COMPORTAMIENTO DE LAS REDES NEURONALES

Ricardo Andres Villalobos Marulanda
ricardovillalobos0990@gmail.com
Anderson Gustavo Llano Davila
andersonllano1927@gmail.com
John Sebastian Nieto Gil
nieto100.jsng@gmail.com

RESUMEN

Las redes neuronales artificiales imitan el comportamiento del cerebro humano, principalmente de las neuronas y sus conexiones, a partir de la descomposición de los elementos naturales de las neuronas como elementos del proceso, enlaces, capas entre otras y su proceso para procesar patrones se puede matemáticamente crear modelos artificiales que solucionen computacionalmente problemas difíciles de resolver mediante tecnicas algoritmicas convencionales., Los algoritmos y todo el proceso de aprendizaje en las ecuaciones realizadas son complejas y suelen llevar bastante tiempo computacionalmente hablando. De acuerdo a lo especificado anteriormente se encuentran distintos modelos de aprendizaje como lo son la red perceptrón o las redes neuronales múltiples las cuales sirven para resolver casos particulares como problemas linealmente separables o no separables. algunos problemas que pueden existir en estos modelos puesto que se deben de desarrollar sobre parámetros estrictamente supervisados por un experto con el fin de evitar problemas ya conocidos como el Underfitting que es la capacidad de una red neuronal de no lograr solucionar e identificar nuevos problemas solo los que conoce o el Overfitting que es es sobre entrenamiento de una red., Lo más importante son los beneficios presentados en los campos de aplicación como el reconocimiento de patrones, problemas de optimización o clasificación, y se pueden integrar en un sistema de ayuda a la toma de decisiones, pero no son una panacea capaz de resolver todos los problemas: todo lo contrario, son modelos muy especializados que pueden aplicarse en dominios muy concretos.

PALABRAS CLAVE

Backpropagation, Perceptrón, Neurona, Regresión Lineal, Aprendizaje, Automatización, Aprendizaje autónomo, Overfitting, Underfitting, Algoritmos.

ABSTRACT

Artificial neural networks mimic the behavior of the human brain, mainly of neurons and their connections, from the decomposition of the natural elements of neurons as elements of the process, links, layers among others and their process for pattern patterns can be Mathematically Create artificial models that solve computationally difficult problems to

solve by conventional algorithmic techniques., The algorithms and the entire learning process in the equations performed are complex and usually take a long time computationally speaking. According to what has been specified recently, several learning models are found, such as red perception or multiple neural networks which serve to solve particular cases as linearly separable or non-separable problems. Some problems that may exist in these models since they must be developed on parameters strictly supervised by an expert in order to avoid problems already known as Underfitting, which is the ability of a red neuronal to fail to solve and identify new problems only those who know or the Overfitting that is about training a network., The most important are the benefits specified in the fields of application such as pattern recognition, optimization or classification problems, and can be integrated into a system of help to take of decisions, but they are not a panacea capable of solving all problems: on the contrary, they are very specialized models that can be controlled in very specific domains.

KEYWORDS

Backpropagation, Perceptron, Neuron, Linear Regression, Learning, Automation, Autonomous Learning, Overfitting, Underfitting, Algorithms.

INTRODUCCIÓN

Los sistemas de cómputo tradicional procesan la información en forma secuencial; un computador serial consiste por lo general de un solo procesador que puede manipular instrucciones y datos que se localizan en la memoria, el procesador lee, y ejecuta una a una las instrucciones en la memoria; este sistema serial es secuencial, todo sucede en una sola secuencia determinística de operaciones.

Las RNA no ejecutan instrucciones, responden en paralelo a las entradas que se les presente. El resultado no se almacena en una posición de memoria, este es el estado de la red para el cual se logra equilibrio. El conocimiento de una red neuronal no se almacena en instrucciones, el poder de la red está en su topología y en los valores de las conexiones (pesos) entre neuronas. Las RNA son una teoría que aún está en proceso de desarrollo, su verdadera potencialidad no se ha alcanzado todavía; aunque los investigadores han desarrollado potentes algoritmos de aprendizaje de gran valor práctico, las representaciones y procedimientos de que se sirve el cerebro, son aún desconocidas Tarde o temprano los estudios computacionales del aprendizaje con RNA acaban por converger a los métodos descubiertos por evolución, cuando eso suceda, muchos datos empíricos concernientes al cerebro comenzarán súbitamente a adquirir sentido y se tornarán factibles muchas aplicaciones desconocidas de las redes neuronales.

ALGORITMOS USADOS

Perceptrón

Red Neuronal Simple

Red Neuronal Múltiple

Backpropagation

LENGUAJE DE PROGRAMACIÓN UTILIZADO

Python:

Python es un lenguaje de scripting multiplataforma y orientado a objetos, preparado para realizar cualquier tipo de programa. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo.

Es un lenguaje fuertemente y dinámicamente tipado lo que permite una gran flexibilidad y optimización a la hora de programar. Otro objetivo del diseño del lenguaje es la facilidad de extensión. Se pueden escribir nuevos módulos fácilmente en C o C++. Python puede incluirse en aplicaciones que necesitan una interfaz programable.

Para finalizar, es importante traer a mención que python cuenta con una gran cantidad de librerías desarrolladas por la amplia comunidad que existe. Lo que permite encontrar una amplia variedad de soluciones a los diferentes tipos de problemas que se presentan.

CÓDIGO DE LOS ALGORITMOS IMPLEMENTADOS

Redes_neuroanles.py:

```
from tkinter import ttk
from tkinter import *
from Perceptron import Perceptron
from RedNeuronalSimple import RedNeuronalSimple
from RedNeuronalMulticapa import RedNeuronalMulticapa
from numpy import array, dot, random
import time

class RedesNeuronales:
  def __init__(self):
      self.ventana = Tk()
      self.ventana.title("Redes Neuronales")
      self.ventana.geometry("600x550")
      self.ventana.minsize(600, 550)
      self.ventana.configure(cursor="bogosity")
```

```
        self.notebook = ttk.Notebook(self.ventana)
        self.notebook.pack(fill='both',expand='yes')
        self.entradas = { }
        self.salidas = { }
        self.bahias = { }
        self.activacion = { }
        self.entrenamiento = { }
        self.instances = { }
        self.ob = { }

    def createFrame(self, instancia, title):
        """
        Recibe la instancion y el titulo de la ventana
        y devuelve la creacion de la pestaña creada con el titulo
        """
        self.instances[instancia] = ttk.Frame(self.notebook)
        self.notebook.add(self.instances.get(instancia), text=title)

    def interface(self, instancia, entradas, descripcion, Ob):
        """
        Recibe la instancia, el numero de entradas, una descripcion y el obejeto
        Ejecuta la interface con la instancia seleccionada, con el numero de entradas y
salidas de la neurona
        """
        etiquetaEntradas = []
        etiquetaSalidas = []
        nombreEntradas = []
        nombreSalidas = []
        self.entradas[instancia] = []
        self.salidas[instancia] = []
        # Etiquetas en la instancia con recomendaciones de USO
        Label(self.instances.get(instancia), text="Ingrese la misma cantidad de item para
cada entrada, separados por coma y sin espacio" + "\ n"
        "como se indica en las guias de ejemplo de color VERDE ", fg='black' ).place(x=0
,y=10)
        Label(self.instances.get(instancia), text=descripcion.get("nombre"), fg='green',
font="Times 14 bold").place(x=5 ,y=60)
        Label(self.instances.get(instancia), text=descripcion.get("entradas"), fg='green',
font="Times 14 bold").place(x=130 ,y=60)
        Label(self.instances.get(instancia), text=descripcion.get("salidas"), fg='green',
font="Times 14 bold").place(x=380 ,y=60)
```

```
    # Genera las entradas y salidas segun el numero de entradas que recibe la inter-
face y se itera segun ese numero generando los campos necesarios
    i = 0
    for i in range(entradas):
        etiquetaEntradas.append(Label(self.instances.get(instancia),text="Entrada # "
+ str(i+1) + ' :').place(x=10 ,y=100 + i * 20))
        self.entradas[instancia].append(StringVar())
        nombreEntradas.append(Entry(self.instances.get(instancia), textvariable=self.entradas[instanci
i * 20))


        self.salidas[instancia].append(StringVar())
        etiquetaSalidas.append(Label(self.instances.get(instancia),text="Salida # " +
str(i+1) + ' :').place(x=300 ,y=100 + i * 20))
        nombreSalidas.append(Entry(self.instances.get(instancia), textvariable=self.salidas[instancia][i]
+ i * 20))
    # Se declara la etiqueta dentron de la instanca, una descripcion, ubicacion y la
variable que contiene el valor dentro de la Entrada
    Label(self.instances.get(instancia),text="Ingrese el sesgo de bahias").place(x=10
,y=100 + (i+2) * 20)
    self.bahias[instancia] = DoubleVar()
    Entry(self.instances.get(instancia),textvariable=self.bahias[instancia]).place(x=200,y=100
+ (i+2) * 20)


    Label(self.instances.get(instancia),text="Ingrese entrenamiento").place(x=10 ,y=100
+ (i+3) * 20)
    self.entrenamiento[instancia] = IntVar()
    Entry(self.instances.get(instancia),textvariable=self.entrenamiento[instancia]).place(x=200,y=100
+ (i+3) * 20)


    Label(self.instances.get(instancia),text="Seleccione la Funcion").place(x=10 ,y=100
+ (i+4) * 20)
    self.activacion[instancia] = StringVar()
    ttk.Combobox(self.instances.get(instancia),values=("sigmoide","tangente"),textvariable=self.activ
+ (i+4) * 20)


    Button(self.instances.get(instancia),text="Entrenar",fg="black", bg="white",
font="fixedsys 14 bold", cursor="spider", command=lambda: self.callNeurona(instancia)).place(x=0
,y=100 + (i+7) * 20, height=40, width=550)
    self.ob[instancia] = Ob


  def callNeurona(self, instancia):
    total = len(self.entradas[instancia])
    entradas = []
```

```
        salidas = []

        for i in range(total):
            entradas.append([float(x) for x in self.entradas[instancia][i].get().split(sep=',')])
             salida = [float(x) for x in self.salidas[instancia][i].get().split(sep=',')]
            if len(salida) == 1:
                salidas.append(float(salida[0]))
            else:
                 salidas.append(salida)

        ob = self.ob[instancia](entradas, salidas, self.bahias[instancia].get(), self.entrenamiento[instancia].g
self.activacion[instancia].get())

        # Se asigna el valor de TIEMPO a la variable inicio, antes de la funcion entrenar
        inicio =time.time()
        ob.entrenar()
        final = time.time()
        tiempo = round(final-inicio,3)
        # Se asigna el valir de TIEMPO a la variable final, despues de la funcion entrenar

        cadena = ob.imprimirResultado()



        Label(self.instances.get(instancia), text="RESULTADO", fg="red").place(x=0
,y=100 + (i+9) * 20)
        Label( self.instances.get(instancia), text=cadena, fg='red' ).place(x=0 ,y=100 +
(i+10) * 20)
        Label( self.instances.get( instancia ), text="El tiempo que se tardo fue de: " +
str(tiempo) + " Segundos", fg='red' ).place( x=0, y=100 + (i + 17) * 20 )


        ob.generarGrafico()


    def sostenerVentana(self):
        """
        No recibe Parametro
        Y ejecuta la ventana en el bucle principal
        """
        self.ventana.mainloop()

per = { 'nombre':'Perceptron' , "entradas" : "0,0,1" , "salidas" : " 1 " }
redsim = { 'nombre':'Red Simple' , "entradas" : "1,1,1" , "salidas" : " 1 " }
```

```
redmul = { 'nombre':'Red Multiple' , "entradas" : "0,1" , "salidas" : " 1,0 " }

ob = RedesNeuronales()
ob.createFrame('red', 'Perceptrón')
ob.createFrame('red2', 'Red neuronal simple')
ob.createFrame('red3', 'Red neuronal multicapa')
ob.interface('red', 4, per, Perceptron)
ob.interface('red2', 4, redsim, RedNeuronalSimple)
ob.interface('red3', 7, redmul, RedNeuronalMulticapa)
ob.sostenerVentana()
```

Perceptron.py

```
from random import choice
from numpy import array, dot, random
import matplotlib.pyplot as plt
from FuncionesActivacion import FuncionesActivacion

class Perceptron(FuncionesActivacion):
  def __init__(self, entradas, salidas, bahias, n, activacion="relu"):
      total = len(entradas)
      entrenamiento = []
      for i in range(total):
          entrenamiento.append(array((array(entradas[i]), salidas[i])))

      self.entrenamiento = entrenamiento
      self.w = random.rand(3)
      self.errores = []
      self.esperados = []
      self.bahias = bahias
      self.n = n

  def entrenar(self):
      for i in range(self.n):
          x, esperado = choice(self.entrenamiento)
          resultado = dot(self.w, x)
          self.esperados.append(esperado)
          error = esperado - self.relu(resultado)
          self.errores.append(error)
          # Ajuste
          self.w += self.bahias * error * x
```

```
  def imprimirResultado(self):
      cadena = ""
      for x, _ in self.entrenamiento:
          resultado = dot(self.w, x)
          cadena += f"{ x[:3]} : { resultado} -> { self.relu(resultado)} \ n"
      return cadena

  def generarGrafico(self):
      plt.title("Perceptron")
      plt.plot(self.errores,'-',color='red', Label="Errores")
      plt.plot(self.esperados,'* ', color='green', Label="Esperados")
      plt.legend(loc="lower right")
      plt.show()
```

RedNeuronalSimple.py

```
from numpy import exp, array, random, dot, ravel, sum, abs
from FuncionesActivacion import FuncionesActivacion
import matplotlib.pyplot as plt

class RedNeuronalSimple(FuncionesActivacion):
  def __init__(self, entradas, salidas, bahias, n, activacion):
      self.nombreactivacion = activacion
      self.activacion = None
      self.activacion_prima = None
      self.pesos_signaticos = 2 * random.random((3,1)) - 1
      self.entradas = array(entradas)
      self.errores = []
      self.esperados= []
      self.salidas = array([salidas]).T
      self.bahias = bahias
      self.n = n

  def entrenar(self):
      self.validaractivacion()
      for i in range(self.n):
          salida = self.pensar(self.entradas)
          error = self.salidas - salida
          self.errores.append(abs(sum(error)))
          ajuste = dot(self.entradas.T, error * self.activacion_prima(salida))
          self.pesos_signaticos += ajuste
          self.esperados.append(ajuste)
```

```
  def pensar(self, entrada):
      return self.activacion(dot(entrada, self.pesos_signaticos))


  def imprimirResultado(self):
      entrada_prueba = array([1,0,0])
      return f"Predicion para la entrada { entrada_prueba} es { self.pensar(entrada_prueba)}
"


  def validaractivacion(self):
      if self.nombreactivacion == 'sigmoide':
          self.activacion = self.sigmoide
          self.activacion_prima = self.sigmoide_derivado
      elif self.nombreactivacion == 'tangente':
          self.activacion = self.tangente
          self.activacion_prima = self.tangente_derivada

  def generarGrafico(self):
      plt.title("Perceptron")
      plt.plot(self.errores, '-',color='red', Label="Errores")
      plt.legend(loc="upper right")
      plt.show()
```

RedNeuronalMulticapa.py

```
from numpy import exp, array, random, dot, ravel, sum, abs
from FuncionesActivacion import FuncionesActivacion
import matplotlib.pyplot as plt

class RedNeuronalSimple(FuncionesActivacion):
  def __init__(self, entradas, salidas, bahias, n, activacion):
      self.nombreactivacion = activacion
      self.activacion = None
      self.activacion_prima = None
      self.pesos_signaticos = 2 * random.random((3,1)) - 1
      self.entradas = array(entradas)
      self.errores = []
      self.esperados= []
      self.salidas = array([salidas]).T
      self.bahias = bahias
      self.n = n
```

```
  def entrenar(self):
      self.validaractivacion()
      for i in range(self.n):
          salida = self.pensar(self.entradas)
          error = self.salidas - salida
          self.errores.append(abs(sum(error)))
          ajuste = dot(self.entradas.T, error * self.activacion_prima(salida))
          self.pesos_signaticos += ajuste
          self.esperados.append(ajuste)

  def pensar(self, entrada):
      return self.activacion(dot(entrada, self.pesos_signaticos))


  def imprimirResultado(self):
      entrada_prueba = array([1,0,0])
      return f"Predicion para la entrada { entrada_prueba} es { self.pensar(entrada_prueba)}
"


  def validaractivacion(self):
      if self.nombreactivacion == 'sigmoide':
          self.activacion = self.sigmoide
          self.activacion_prima = self.sigmoide_derivado
      elif self.nombreactivacion == 'tangente':
          self.activacion = self.tangente
          self.activacion_prima = self.tangente_derivada

  def generarGrafico(self):
      plt.title("Perceptron")
      plt.plot(self.errores, '-',color='red', Label="Errores")
      plt.legend(loc="upper right")
      plt.show()
```

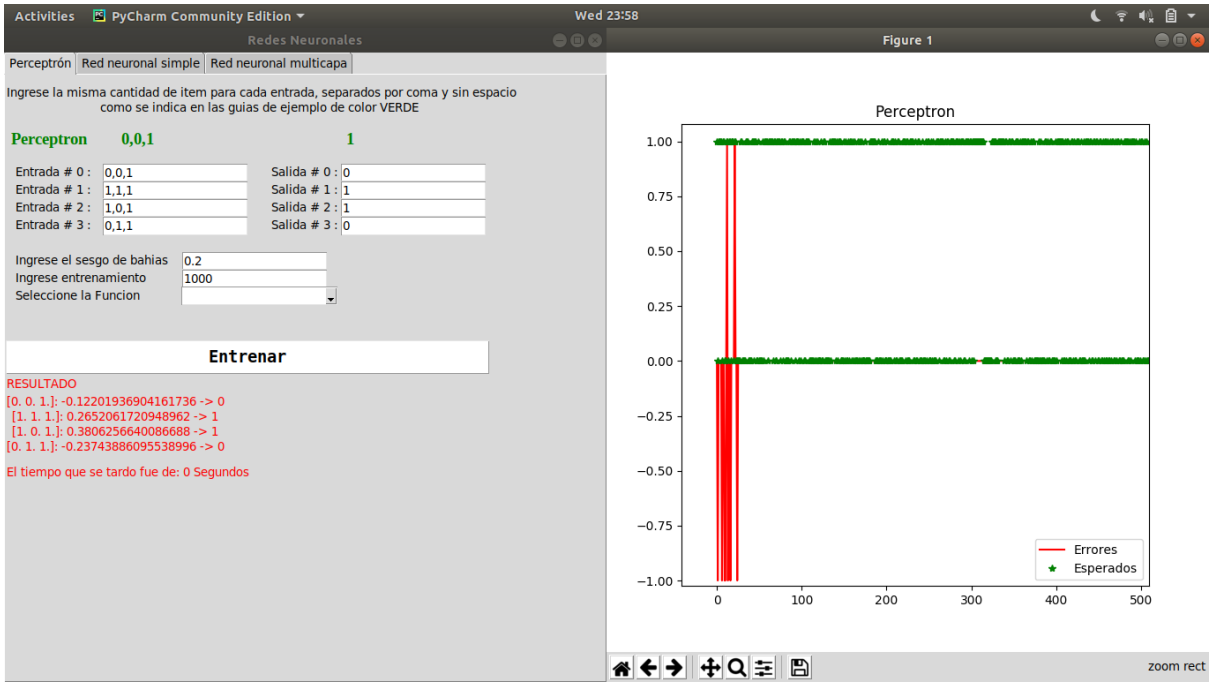
FuncionesActivacion.py

```
import numpy as np

class FuncionesActivacion:
```

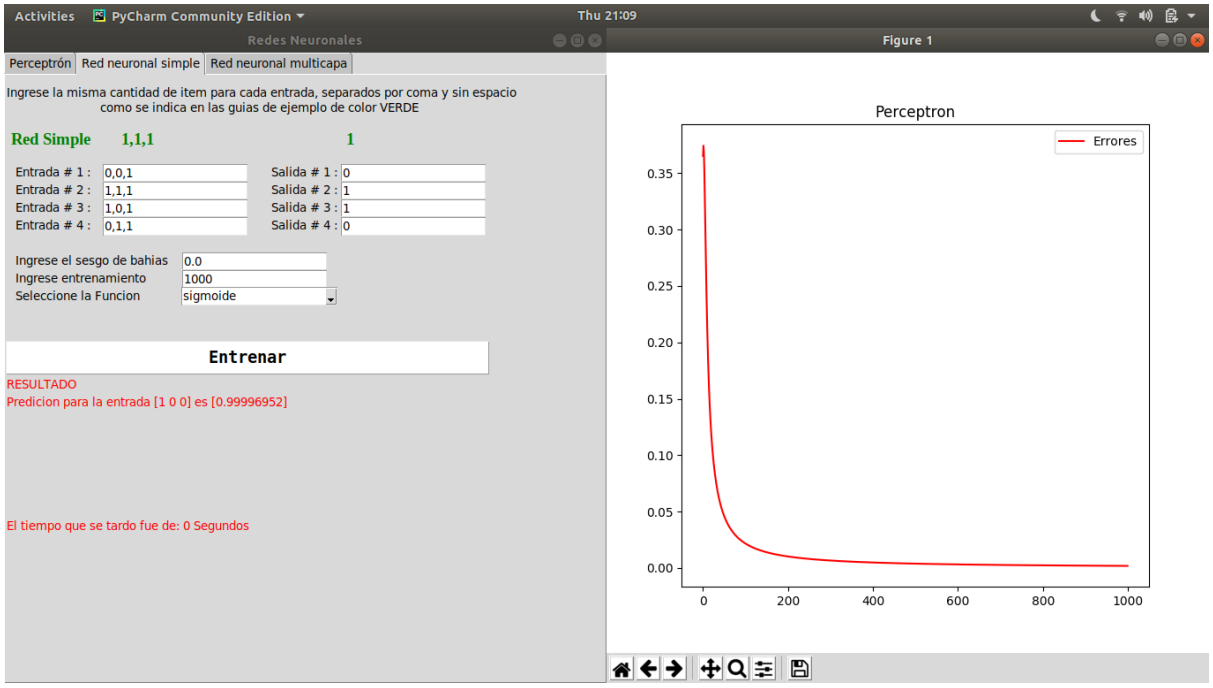
```
def sigmoide(self, x):
    return 1/(1 + np.exp(-x))

def sigmoide_derivado(self, x):
    return self.sigmoide(x) * (1 - self.sigmoide(x))

def tangente(self, x):
    return np.tanh(x)

def tangente_derivada(self, x):
    return 1 - x** 2

def relu(self, x):
    return 0 if x < 0 else 1
```

RESULTADOS ALCANZADOS

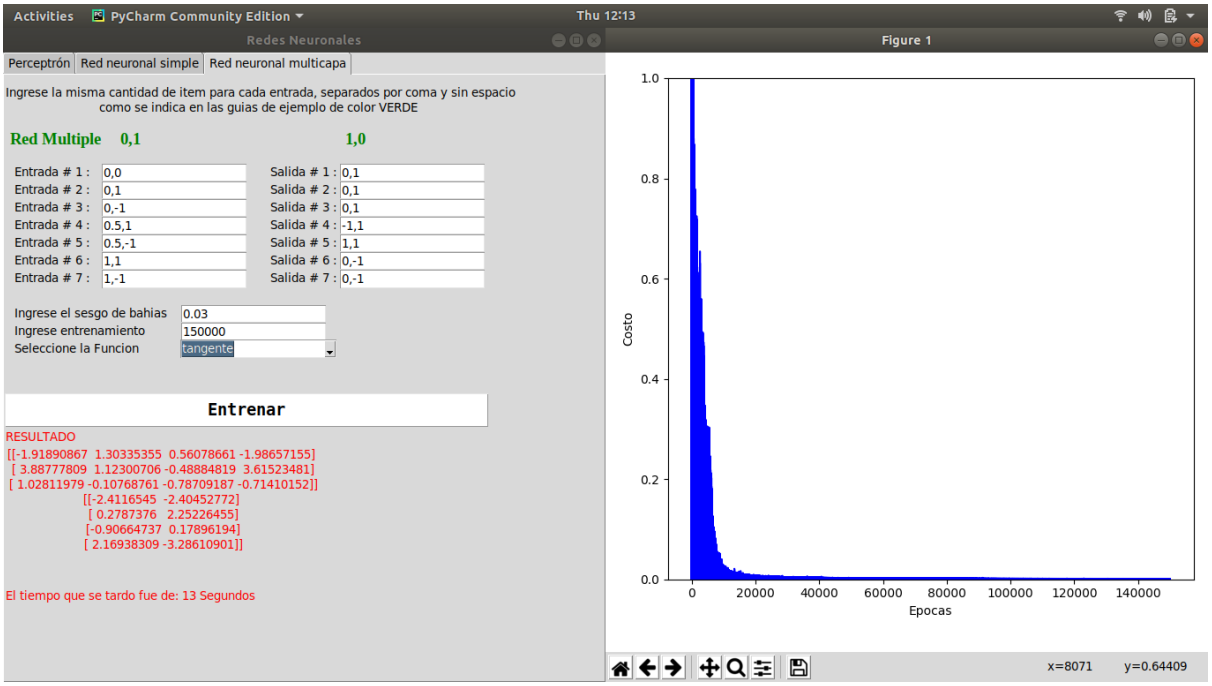
Perceptron



Red Neuronal Simple



Red Neuronal Múltiple



EVALUACIÓN DE LOS RESULTADOS DE LOS ALGORITMOS IMPLEMENTADOS

En la red de perceptrón si se utiliza un sesgo de bias con un número decimal muy pequeño este repercute que se tengan que utilizar más datos de entrenamiento de la red neuronal así como más se demora en encontrar una constancia entre sus valores esperados y de error, caso contrario si se utiliza un sesgo de bias con unos pocos decimales, recordemos que el sesgo inclina la recta y puede ser que si la movemos muy poco en cada interacción esta se demore más en encontrar la solución. en un segundo caso se tomaron las mismas entradas con las salidas opuestas y no se tuvo un resultado distinto.

En la Red Neuronal Simple al utilizar un sesgo de bias menor a una unidad nos damos cuenta que es necesario aumentar las épocas de entrenamiento para que la neurona intérprete y mejore su trabajo erróneo de decisiones, es de resaltar que esta neurona trabaja de una manera lineal sin capas ocultas, por lo tanto su forma de aprendizaje está determinada por una aplicación de la sumatoria de sus entradas multiplicadas por la cantidad errónea, es de allí su nombre simple RNA.

En cuanto a la red Multicapa que cuenta con 2 capas de entrada, 3 ocultas y 2 de salida se encuentra un patrón similar en cuanto al factor de aprendizaje el cual entre más decimales obtenga se tendrá que introducir una gran cantidad de datos de entrenamiento para lograr un costo menor durante las épocas de entrenamiento todo esto en con la función de activación tangente., ya que con la sigmoide se encuentra una constante de coste más alta dentro de todo el entrenamiento sin lograr reducción alguna

CONCLUSIONES

Estas redes neuronales fueron diseñadas con éxito, ya que al interactuar con ellas y revisar cada funcionalidad en el perceptrón, red neuronal simple y en la red neuronal multicapa podemos notar cómo al ingresar entradas de diferentes valores, con salidas esperadas y épocas de entrenamiento nos da resultados esperados, como resultados con gran cantidad errónea, es decir cada vez que entrenamos estas RNA y supervisamos sus resultados aprenderemos a identificar la funcionalidad de cada una y como interactúan a las respuestas esperadas, así podemos concluir que han sido de gran utilidad, la manera como se asimila cada vez mejor una RNA bien entrenada a la capacidad de razonabilidad de una persona, ya que con reconocimiento de patrones y la habilidad de para clasificar en base a ellos, podemos llegar a aplicaciones mucho más complejas y así mejorar nuestro conocimiento en este lenguaje de programación como en la interacción con RNA.

RECOMENDACIONES

El algoritmo implementado de perceptrón simple es útil en ciertas circunstancias, donde los datos son linealmente separables, se conocen los datos de entrada y de salida y este no va a garantizar el resultado óptimo o 0, para los casos donde se opte por utilizar los algoritmos de una red neuronal simple o múltiple se debe contemplar que los datos pueden abarcar un espacio multidimensional osea que no son linealmente separable este proceso de aprendizaje de acompañado de funciones de activación que apoyan la resolución de estos casos. al momento de evaluar la red a utilizar tenga en cuenta el campo donde se va implementar ya que para un problema de estadística puede ser de buen uso pero para un problema de clasificación puede que necesite otro algoritmo más complejo

DESCRIPCIÓN DE ALGORITMOS USADO

FUNCION DE ACTIVACION:
Importación de la librería numpy, después creamos una clase FuncionActivacion con 5 funciones que son la sigmoide, sigmoide_derivado,tangente,tangente_derivada y relu.
En la cual estas funciones le darán la debida activación a la neurona y dependiendo de cada una de ellas, se realizara una operación específica asignada.

RED NEURONAL SIMPLE:
Desde numpy, FuncionActivacion importamos parámetros de asignación como la de random, ravel,FuncionesActivacion entre otras, además la librería matplotlib para las respectivas gráficas. Es así como empezamos a describir este algoritmo en el cual definimos una clase RedNeuronalSimple que tiene objetos propios privada con atributos propios de entradas, salidas, bahías, las épocas y la respectiva función de activación, debido a la cantidad de entradas, continuamente construiremos arrays para efectuar en un recuadro cada una de las entradas y ver en pantalla a través de TKinter las gráficas deseadas.

COMPUTADOR USADO

PC1
LENOVO E460 - INTEL CORE I5 5200U - 12GB RAM - 750GB HDD
PC2
SAMSUNG -INTEL(R) CORE I5-2450M- 4 GB RAM
PC3
LENOVO G470 - INTEL CORE I3 2350 -8GB RAM - 500GB

PERCEPTRON - PRUEBA # 1

COMPUTADORES	PC1	PC2	PC3
ENTRADAS	0,0,1 1,1,1 1,0,1 0,1,1	0,0,1 1,1,1 1,0,1 0,1,1	0,0,1 1,1,1 1,0,1 0,1,1
SALIDAS	0,1,1,0 .T	0,1,1,0 .T	0,1,1,0 .T
FACTOR DE APRENDIZAJE	0.2	0.2	0.2
ÉPOCAS	10000	10000	10000
FUNCIÓN DE ACTIVACIÓN	Relu	Relu	Relu
TIEMPO	0.136s	0.367s	0.766s

PERCEPTRON - PRUEBA # 2

COMPUTADORES	PC1	PC2	PC3
ENTRADAS	0,0,1 1,1,1 1,0,1 0,1,1	0,0,1 1,1,1 1,0,1 0,1,1	0,0,1 1,1,1 1,0,1 0,1,1
SALIDAS	1,0,0,1 .T	1,0,01 .T	1,0,0,1 .T
FACTOR DE APRENDIZAJE	0.0002	0.0002	0.0002
ÉPOCAS	1800000	1800000	1800000
FUNCIÓN DE ACTIVACIÓN ACTIVACIÓN	Relu	Relu	Relu
TIEMPO	13.715s	36.87	52.87s

PERCEPTRON - PRUEBA # 3

COMPUTADORES	PC1	PC2	PC3
ENTRADAS	1,1,1 1,0,1 0,0,1 0,1,0	1,0,1 1,0,1 1,0,1 1,1,1	1,0,1 0,1,1 1,1,1 0,0,1
SALIDAS	1,1,0,1 .T	1,0,1,1 .T	1,1,0,1 .T
FACTOR DE APRENDIZAJE	0.05	0.05	0.05
ÉPOCAS	300000	300000	300000
FUNCIÓN DE ACTIVACIÓN	Relu	Relu	Relu
TIEMPO	1.947s	5.737s	7.901s

RED NEURONAL SIMPLE-PRUEBA # 1

COMPUTADORES	PC1	PC2	PC3
ENTRADAS	1,1,1 1,0,1 0,0,1 0,1,0	1,1,1 1,0,1 0,0,1 0,1,0	1,1,1 1,0,1 0,0,1 0,1,0
SALIDAS	1,1,0,1 .T	1,0,1,1 .T	1,1,0,1 .T
ÉPOCAS	200000	200000	200000
FUNCIÓN DE ACTIVACIÓN	SIGMOIDE	SIGMOIDE	SIGMOIDE
TIEMPO	5.859s	14.622s	21.001s

RED NEURONAL SIMPLE- PRUEBA # 2

COMPUTADORES	PC1	PC2	PC3
ENTRADAS	1,1,1 1,0,1 0,0,1 0,1,0	1,1,1 1,0,1 0,0,1 0,1,0	1,1,1 1,0,1 0,0,1 0,1,0
SALIDAS	1,1,0,1 .T	1,0,1,1 .T	1,1,0,1 .T
ÉPOCAS	200000	200000	200000
FUNCIÓN DE ACTIVACIÓN	TANGENTE	TANGENTE	TANGENTE
TIEMPO	3.677s	9.421	16.234

RED NEURONAL MULTICAPA- PRUEBA # 1

COMPUTADORES	PC1	PC2	PC3
ENTRADAS	1,0 1,1 0,1 1,1 1,-1 0.5.1 0.5-1	1,0 1,1 0,1 1,1 1,-1 0.5.1 0.5-1	1,0 1,1 0,1 1,1 1,-1 0.5.1 0.5-1
SALIDAS	1,1 1,0 0,1 0,0 0.5,-1 1,-1 -1,0.5	1,1, 1,0 0,1 0,0 0.5,-1 1,-1 -1,0.5	1,1, 1,0 0,1 0,0 0.5,-1 1,-1 -1,0.5
FACTOR DE APRENDIZAJE	0.009	0.009	0.009
ÉPOCAS	500000	500000	500000
FUNCIÓN DE ACTIVACIÓN	SIGMOIDE	SIGMOIDE	SIGMOIDE
TIEMPO	41.582s	83.523s	98.003s

RED NEURONAL MULTICAPA-PRUEBA # 2

COMPUTADORES	PC1	PC2	PC3
ENTRADAS	1,0 1,1 0,1 1,1 1,-1 0.5.1 0.5-1	1,0 1,1 0,1 1,1 1,-1 0.5.1 0.5-1	1,0 1,1 0,1 1,1 1,-1 0.5.1 0.5-1
SALIDAS	1,1 1,0 0,1 0,0 0.5,-1 1,-1 -1,0.5	1,1 1,0 0,1 0,0 0.5,-1 1,-1 -1,0.5	1,1 1,0 0,1 0,0 0.5,-1 1,-1 -1,0.5
FACTOR DE APRENDIZAJE	0.06	0.06	0.06
ÉPOCAS	700000	700000	700000
FUNCIÓN DE ACTIVACIÓN	TANGENTE	TANGENTE	TANGENTE
TIEMPO	79.368	83.158s	95.666s