|  |  |
| --- | --- |
| IA 2  Muñiz Ramos Oswaldo De Jesús  Reynaldo Javier Salazar Ochoa | Practica #1  Adaline  Sección D01 |

Adaline

El término Adaline es una sigla sin embargo su significado cambio ligeramente a finales de loa años sesenta cuando decayó el estudio de las redes neuronales, inicialmente se llamaba ADAptative Linear Neuron (Neuron Lineal Adaptativa), para pasar después a ser Adaptative Linear Element(Elemento Lineal Adaptativo),este cambio se debió a que la Adaline es un dispositivo que consta de un único elemento de procesamiento, como tal no es técnicamente una red neuronal.

import sys

from matplotlib.figure import Figure

from matplotlib.backend\_bases import key\_press\_handler

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk

from Adaline import \*

from AdalineGD import \*

import numpy as np

import time

import matplotlib

matplotlib.use('TkAgg')

if sys.version\_info[0] < 3:

    import Tkinter as Tk

else:

    import tkinter as Tk

import math

import random as rnd

root = Tk.Tk()

root.wm\_title("Adaline")

f = Figure(figsize=(9, 6), dpi=100)

a = f.add\_subplot(211)

a.set\_xlim([-1, 1])

a.set\_ylim([-1, 1])

a.grid(True)

err = f.add\_subplot(212)

err.set\_title('Error acumulado')

err.set\_ylabel('Error', Fontsize=12)

err.set\_xlabel('Épocas', Fontsize=12)

canvas = FigureCanvasTkAgg(f, master=root)

canvas.get\_tk\_widget().pack(side=Tk.TOP, fill=Tk.BOTH, expand=1)

toolbar = NavigationToolbar2Tk(canvas, root)

toolbar.update()

canvas.\_tkcanvas.pack(side=Tk.TOP, fill=Tk.BOTH, expand=1)

lines = []

colors = ['r', 'g']

def newLine(w):

    m = -w[0]/w[2]

    b = -w[1]/w[3]

    return [m, b]

def clear(i):

    global lines

    if i < len(lines):

        l = lines.pop(i)

        l.remove()

        del l

def plot(weights):

    global lines

    [m, b] = newLine(weights)

    clear(0)

    lines = a.plot(

        [ -1, 0, 1],

        [ (m\*-1)+b, (m\*0)+b, (m\*1)+b],

        'k'

    )

    canvas.draw()

if \_\_name\_\_ == '\_\_main\_\_':

    xi = []

    d = []

    adalineTrained = False

    lr = 0.1

    epochs = 500

    weights = []

    precision = 0.001

    w\_ajustado = []

    n\_muestras = len(d)

    def clicked(event):

        global n\_muestras

        if event.xdata and event.ydata and not adalineTrained and event.inaxes == a:

            desiredClass = -1

            if event.button == 3:

                desiredClass = 1

            xi.append([event.xdata, event.ydata,1])

            d.append(desiredClass)

            color = ''

            if desiredClass == 1:

                color = 'g'

            else:

                color = 'r'

            a.scatter(event.xdata, event.ydata, c=color)

            n\_muestras = len(d)

            canvas.draw()

    f.canvas.callbacks.connect('button\_press\_event', clicked)

    canvas.draw()

    def learningRate(event):

        global lr

        try:

            lr = float(lrEntry.get())

        except ValueError:

            print('El numero ingresado no es correcto')

    lrLabel = Tk.Label(root, text="Learning rate")

    lrLabel.pack(side='left')

    lrEntry = Tk.Entry(root)

    lrEntry.insert(0, lr)

    lrEntry.bind("<Return>", learningRate)

    lrEntry.pack(side='left')

    def maxEpochs(event):

        global epochs

        try:

            epochs = float(epochsEntry.get())

        except ValueError:

            print('El numero ingresado no es correcto')

    epochsLabel = Tk.Label(root, text="Maximo de epocas")

    epochsLabel.pack(side='left')

    epochsEntry = Tk.Entry(root)

    epochsEntry.insert(0, epochs)

    epochsEntry.bind("<Return>", maxEpochs)

    epochsEntry.pack(side='left')

    def maxPresition(event):

        global precision

        try:

            precision = float(presitionEntry.get())

        except ValueError:

            print('El numero ingresado no es correcto')

    presitionLabel = Tk.Label(root, text="Presicion")

    presitionLabel.pack(side="left")

    presitionEntry= Tk.Entry(root)

    presitionEntry.insert(0,precision)

    presitionEntry.bind("<Return>", maxPresition)

    presitionEntry.pack(side='left')

    def initWeights():

        global weights

        # weights = np.random.uniform(-5, 5, np.array(xi).shape[1] + 1)

        weights = np.random.uniform(-1, 1, np.array(xi).shape[1]+1)

        #plot(weights)

    weightsButton = Tk.Button(

        root, text="Inicializar pesos", command=initWeights)

    weightsButton.pack(side='left')

    newX = 0

    newY = 0

    def newValueX(event):

        global newX

        try:

            newX = float(newXEntry.get())

        except ValueError:

            print('El numero ingresado no es correcto')

    newXEntry = None

    def newXAvailable():

        global newXEntry, newX

        Tk.Label(root, text="valor de x").pack(side='left')

        newXEntry = Tk.Entry(root)

        newXEntry.insert(0, newX)

        newXEntry.bind("<Return>", newValueX)

        newXEntry.pack(side='left')

    def newValueY(event):

        global newY

        try:

            newY = float(newYEntry.get())

        except ValueError:

            print('El numero ingresado no es correcto')

    newYEntry = None

    perceptron = None

    def newYAvailable():

        global newYEntry, newY, perceptron

        Tk.Label(root, text="valor de y").pack(side='left')

        newYEntry = Tk.Entry(root)

        newYEntry.insert(0, newY)

        newYEntry.bind("<Return>", newValueY)

        newYEntry.pack(side='left')

        evalButton = Tk.Button(root, text="Evaluar", command=evalXY)

        evalButton.pack(side='left')

    def evalXY():

        result = ad.predict([newX, newY, 1])

        color = ''

        if result == 1:

            color = 'g'

        else:

            color = 'r'

        a.scatter(newX, newY, c=color)

        canvas.draw()

    def train():

        global ad

        epoch = 0

        E = 1.0

        ad =  AdalineGD(np.array(xi).shape[1])

        \_error = []

        while epoch < epochs and np.abs(E) > precision:

            print('-------------- Epoca {} ----------------'.format(epoch + 1))

            a.title.set\_text('Epoca {}'.format(epoch + 1))

            [adjust\_w,E] = ad.fit(np.array(xi),np.array(d))

            \_error.append(np.abs(E))

            print('pesos',adjust\_w)

            print('presition',E)

            epoch = epoch+1

            errorCuadratico(epoch,\_error)

            root.update()

            if E < precision:break

        lrEntry.destroy()

        lrLabel.destroy()

        epochsEntry.destroy()

        epochsLabel.destroy()

        weightsButton.destroy()

        trainButton.destroy()

        newXAvailable()

        newYAvailable()

    def errorCuadratico(epocas, error):

        x = np.arange(epocas)

        err.plot(x,error, 'r')

        canvas.draw()

    trainButton = Tk.Button(root, text="Entrenar", command=train)

    trainButton.pack(side='left')

    Tk.mainloop()

import numpy as np

class AdalineGD(object):

    def \_\_init\_\_(self,n, eta = 0.01, n\_iter = 50):

        self.eta = eta

        self.n\_iter = n\_iter

        self.w\_ = np.random.uniform(-1,1,1 + n)

        self.cost\_ = []

    def fit(self, X, y):

        output = self.net\_input(X)

        errors = (y - output)

        self.w\_[1:] += self.eta \* X.T.dot(errors)

        self.w\_[0] += self.eta \* errors.sum()

        cost = (errors \*\* 2).sum() / 2.0

        self.cost\_.append(cost)

        return [self.w\_,cost]

    def net\_input(self, X):

        return np.dot(X, self.w\_[1:]) + self.w\_[0]

    def activation(self, X):

        return self.net\_input(X)

    def predict(self, X):

        return np.where(self.activation(X) >= 0.0, 1, -1)

Conclucion:

El Adaline a parte de ser un poco diferente que el perceptrón es un algoritmo más complejo he inteligente, si bien el perceptrón nos permite realizar una separación entre dos grupos y catalogarlos, el adaline nos da la bondad de tener una clasificación mas exacta ya que busca la recta que divide los dos puntos soporte y se trata de colocar justo en medio de estos para tener una convergencia más exacta, también su método de ajuste de pesos es mas sofisticado ya que por medio de la formula se calcula y se determina si un peso debe de ser aumentado o disminuido. Sin duda una neurona muy potente y con una escalabilidad enorme

https://drive.google.com/open?id=1NvUJlLX4ht2Sn-avlHEciuTQ8mu-lJAn&authuser=1