 **V**

Practica 5

11 de mayo del 2020

Julio Esteban Valdés López

Maestro:

Seminario de Resolución de Problemas de Inteligencia Artificial 2

Materia:

Carrera:

Ingeniería en Computación

218744996

Código:

Flores Moreno Angel

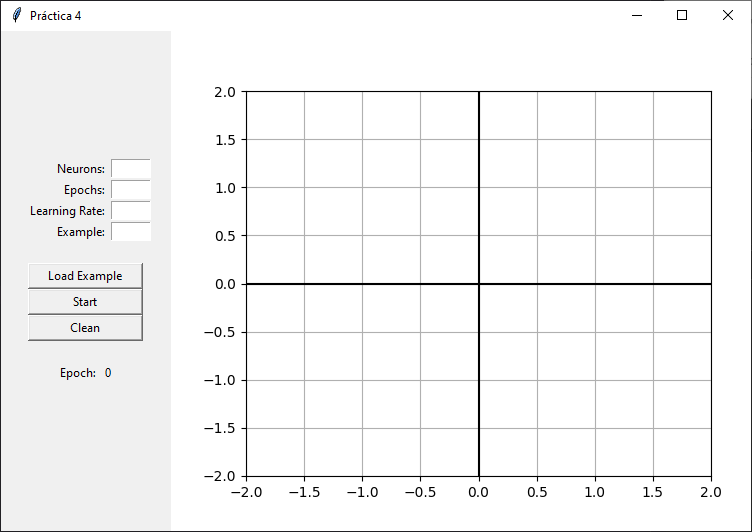
Nombre:

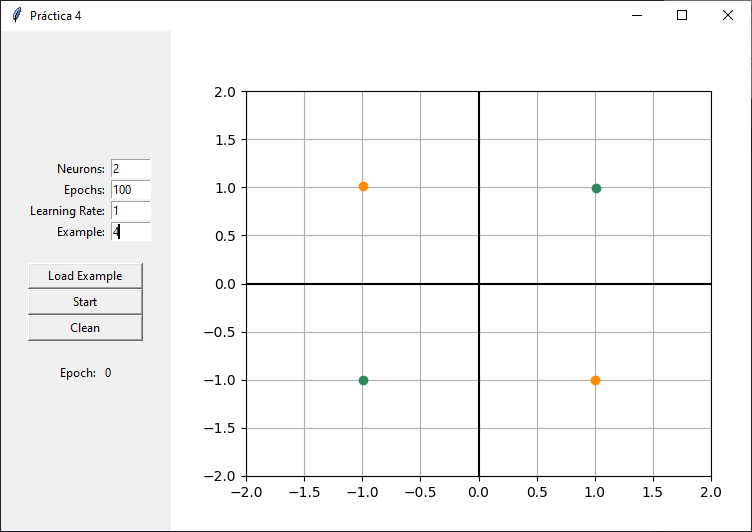
Centro Universitario de Ciencias Exactas e Ingeniería

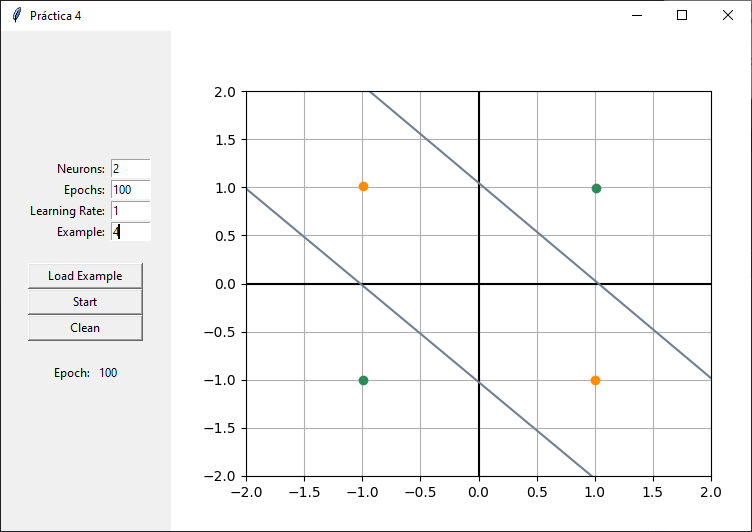
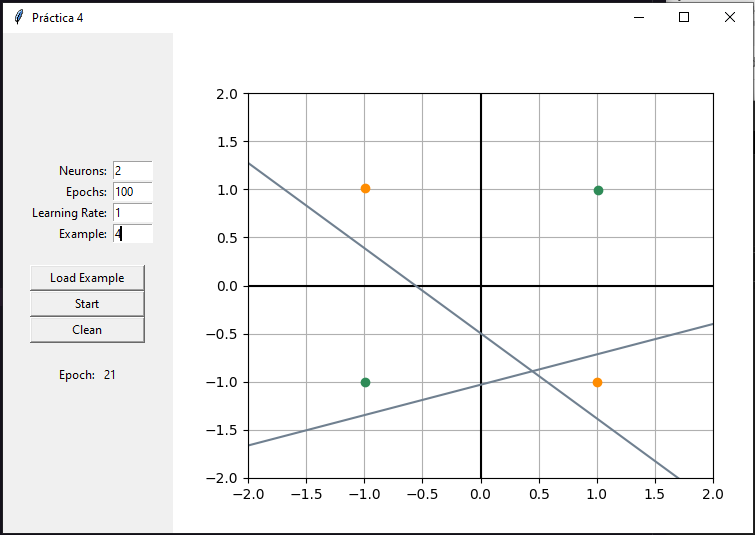
# Introducción

En esta práctica se llevó a cabo la realización de un Adaline multicapa con varias neuronas en la capa oculta y una sola neurona en la capa de salida, la cual deba ser capaz de separar diferentes puntos ingresados por el usuario ya clasificados, con 2 clases de por medio dividiéndolos por más de una línea, ya que el problema no es linealmente separable, al principio, muy probablemente las clases estén erróneamente separadas, por lo que el Adaline irá actualizando sus pesos de tal manera que al final del entrenamiento, las línea separe correctamente los puntos dentro de la gráfica lo más intermediamente posible, donde se irá reduciendo un error cuadrático promedio de los puntos ingresados, este proceso solo se encarga de reducir el error más no de eliminarlo completamente puesto que es muy improbable que esto suceda.

# Desarrollo







# Conclusiones

Gracias al uso del Adaline multicapa de esta práctica mejoramos las tareas de clasificación de la practica pasada, puesto que en este caso, tenemos problemas que no son linealmente separables, y nuestros puntos en la gráfica ya están clasificados antes de que el Adaline se ponga a realizar su tarea, haciendo que el Adaline aprenda hasta que las líneas separe las clases de manera correcta totalmente en medio de los puntos a clasificar sin modificar los puntos a placer, este Adaline nos ayuda a entender otro tipo de algoritmos de aprendizaje.

# Código Fuente

import numpy as np

class Adaline:

    def \_\_init\_\_(*self*, *lr*, *inputs*=3, *outputLayer*=False):

        self.lr = lr

        self.inputs = inputs

        self.outputLayer = outputLayer

        self.w = np.zeros(inputs)

        for i in range(self.inputs):

            self.w[i] = np.random.uniform(-2, 2)

        self.y = np.zeros(0)

        self.localGradient = 0

    def guess(*self*, *p*):

        return self.sigmoid(np.dot(p, self.w))

    def sigmoid(*self*, *x*, *derivative*=False):

        if derivative == True:

            return x \* (1 - x)

        return 1 / (1 + np.exp(-x))

    def getOutput(*self*, *p*):

        self.y = self.guess(p)

    def backPropagation(*self*, *prevLayerY*=None, *nextLayer*=None, *pointY*=None):

        if self.outputLayer == True:

            self.localGradient = self.localGradientFunc(*pointY*=pointY)

        else:

            self.localGradient = self.localGradientFunc(

*pointY*=pointY, *nextLayer*=nextLayer

            )

        for i in range(self.inputs):

            self.w[i] += self.localGradient \* prevLayerY[i] \* self.lr

    def localGradientFunc(*self*, *pointY*, *nextLayer*=None):

        sigmoid = self.sigmoid(self.y, True)

        if self.outputLayer == True:

            error = pointY - self.y

            return sigmoid \* error

        wSum = sum(nextLayer.w)

        return sigmoid \* wSum \* nextLayer.localGradient

neurons = int(self.entries[0].get())

        totalEpochs = int(self.entries[1].get())

        lr = float(self.entries[2].get())

        epoch = 0

        hiddenLayer = np.array([])

        outputLayer = Adaline(*lr*=lr, *inputs*=neurons + 1, *outputLayer*=True)

        for i in range(neurons):

            hiddenLayer = np.append(hiddenLayer, Adaline(*lr*=lr))

        grid = np.zeros((neurons,len(self.inputs)))

        while epoch <= totalEpochs:

            self.window.update()

            self.epochLabel.config(*text*=epoch)

            self.configGraph()

            for i in range(len(self.points)):

                [layer.getOutput(self.points[i]) for layer in hiddenLayer]

                outputLayer.getOutput(np.array([1] + [n.y for n in hiddenLayer]))

                outputLayer.backPropagation(

*prevLayerY*=[1] + [n.y for n in hiddenLayer], *pointY*=self.pointsY[i]

                )

                [

                    layer.backPropagation(

*prevLayerY*=self.points[i], *nextLayer*=outputLayer

                    )

                    for layer in hiddenLayer

                ]

            for neuron in hiddenLayer:

                self.graph.plot(

                    [-2, 2], [guessY(-2, neuron.w), guessY(2, neuron.w)], *c*="slategrey"

                )

            for i in range(len(self.points)):

                self.graph.plot(

                    self.points[i][1],

                    self.points[i][2],

*marker*="o",

*c*=pointColor(self.pointsY[i]),

                )

            epoch += 1

            self.canvas.draw()