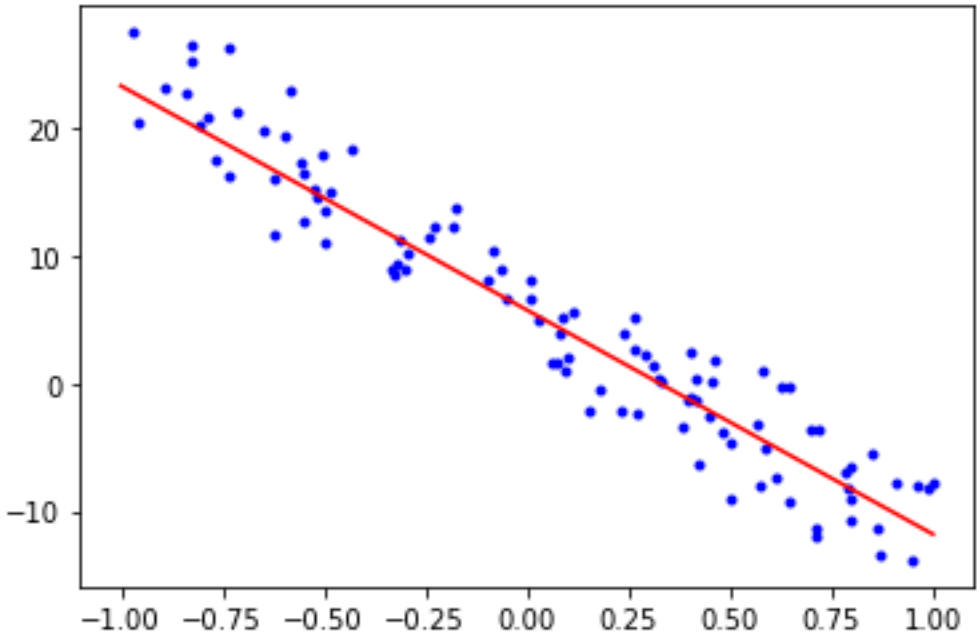
Maestro Carlos Alberto Villaseñor Padilla

2 de diciembre de 2023

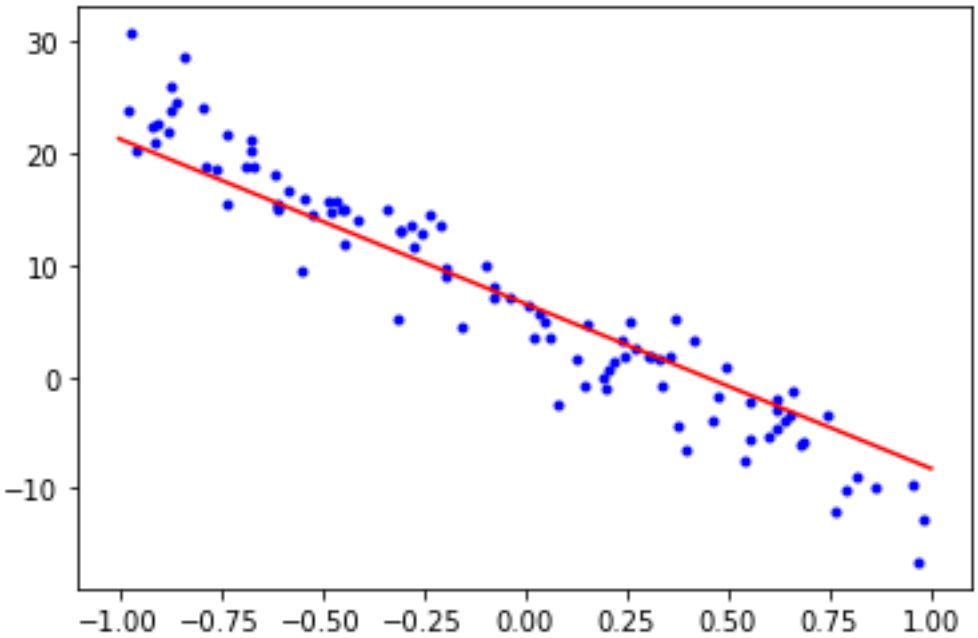
Reporte

Durante el desarrollo de esta actividad se implementaron mejoras al perceptrón para que pudiera hacer un mayor número de cálculos al trabajar con matrices. Al final se obtuvo una gran mejora respecto a los tiempos de ejecución del programa. Además, se implementaron 3 métodos vistos durante la clase, los cuales pueden ser útiles para resolver distintos tipos de problemas.

SGD



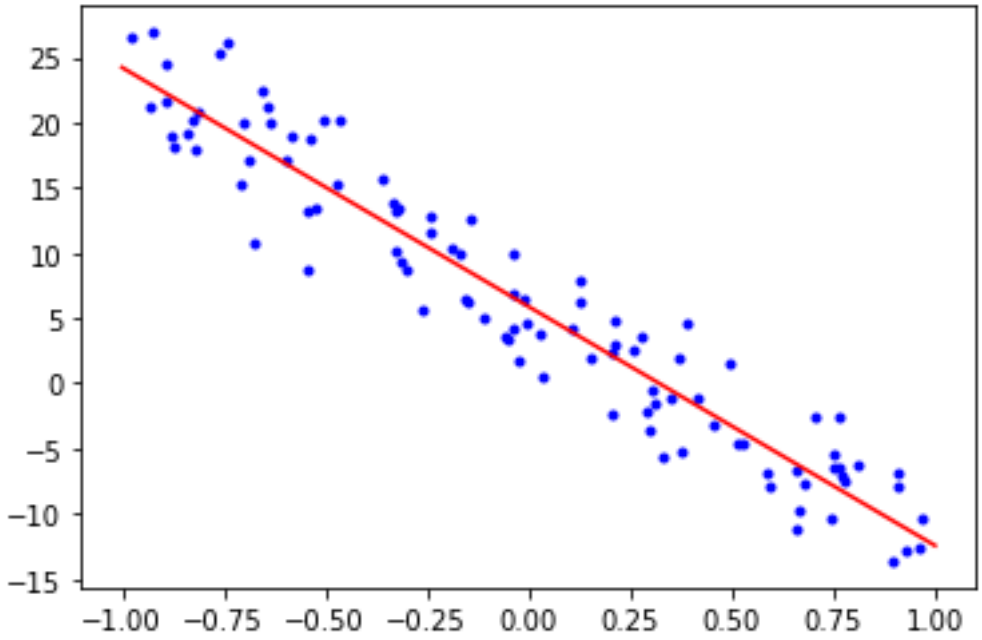
BGD



Seminario de Solución de Problemas de Inteligencia Artificial 2

Jesús Eduardo Quintero Gómez

Pseudoinversa



Código

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

class LinearNeuron:
    def __init__(self, n_input):
        self.w= -1 + 2 * np.random.rand(n_input)
        self.b = -1 + 2 * np.random.rand()

    def predict(self, X):
        Yest = np.dot (self.w, X) + self.b
        return Yest
    def fit(self, X, Y, epochs=50, lr=0.1, solver='BGD'):
        p = X.shape[1]
        print(p)
        if solver == 'SGD': #Stochastic Gradient Descent
            for _ in range(epochs):
                for i in range(p):
                    yest = self.predict(X[:,i])
                    self.w += lr * (Y[:, i] - yest)* X[:,i]
                    self.b += lr * (Y[:, i] - yest)
        elif solver == 'BGD': # Batch Gradient Descent
            for _ in range(epochs):
                Yest = self.predict(X)
                self.w += (lr/p) * ((Y-Yest) @ X.T).ravel()
                self.b += (lr/p) * np.sum (Y-Yest)
        else: # Direct (Pseudoinverse)
            Xhat = np.concatenate ((np.ones((1,p)), X), axis=0)
            what = Y.reshape(1,-1) @ np.linalg.pinv(Xhat)
            self.w = what [0,1:]
            self.b = what [0,0]
p = 100
x = -1 + 2*np.random.rand(p).reshape(1,-1)
y = -18 * x + 6 + 3 *np.random.randn(p)
neuron = LinearNeuron (1)
neuron.fit(x,y, solver='Pseudoinversa')
# Dibujo
plt.plot(x, y, '.b')
xnew = np.array([[-1, 1]])
ynew = neuron.predict(xnew)
plt.plot(xnew.ravel(), ynew, '-r')
```