Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías
Inteligencia Artificial II

Felipe Alejandro Jimenez Castillo
215461386

Neurona Lineal

Una neurona lineal es un tipo de neurona artificial en la que la salida es una función lineal de la suma ponderada de las entradas. En el contexto del aprendizaje automático, las neuronas lineales se utilizan a menudo como capas de entrada o capas de salida en redes neuronales. También se pueden usar como capas ocultas, aunque en general, las capas no lineales son más efectivas para capturar relaciones complejas entre las entradas y las salidas.

Es importante tener en cuenta que la neurona lineal es una de las funciones de activación más simples, y su salida no está limitada a un rango específico, lo que puede dar lugar a problemas de saturación si los pesos son demasiado grandes.

Función Objetivo

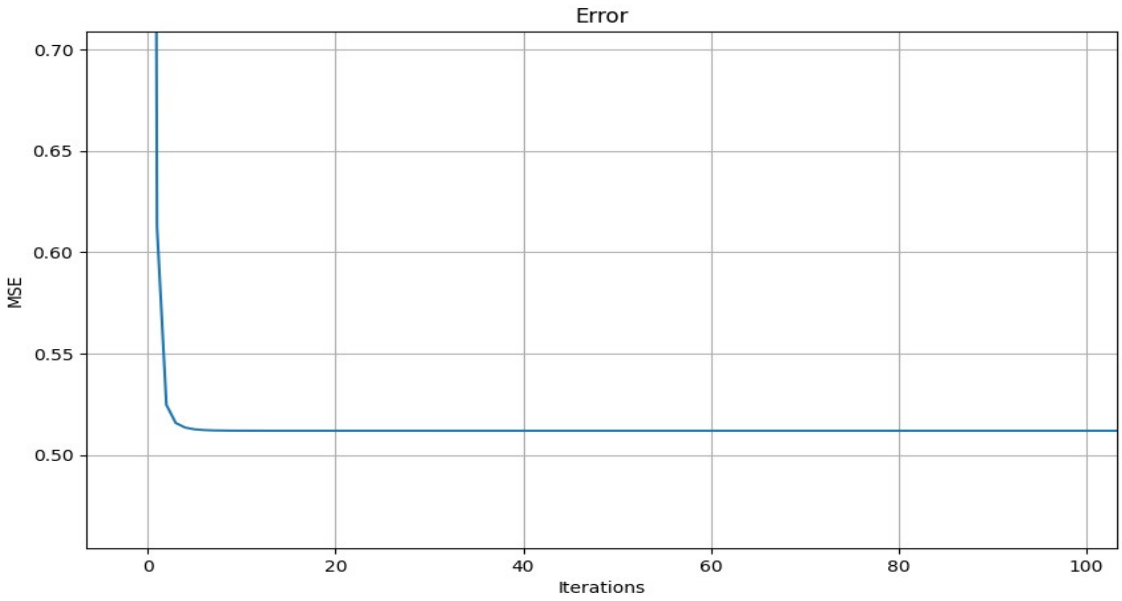
$f(x) = -x + 7 * N(0,1)$

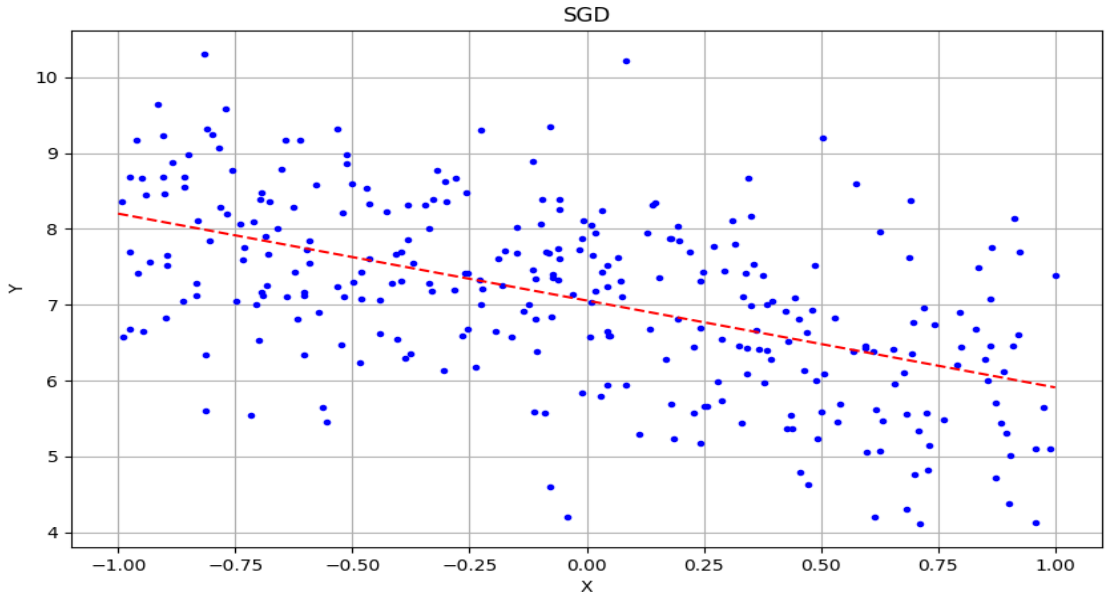
$\eta = 0.005$

$Epoch = 500$

Stochastic Gradient Descent

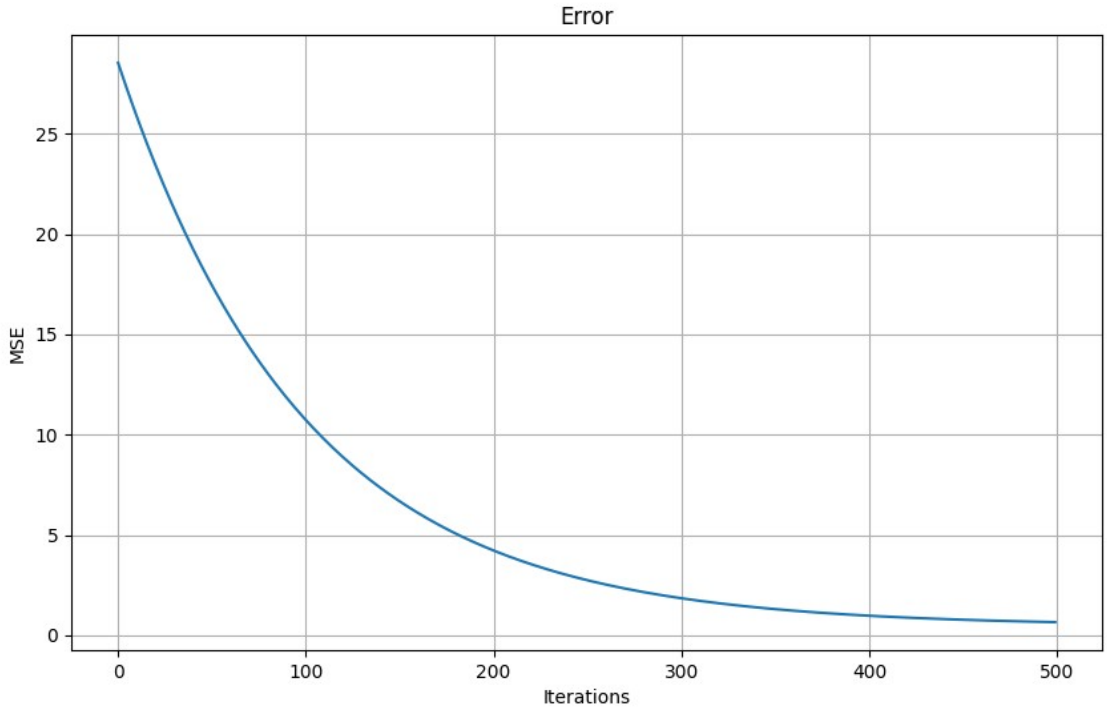
$w_0 = [0.3836763], w_f = [-1.14541754]$

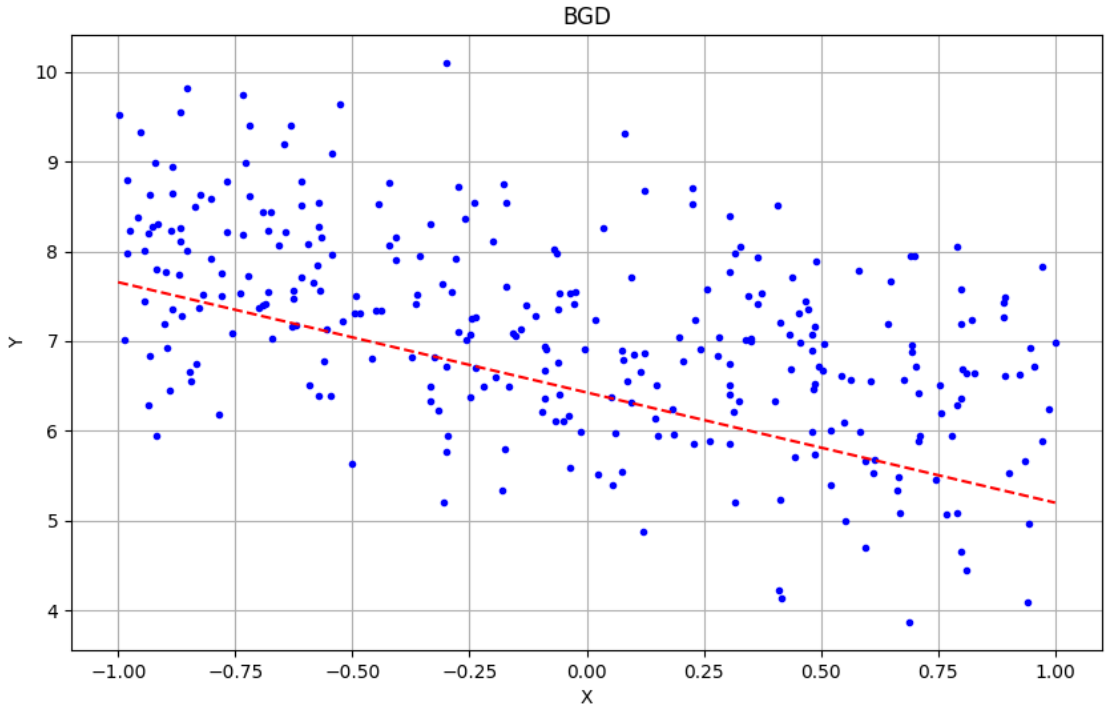




Batch Gradient Descent

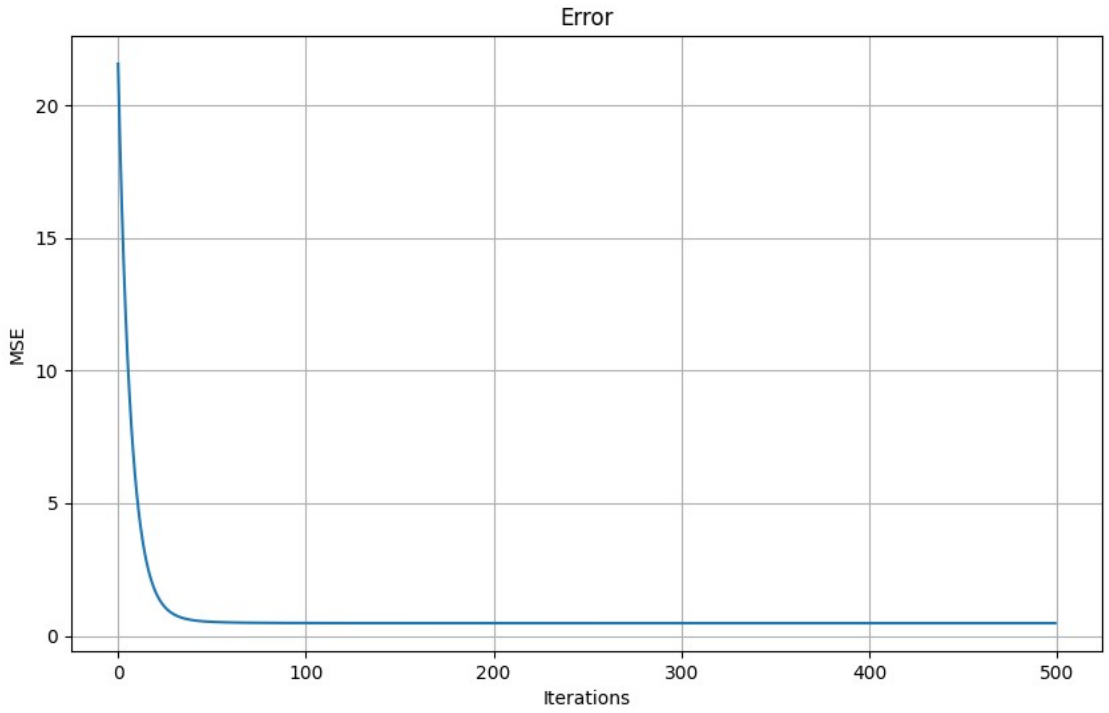
$w_0=[-0.96160127], w_f=[-1.22778493]$

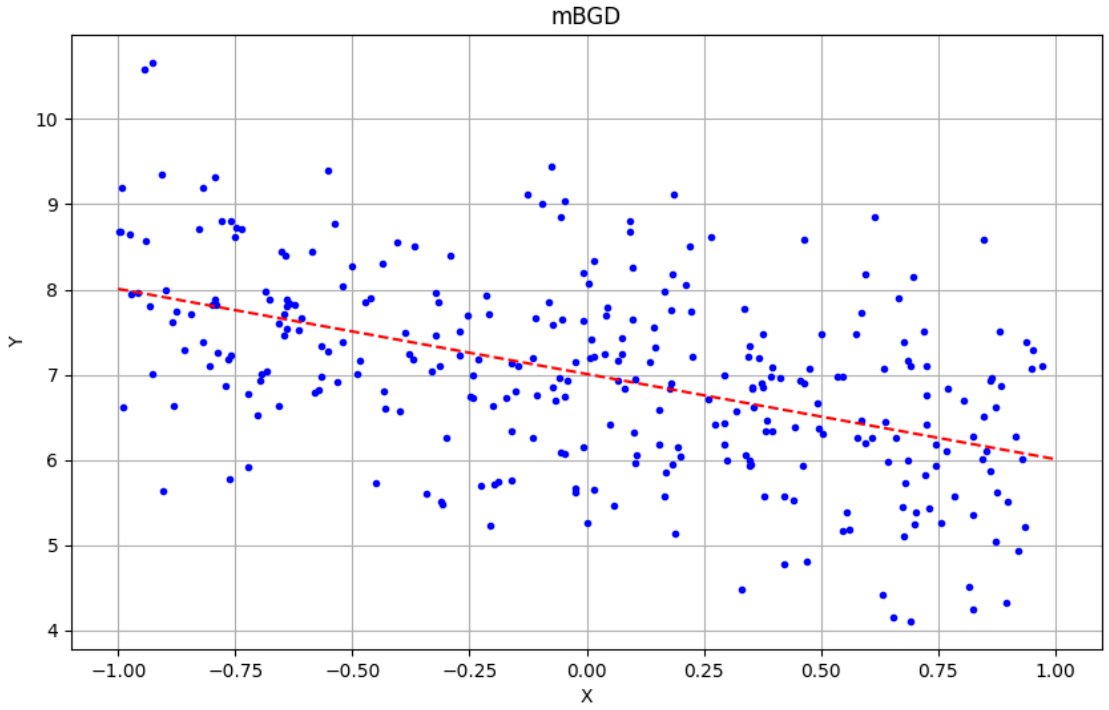




Mini Batch Gradient Descent

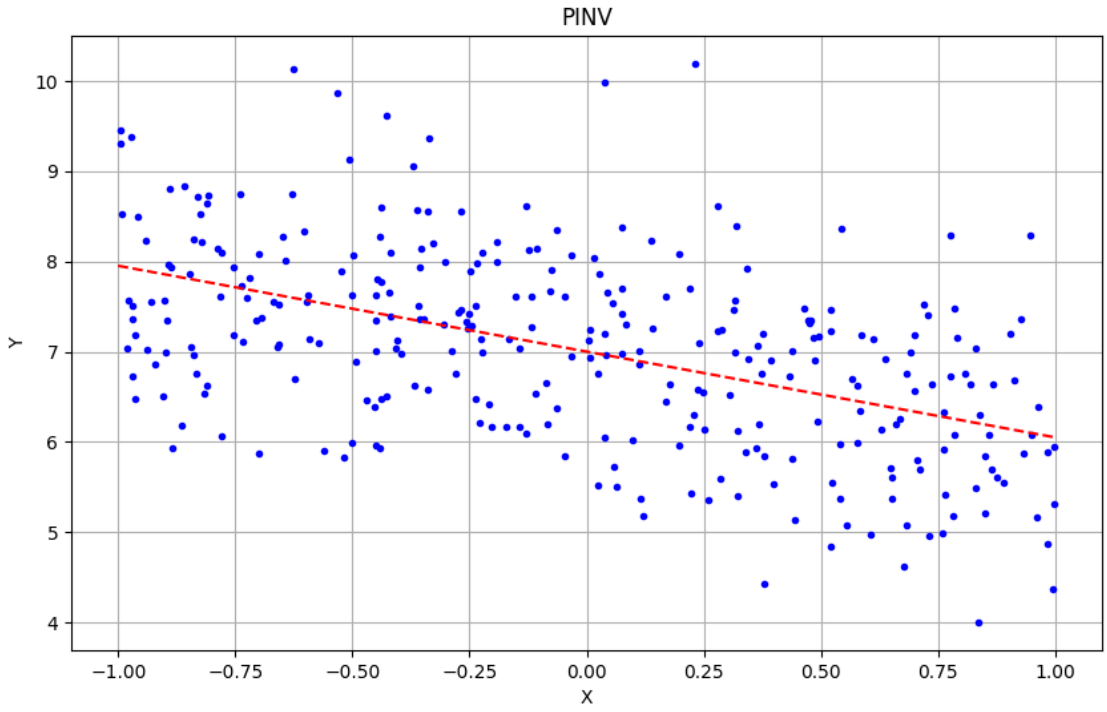
$w_0 = [0.38450942], w_f = [-0.99936729]$





Pseudo Inverse

$w_0 = [0.27888098], w_f = [-0.95101292]$



Conclusión

Podemos denotar, en las diferentes graficas, como cada formato reduce su error cuadrático medio respecto a las iteraciones, con estos datos podemos concluir que no solo importa el algoritmo, sino también los parámetros de ajuste y las épocas que iteraremos, siendo una correlación dependiente, con ellos veremos nuestros resultados mejorar u oscilar de ser mal ajustados dichos parámetros. Como podemos observar, la relación de tasa de aprendizaje y el algoritmo, es importante para objetivar la cantidad de épocas necesarias para una solución aceptable, ya que, como vimos en las graficas previas, podemos tener cientos de iteraciones sin mejoras en nuestro modelo.

Código

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

## Linear Neuron
class LinearNeuron:
	def __init__(self, n_inputs, learning_rate=0.1 ) -> None:
		self.w = -1 + 2 * np.random.rand(n_inputs)
		self.b = -1 + 2 * np.random.rand()
		self.eta = learning_rate

	def predict(self, X):
		Y_est = np.dot(self.w, X) + self.b
		return Y_est

	## Generator to get a batch
	def batcher(self, X, Y, batch_size):
		p = X.shape[1]
		li, lu = 0, batch_size
		while True:
			if li < p:
				yield X[:, li:lu], Y[:, li:lu]
				li, lu = li + batch_size, lu + batch_size
			else:
				return None

	## Medium Square Error
	def MSE(self, X, Y):
		p = X.shape[1]
		Y_est = self.predict(X)
		return (1/(2*p))*np.sum((Y-Y_est)**2)

	def fit(self, X, Y, solver='BGD', epochs=500, batch_size=20):
		error_history = []
		p = X.shape[1]

		# Stochastic Gradient Descent
		if solver == 'SGD':
			for _ in range(epochs):
				for i in range(p):
					y_est = self.predict(X[:, i])
					self.w += self.eta * (Y[:, i] - y_est) * X[:, i]
					self.b += self.eta * (Y[:, i] - y_est)
				error_history.append(self.MSE(X, Y))

		# Batch Gradient Descent
		elif solver == 'BGD':
			for _ in range(epochs):
				Y_est = self.predict(X)
				self.w += (self.eta/p) * ((Y - Y_est) @ X.T).ravel()
				self.b += (self.eta/p) * np.sum(Y - Y_est)
				error_history.append(self.MSE(X, Y))

		# Mini Batch Gradient Descent
		elif solver == 'mBGD':
			for _ in range(epochs):
				mini_batch = self.batcher(X, Y, batch_size)
```

```
                        for mX, mY in mini_batch:
                                p = mX.shape[1]
                                Y_est = self.predict(mX)
                                self.w += (self.eta/p) * ((mY - Y_est) @ mX.T).ravel()
                                self.b += (self.eta/p) * np.sum(mY - Y_est)
                        error_history.append(self.MSE(X, Y))

            # Pseudo-inverse (Direct Method)
            elif solver == "PINV":
                    X_hat = np.concatenate((np.ones((1, p)), X), axis=0)
                    w_hat = np.dot(Y, np.linalg.pinv(X_hat))
                    self.w = w_hat[0, 1:]
                    self.b = w_hat[0, 0]

            return error_history


if __name__=="__main__":
        p = 300
        x = -1 + 2 * np.random.rand(p).reshape(1, -1)
        y = ( -x + 19 - 12 ) + np.random.randn(1, p)

        solver = 'PINV' # ['SGD', 'BGD', 'mBGD', 'PINV']
        neuron = LinearNeuron(1, 0.005)
        print( neuron.w )
        error = neuron.fit(x, y, solver=solver)
        print( neuron.w )

        plt.figure(figsize=(10,6))
        plt.title(solver)
        plt.xlabel('X')
        plt.ylabel('Y')
        plt.grid(True)
        plt.plot(x, y, '.b')
        xn = np.array([[-1, 1]])
        plt.plot(xn.ravel(), neuron.predict(xn), '--r')
        ## MSE
        plt.figure(figsize=(10,6))
        plt.title("Error")
        plt.ylabel("MSE")
        plt.xlabel("Iterations")
        plt.grid(True)
        plt.plot(error)
        plt.show()
```