**Trabajo práctico 4**

**Red con Neuronas Variables**

Valentin Faraz

Ing.Computación

from math import e

from random import random, uniform

import matplotlib.pyplot as plt

def entrenar\_perceptron(tabla, lr, iteraciones, n):

# Pesos de la Capa oculta

w\_oculta = []

# Pesos ultima Neurona

w\_salida = []

l\_error = []

for i in range(n):

w\_oculta.append([])

for \_ in range(3):

w\_oculta[i].append(round(uniform(-1, 1), 2))

# Pesos de la Ultima Neurona

for i in range(n+1):

w\_salida.append(round(uniform(-1, 1), 2))

print("Pesos capa oculta:", w\_oculta)

print("Pesos ultima neurona:", w\_salida)

for iteraciones in range(iteraciones):

print("Iteracion: ", iteraciones+1)

for combinacion in tabla:

x = 0

# Salida de las neuronas de la capa oculta

salida = [1]

# Delta de los pesos de la neurona de salida

dw\_n\_salida = []

# Delta de las neuronas capa oculta

delta\_co = []

# Delta de los pesos de las neuronas capa oculta

dw\_co = []

#print("listas vacias:",salida,dw\_co,dw\_n\_salida,delta\_co)

print("Fila de la tabla de la verdad:", combinacion)

# entradas iniciales

e0 = 1

e1 = int(combinacion[0])

e2 = int(combinacion[1])

solucion\_deseada = int(combinacion[2])

x\_neu\_salida = 0

for i in range(n):

# x es sumatoria de cada peso por su entrada

x = (e0\*w\_oculta[i][0]) + \

(e1\*(w\_oculta[i][1])) + (e2\*w\_oculta[i][2])

# salida de las neuronas capa oculta

salida.append(1/(1+e\*\*(-x)))

for i in range(n+1):

# salida de la ultima neurona

x\_neu\_salida += salida[i]\*w\_salida[i]

salida\_real = 1/(1+e\*\*(-x\_neu\_salida))

error = solucion\_deseada-salida\_real

deltaf = salida\_real\*(1-salida\_real)\*error

# Delta de los pesos de la neurona de salida

# dw\_n\_salida.append(lr\*e0\*deltaf)

for i in range(n+1):

dw\_n\_salida.append(lr\*salida[i]\*deltaf)

# Nuevos pesos ultima Neurona

w\_salida[i] = w\_salida[i]+dw\_n\_salida[i]

salida.pop(0)

for i in range(n):

# Deltas de las neuronas capa oculta

delta\_co.append(salida[i]\*(1-salida[i])\*deltaf)

# Delta de los pesos de la capa oculta

dw\_co.append(lr\*e0\*delta\_co[i])

dw\_co.append(lr\*e1\*delta\_co[i])

dw\_co.append(lr\*e2\*delta\_co[i])

count = 0

for i in range(n):

for j in range(3):

# Nuevos pesos de la capa oculta

w\_oculta[i][j] = w\_oculta[i][j]+dw\_co[count]

count += 1

#print("Deltas de los pesos ultima Neurona:",dw\_n\_salida)

#print("Pesos ultima neurona:",w\_salida)

#print("Nuevos pesos:",w\_oculta)

print("Salida real:", salida\_real)

print("Salida deseada:", solucion\_deseada)

print("error:", error)

print("-------------------------")

l\_error.append(-error)

plt.plot(l\_error)

plt.ylabel('Error')

plt.xlabel('Iteraciones')

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

# Tabla de la verdad de XOR

tabla\_xor = ['000', '011', '101', '110']

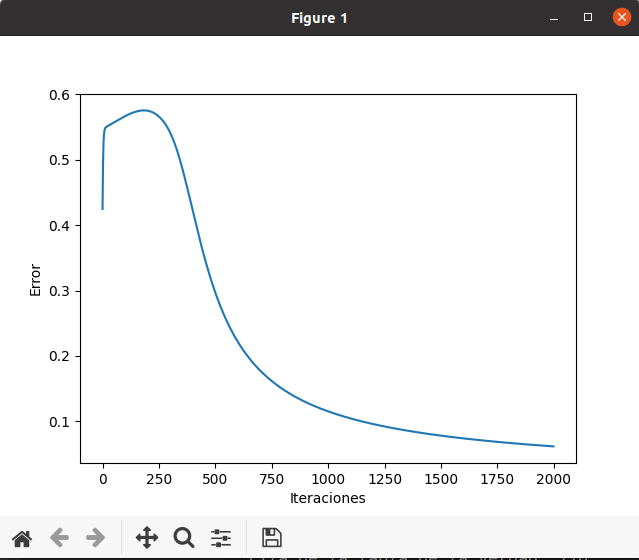
lr = 0.5

iteraciones = 2000

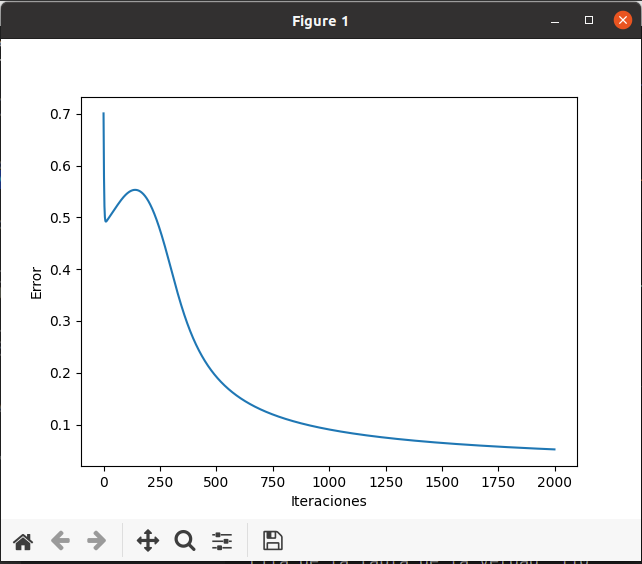
neuronas = 20

entrenar\_perceptron(tabla\_xor, lr, iteraciones, neuronas)

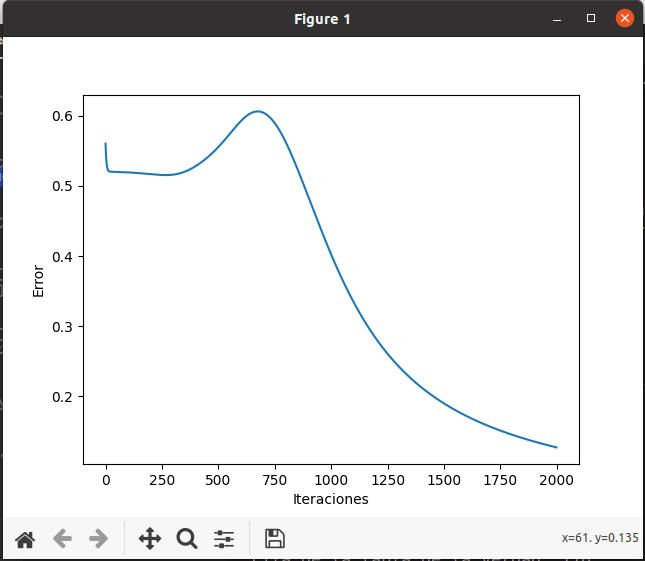
10 neuronas 2000 iteraciones



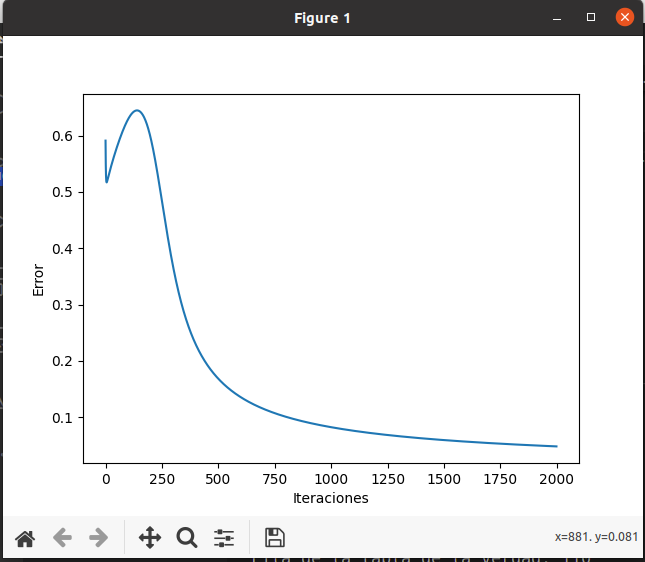
20 neuronas 2000 iteraciones



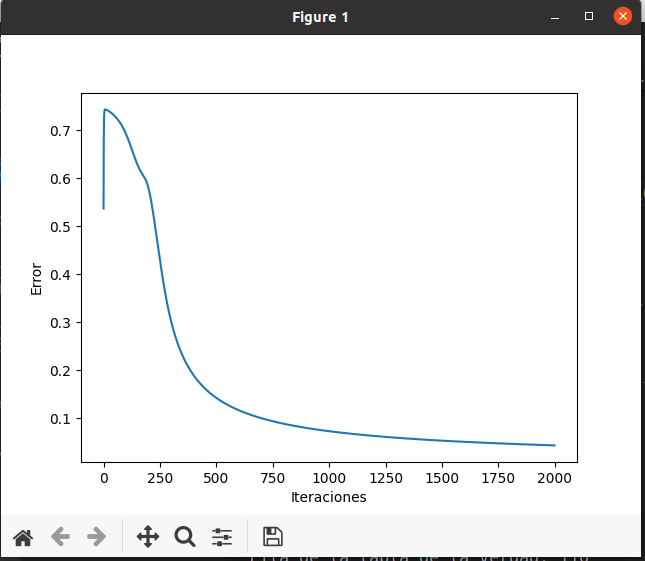
3 neuronas 2000 iteraciones



30 neuronas 2000 iteraciones



50 neuronas y 2000 iteraciones



PODEMOS CONCLUIR QUE LA VELOCIDAD DE APRENDIZAJE DE LA RED ES DIRECTAMENTE PROPORCIONAL A NÚMERO DE NEURONAS.

SE OBSERVA CLARAMENTE QUE DEJANDO FIJO EL NÚMERO DE ITERACIONES Y AUMENTANDO EL NÚMERO DE NEURONAS EN LA CAPA OCULTA DE LA RED, LA GRÁFICA DEL ERROR TIENDE MÁS RÁPIDAMENTE A 0. ES DECIR APRENDE MÁS RÁPIDO CON MÁS NEURONAS