Parcial 2 Inteligencia artificial

Stefany Lorena Sanchez Bernal - Lorena10sanchez18@gmail.com
Elkin De Jesus Ramirez Jimenez - eknramirez@gmail.com
Juan Felipe Marin Arenas - juanfelipemarin3@outlook.com

Resumen

Este page está desarrollado para el trabajo parcial de inteligencia artificial dictado por el ingeniero Carlos Londoño.
El trabajo esta direccionado a la comprensión de la terminología y técnicas de manejo de redes neuronales básicas hasta las redes neuronales más complejas o con capas ocultas. La inteligencia artificial como una de las ramas de la informática que más está siendo utilizada para el aprovechamiento en numerosos procesos, tanto industriales como cotidianos.
Buscamos saber en este proceso de aprendizaje como funciona una red neuronal. Cuanto aprende referente a los datos que le podemos ingresar, y como se entrena e itera para poder resolver una seria de procesos y poder arrojar un resultado esperado, nos apoyamos en la gráfica para poder evidenciar los resultados de una forma más comprensible e interactiva.
También trabajamos con librerías de control de tiempo para poder identificar los tiempos de compilaciones de los algoritmos en diferentes equipos informáticos.

Englihs Summary

His page is developed for the artificial intelligence partial work dictated by the engineer Carlos Londoño. The work is aimed at understanding the terminology and management techniques of basic neural networks to more complex neural networks or with hidden layers. Artificial intelligence as one of the branches of computer science that is being used most for the use in many processes, both industrial and everyday. We seek to know in this learning process how a red neuronal works. How much you learn to refer to the data that we can enter, and how you train and iterate to be able to solve a series of processes and be able to yield an expected result, we rely on the graph to be able to show the results in a more understandable and interactive way. We also work with time control libraries to identify the compilation times of the algorithms on different computer equipment.

Introducción

En este paper veremos un artículo dedicado al manejo e implementación de las redes Neuronales, desde un perceptron hasta una red neuronal multípara. Estudiando así, su funcionamiento y comportamiento de las neuronas.

Las Redes Neuronales (NN: Neural Networks) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos, constituidos por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros. El primer modelo de red neuronal fue propuesto en 1943 por McCulloch y Pitts en términos de un modelo computacional de actividad nerviosa.

Este modelo era un modelo binario, donde cada neurona tenía un escalón o umbral prefijado, y sirvió de base para los modelos posteriores. Una primera clasificación de los modelos de NN es: 1. Modelos inspirados en la Biología: Estos comprenden las redes que tratan de simular los sistemas neuronales biológicos, así como ciertas funciones como las auditivas o de visión. 2. Modelos artificiales aplicados: Estos modelos no tienen por qué guardar similitud estricta con los sistemas biológicos. Sus arquitecturas están bastante ligadas a las necesidades de las aplicaciones para las que son diseñados.

Lenguaje de programación usado

Utilizamos el lenguaje de programación Python el cual es un lenguaje de programación muy versátil para el trabajo de inteligencia artificial

Código de algoritmo implementado

1. Código para Perceptron

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

```
#Funcion de Activación
activacion = lambda x:0 if x < 0 else 1
```

```
#Set de Entrenamiento
entrenamiento = [
    (array([0,0,1]),0),
    (array([0,1,1]),1),
    (array([1,0,1]),1),
    (array([1,1,1]),1),
]
```

```
w = random.rand(3)
errores = []
esperados = []
bahias = 0.2
n = 100000
```

```
#Entrenamiento

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

tiempo1=(timeit("'Hola!'.replace('Hola', 'adios')"))
```

```
for x, _ in entrenamiento:
    resultado = dot(w, x)
    print("{}: {} -> {}".format(x[:3], resultado, activacion(resultado)))

tiempo2=(timeit("'Hola!'.replace('Hola', 'adios')"))
print("Tiempo total: ", tiempo2+tiempo1)
```

```
plt.plot(errores,'-',color='red')
plt.plot(esperados,'*', color='blue')
```

2. Código para Red Neuronal Simple

En una red simple con función de activación tangente a unas mayores épocas de entrenamientos el algoritmo responde de manera muy acertada o acoplada a los ajustes a errores. No podemos negar que algunas neuronas cuando de entrenan a muchas épocas o iteraciones de distorsionan en sus resultados esperados.

Resultados alcanzados

Parámetros

```
#Set de Entrenamiento
entrenamiento = [
    (array([0,0,1]),0),
    (array([0,1,1]),1),
    (array([1,0,1]),1),
    (array([1,1,1]),1),
]
```

Iteración del equipo 1 : Lorena Sanchez
Descripción Lenovo yoga 510, memoria ram: 8192MB, Intel Core i5 2.30Ghz, sistema operativo Win 10 64Bits

Perceptron

Se realizo el entrenamiento a una neurona usando perceptron

Se realizo la prueba cambiando el numero de interacciones

Numero de Iteraciones 1000.

Resultado:

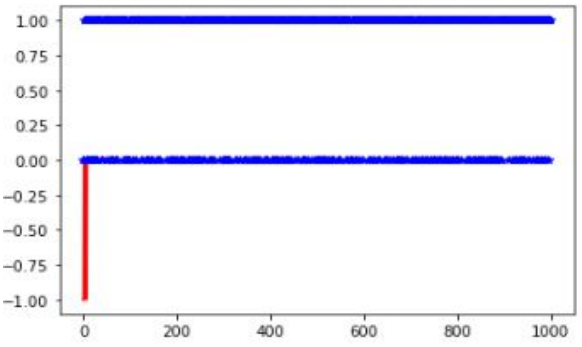
```
bahias = 0.2
n = 1000
```

```
#Entrenamiento
inicia_tiempo= time()
for i in range(n):
    x, esperado = choice(entrenamiento)
    resultado = dot(w, x)
    esperados.append(esperado)
    error = esperado - activacion(resultado)
    errores.append(error)
    #Ajuste
    w += bahias * error * x
```

```
for x, _ in entrenamiento:
    resultado = dot(w, x)
    print("{}: {} -> {}".format(x[:3], resultado, activacion(resultado)))

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.014441915666668026 -> 0
[0 1 1]: 0.1500520377661661 -> 1
[1 0 1]: 0.14649906260375617 -> 1
[1 1 1]: 0.3109930160365903 -> 1
Tiempo total:  1.0353410243988037
```

Gráfica:



Numero de Iteraciones 10000

Resultado:

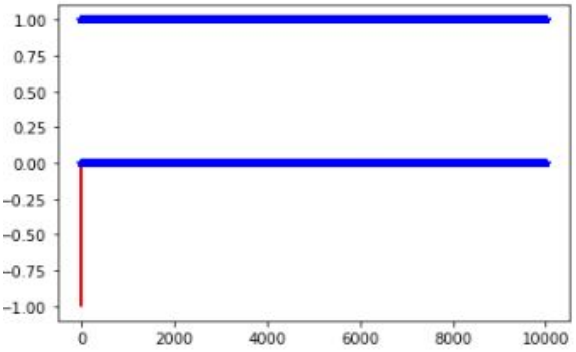
```
bahias = 0.2
n = 10000
```

```
#Entrenamiento
inicia_tiempo= time()
for i in range(n):
    x, esperado = choice(entrenamiento)
    resultado = dot(w, x)
    esperados.append(esperado)
    error = esperado - activacion(resultado)
    errores.append(error)
    #Ajuste
    w += bahias * error * x
```

```
for x, _ in entrenamiento:
    resultado = dot(w, x)
    print("{}: {} -> {}".format(x[:3], resultado, activacion(resultado)))


tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.12614776333721917 -> 0
[0 1 1]: 0.089763833826836 -> 1
[1 0 1]: 0.8417457604425393 -> 1
[1 1 1]: 1.0576573576065944 -> 1
Tiempo total:  0.861661434173584
```

Gráfica:



Numero de Iteraciones 100000

Resultado:

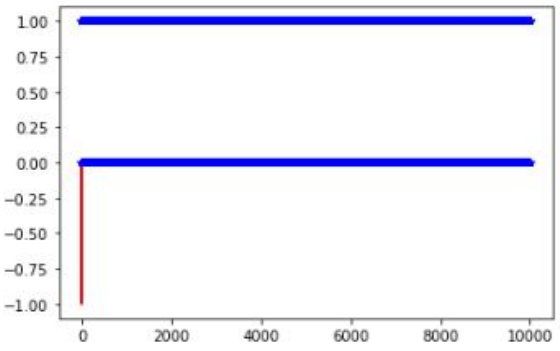
```
bahias = 0.2
n = 100000
```

```
#Entrenamiento
inicia_tiempo= time()
for i in range(n):
    x, esperado = choice(entrenamiento)
    resultado = dot(w, x)
    esperados.append(esperado)
    error = esperado - activacion(resultado)
    errores.append(error)
    #Ajuste
    w += bahias * error * x
```

```
for x, _ in entrenamiento:
    resultado = dot(w, x)
    print("{}: {} -> {}".format(x[:3], resultado, activacion(resultado)))

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.01293554713139927 2 -> 0
[0 1 1]: 0.6639728199555996 -> 1
[1 0 1]: 0.45691109166959737 -> 1
[1 1 1]: 1.1338194587565964 -> 1
Tiempo total:  0.9689650535583496
```

Gráfica:



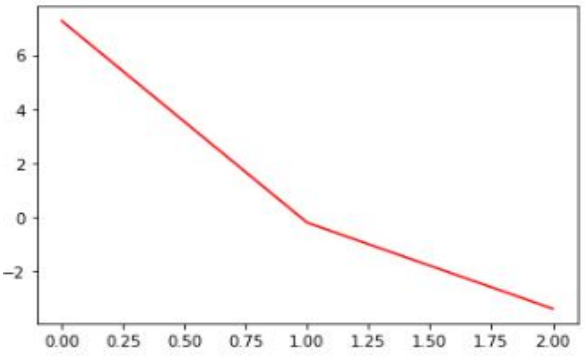
Red Neuronal Simple

Numero de Iteraciones con 1000

```
if __name__ == '__main__':
    red_neuronal = RedNeuronal()
    entradas = array([[0,0,1], [1,1,1], [1,0,1], [0,1,1]])
    salidas = array([[0,1,1,0]]).T
    red_neuronal.entrenamiento(entradas,salidas,1000)
    print(red_neuronal.pesos_signaticos)
    print(red_neuronal.pensar(array([1,0,0])))

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[[ 7.26325933]
 [-0.21982707]
 [-3.41504698]]
[0.99929967]
Tiempo total:  0.030916690826416016
```

Gráfica:

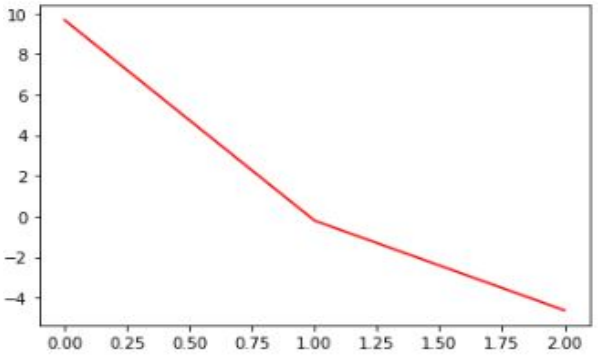


Numero de Iteraciones con 10000

```
if __name__ == '__main__':
    red_neuronal = RedNeuronal()
    entradas = array([[0,0,1], [1,1,1], [1,0,1], [0,1,1]])
    salidas = array([[0,1,1,0]]).T
    red_neuronal.entrenamiento(entradas,salidas,10000)
    print(red_neuronal.pesos_signaticos)
    print(red_neuronal.pensar(array([1,0,0])))

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[[ 9.67329334]
 [-0.20789393]
 [-4.62975692]]
[0.99993706]
Tiempo total:  0.13962388038635254
```

Gráfica:

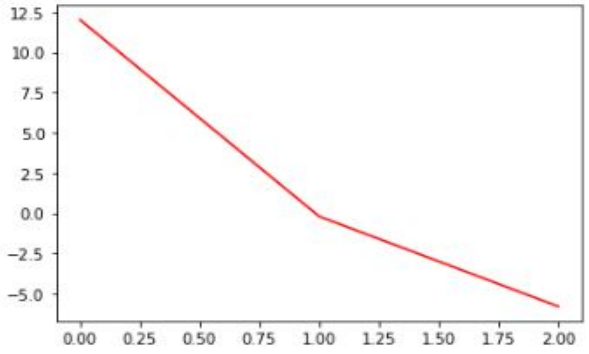


Numero de Iteraciones con 100000

```
if __name__ == '__main__':
    red_neuronal = RedNeuronal()
    entradas = array([[0,0,1], [1,1,1], [1,0,1], [0,1,1]])
    salidas = array([[0,1,1,0]]).T
    red_neuronal.entrenamiento(entradas,salidas,100000)
    print(red_neuronal.pesos_signaticos)
    print(red_neuronal.pensar(array([1,0,0])))

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[[12.00866243]
 [-0.20442235]
 [-5.80025669]]
[0.99999391]
Tiempo total:  1.2548887729644775
```

Gráfica:



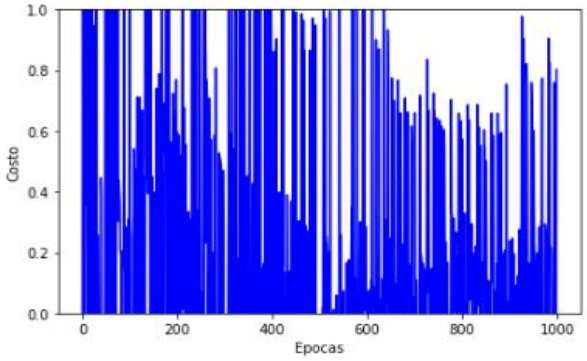
Red Neuronal Múltiple

Numero de Iteraciones con 1000

```
nn.ajuste(X, y, factor_aprendizaje = 0.03, epocas = 1000
          )

index = 0
for e in X:
    print("X: ", e, "y: ", y[index], "Red: ", nn.predecir(e))
    index = index + 1

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
epocas: 0
X:  [0. 0.] y:  [0 1] Red:  [-0.05322221  0.99003666]
X:  [0. 1.] y:  [0 1] Red:  [-0.32741772  0.98290567]
X:  [ 0. -1.] y:  [0 1] Red:  [0.31499211 0.98985353]
X:  [0.5 1. ] y:  [-1  1] Red:  [-0.31696527  0.74182165]
X:  [ 0.5 -1. ] y:  [1 1] Red:  [0.45329099 0.75510605]
X:  [1. 1.] y:  [ 0 -1] Red:  [-0.23858508 -0.8599816 ]
X:  [ 1. -1.] y:  [ 0 -1] Red:  [ 0.31442049 -0.84635732]
Tiempo total:  0.5907208919525146
```

Gráfica:

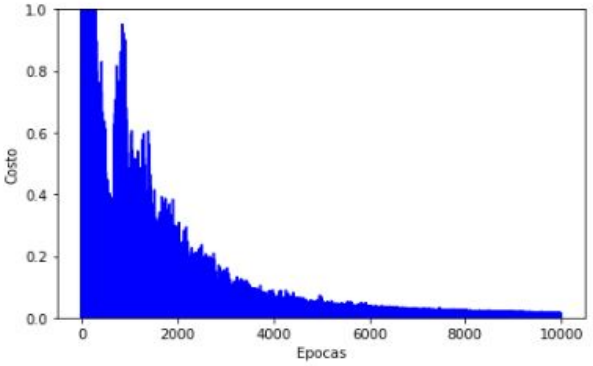


Numero de Iteraciones con 10000

```
nn.ajuste(X, y, factor_aprendizaje = 0.03, epocas = 10000)

index = 0
for e in X:
    print("X: ", e, "y: ", y[index], "Red: ", nn.predecir(e))
    index = index + 1

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
epocas: 0
X:  [0. 0.] y:  [0 1] Red:  [-0.01106809  0.99997211]
X:  [0. 1.] y:  [0 1] Red:  [-0.00294145  0.99998063]
X:  [ 0. -1.] y:  [0 1] Red:  [0.00309187 0.99968204]
X:  [0.5 1. ] y:  [-1  1] Red:  [-0.88674341  0.91785414]
X:  [ 0.5 -1. ] y:  [1 1] Red:  [0.93592421 0.96085307]
X:  [1. 1.] y:  [ 0 -1] Red:  [-0.02614836 -0.92857287]
X:  [ 1. -1.] y:  [ 0 -1] Red:  [ 0.00978891 -0.99125839]
Tiempo total:  104.39417934417725
```

Gráfica:

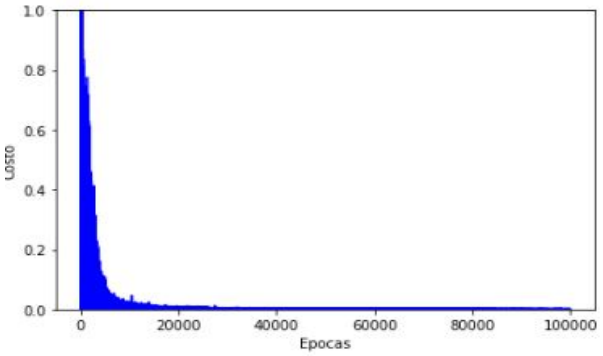


Numero de Iteraciones con 100000

```
nn.ajuste(X, y, factor_aprendizaje = 0.03, epocas = 100000
          )

index = 0
for e in X:
    print("X: ", e, "y: ", y[index], "Red: ", nn.predecir(e))
    index = index + 1

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
epocas: 0
epocas: 10000
epocas: 20000
epocas: 30000
epocas: 40000
epocas: 50000
epocas: 60000
epocas: 70000
epocas: 80000
epocas: 90000
X:  [0. 0.] y:  [0 1] Red:  [-9.85643715e-05  9.99996779e-01]
X:  [0. 1.] y:  [0 1] Red:  [-9.90204785e-05  9.99979096e-01]
X:  [ 0. -1.] y:  [0 1] Red:  [1.25032131e-04 9.99996496e-01]
X:  [0.5 1. ] y:  [-1  1] Red:  [-0.98588563  0.9857546 ]
X:  [ 0.5 -1. ] y:  [1 1] Red:  [0.98491095 0.98564526]
X:  [1. 1.] y:  [ 0 -1] Red:  [-2.15242390e-04 -9.91406833e-01]
X:  [ 1. -1.] y:  [ 0 -1] Red:  [ 2.25262163e-04 -9.91107224e-01]
Tiempo total:  4.262856960296631
```

Gráfica:



Iteración del equipo 2 :Elkin Ramirez
Descripción Acer aspire ES1-572, memoria ram: 8192MB, Intel Core i3 2.50Ghz, sistema operativo Win 10 64Bits

Perceptron

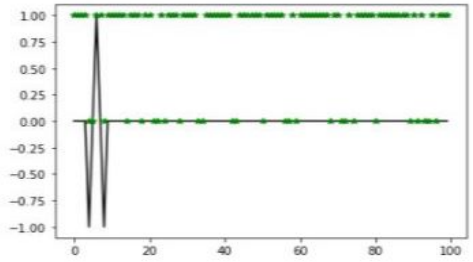
Numero de Iteraciones con 100

```
print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.15507007509673681 -> 0
[0 1 1]: 0.17467384795497054 -> 1
[1 0 1]: 0.20488776681274784 -> 1
[1 1 1]: 0.5346316898644552 -> 1
Tiempo total:  0.5698540210723877
```

```
.2]: plt.plot(errores,'-',color='black')
     plt.plot(esperados,'*', color='green')
```

```
.2]: [<matplotlib.lines.Line2D at 0x2705cfcca58>]
```



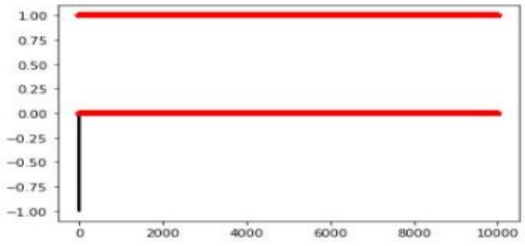
Numero de Iteraciones con 10000

```
tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.15188522811069177 -> 0
[0 1 1]: 0.40678283087210304 -> 1
[1 0 1]: 0.5215284182091433 -> 1
[1 1 1]: 1.0801964771919383 -> 1
Tiempo total:  1.316603422164917
```

```
]: plt.plot(errores,'-',color='black')
   plt.plot(esperados,'*', color='red')
```

```
]: [<matplotlib.lines.Line2D at 0x1814b113438>]
```



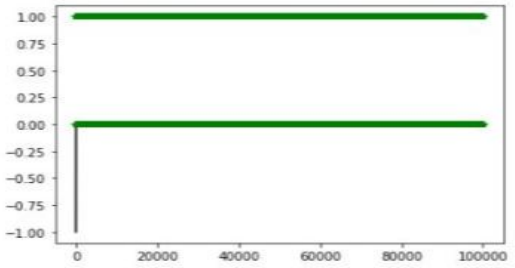
Numero de Iteraciones con 100000

```
print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.10171094969730021 -> 0
[0 1 1]: 0.48209908513820454 -> 1
[1 0 1]: 0.03932960344571168 -> 1
[1 1 1]: 0.6231396382812164 -> 1
Tiempo total:  1.0405185222625732
```

```
[7]: plt.plot(errores,'-',color='black')
     plt.plot(esperados,'*', color='green')
```

```
[7]: [<matplotlib.lines.Line2D at 0x2705c663470>]
```



Red Neuronal Simple

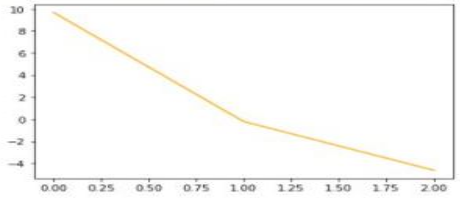
Numero de Iteraciones con 10000

```
    red_neuronal.entrenamiento(entradas,salidas,10000)
    print(red_neuronal.pesos_signaticos)
    print(red_neuronal.pensar(array([1,0,0])))

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[[ 9.67274479]
 [-0.20868013]
 [-4.62901336]]
[0.99993703]
Tiempo total:  0.1585419178009033
```

```
]: plt.plot(red_neuronal.pesos_signaticos,'-',color='orange')
   #plt.plot(salida,'*',color='red')
```

```
]: [<matplotlib.lines.Line2D at 0x2039a1b9a20>]
```



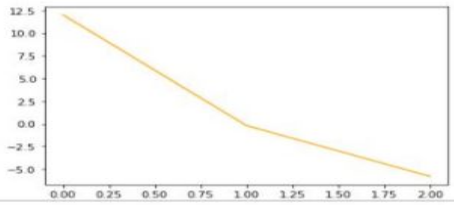
Numero de Iteraciones con 100000

```
    salidas = array([[0,1,1,0]]).T
    red_neuronal.entrenamiento(entradas,salidas,100000)
    print(red_neuronal.pesos_signaticos)
    print(red_neuronal.pensar(array([1,0,0])))

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[[12.00869918]
 [-0.2044433 ]
 [-5.80026258]]
[0.99999391]
Tiempo total:  1.5884432792663574
```

```
plt.plot(red_neuronal.pesos_signaticos,'-',color='orange')
#plt.plot(salida,'*',color='red')
```

```
[<matplotlib.lines.Line2D at 0x1af94a51208>]
```



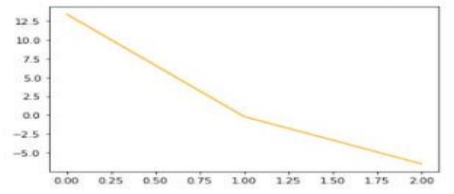
Numero de Iteraciones con 400000

```
    red_neuronal = RedNeuronal()
    entradas = array([[0,0,1], [1,1,1], [1,0,1], [0,1,1]])
    salidas = array([[0,1,1,0]]).T
    red_neuronal.entrenamiento(entradas,salidas,400000)
    print(red_neuronal.pesos_signaticos)
    print(red_neuronal.pensar(array([1,0,0])))

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
[[13.40253743]
 [-0.2036065 ]
 [-6.49784866]]
[0.99999849]
Tiempo total:  6.629265785217285
```

```
plt.plot(red_neuronal.pesos_signaticos,'-',color='orange')
#plt.plot(salida,'*',color='red')
```

```
[<matplotlib.lines.Line2D at 0x22ea3800278>]
```



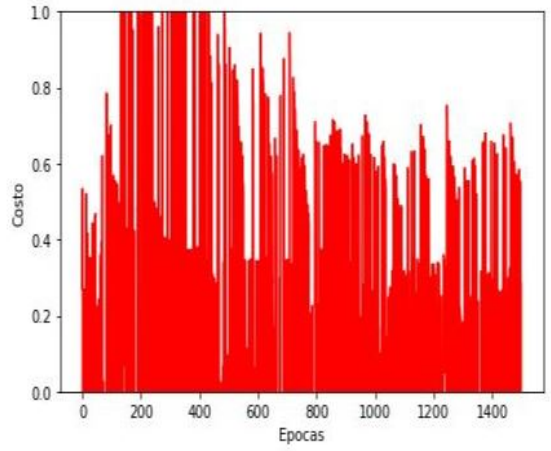
Red Neuronal Múltiple

Numero de Iteraciones con 150000

```
            [0, -1],]) #retroceder
nn.ajuste(X, y, factor_aprendizaje = 0.03, epocas = 150000)

index = 0
for e in X:
    print("X: ", e, "y: ", y[index], "Red: ", nn.predecir(e))
    index = index + 1

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
epocas: 30000
epocas: 40000
epocas: 50000
epocas: 60000
epocas: 70000
epocas: 80000
epocas: 90000
epocas: 100000
epocas: 110000
epocas: 120000
epocas: 130000
epocas: 140000
X:  [0. 0.] y:  [0 1] Red:  [1.90289920e-04 9.99999646e-01]
X:  [0. 1.] y:  [0 1] Red:  [-3.97716794e-04  9.99999388e-01]
X:  [ 0. -1.] y:  [0 1] Red:  [8.32535033e-05 9.99997332e-01]
X:  [0.5 1. ] y:  [-1  1] Red:  [-0.98466678  0.98780246]
X:  [ 0.5 -1. ] y:  [1 1] Red:  [0.99131881 0.99190758]
X:  [1. 1.] y:  [ 0 -1] Red:  [-1.09618012e-04 -9.89788344e-01]
X:  [ 1. -1.] y:  [ 0 -1] Red:  [ 1.48709859e-04 -9.94803026e-01]
Tiempo total:  9.888447999954224
```

Gráfica:

```
plt.show()
```

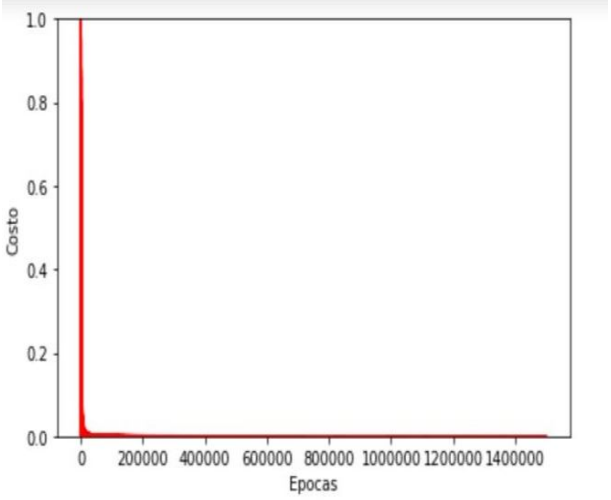


Numero de Iteraciones con 1500000

```
nn.ajuste(X, y, factor_aprendizaje = 0.03, epocas = 1500000)

index = 0
for e in X:
    print("X: ", e, "y: ", y[index], "Red: ", nn.predecir(e))
    index = index + 1

tiempo_final = time()
tiempo_ejecucion= tiempo_final - inicia_tiempo

print("Tiempo total: ", tiempo_ejecucion)
```

```
epocas: 1380000
epocas: 1390000
epocas: 1400000
epocas: 1410000
epocas: 1420000
epocas: 1430000
epocas: 1440000
epocas: 1450000
epocas: 1460000
epocas: 1470000
epocas: 1480000
epocas: 1490000
X:  [0. 0.] y:  [0 1] Red:  [1.18552192e-05 9.99999979e-01]
X:  [0. 1.] y:  [0 1] Red:  [1.94404579e-05 9.99999735e-01]
X:  [ 0. -1.] y:  [0 1] Red:  [2.20254351e-05 9.99999977e-01]
X:  [0.5 1. ] y:  [-1  1] Red:  [-0.99876538  0.99637952]
X:  [ 0.5 -1. ] y:  [1 1] Red:  [0.99499934 0.99604346]
X:  [1. 1.] y:  [ 0 -1] Red:  [-1.74592190e-05 -9.98100892e-01]
X:  [ 1. -1.] y:  [ 0 -1] Red:  [ 4.82132434e-05 -9.96454983e-01]
Tiempo total:  70.34096717834473
```

Gráfica:



Iteración del equipo 3 : Juan Felipe Marin

Descripción Acer Aspire n214, memoria ram: 4GB, Intel Core i3 1.7Ghz, sistema operativo Win 10 64Bits

Perceptron

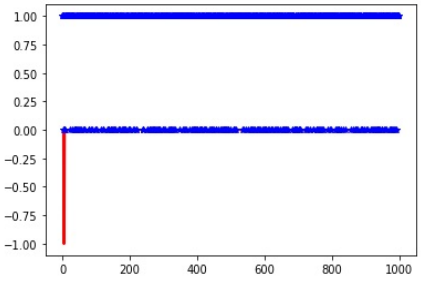
Numero de Iteraciones con 1000

```
print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.005474775076665439 -> 0
[0 1 1]: 0.11742806922048449 -> 1
[1 0 1]: 0.4310591058863214 -> 1
[1 1 1]: 0.5539619501834714 -> 1
Tiempo total:  0.5311741828918457
```

[29]:
```
plt.plot(errores,'-',color='red')
plt.plot(esperados,'*', color='blue')
```

[29]:
```
[<matplotlib.lines.Line2D at 0x5ef2890>]
```



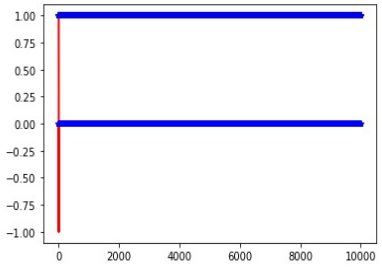
Numero de Iteraciones con 10000

```
print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.13841640031789404 -> 0
[0 1 1]: 0.11314612075294361 -> 1
[1 0 1]: 0.10837860026087581 -> 1
[1 1 1]: 0.35994112133171347 -> 1
Tiempo total:  0.9965085983276367
```

[22]:
```
plt.plot(errores,'-',color='red')
plt.plot(esperados,'*', color='blue')
```

[22]:
```
[<matplotlib.lines.Line2D at 0xb347ad0>]
```



[ ]:

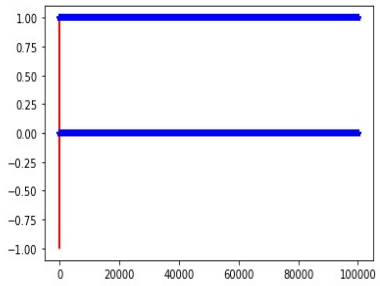
Numero de Iteraciones con 100000

```
print("Tiempo total: ", tiempo_ejecucion)
```

```
[0 0 1]: -0.11808585081977896 -> 0
[0 1 1]: 0.12463973993332517 -> 1
[1 0 1]: 0.6533470559593109 -> 1
[1 1 1]: 0.896072646712415 -> 1
Tiempo total:  1.718632459640503
```

[36]:
```
plt.plot(errores,'-',color='red')
plt.plot(esperados,'*', color='blue')
```

[36]: `[<matplotlib.lines.Line2D at 0x5f403f0>]`



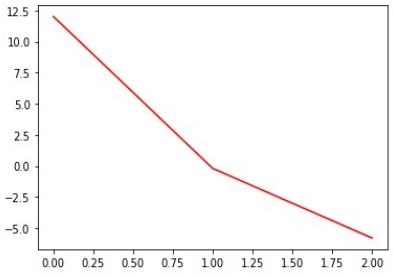
Red Neuronal Simple

Numero de Iteraciones con 1000

```
[[ 7.26683259]
 [-0.22293216]
 [-3.41498595]]
[0.99930217]
Tiempo total:  0.05794715881347656
```

[18]:
```
plt.plot(red_neuronal.pesos_signaticos,'-',color='red')
#plt.plot(salida,'*',color='red')
```

[18]: `[<matplotlib.lines.Line2D at 0x4e30050>]`

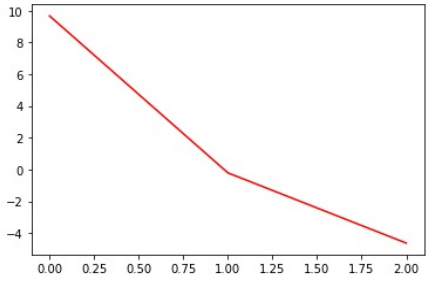


Numero de Iteraciones con 10000

```
[[ 9.6725773 ]
 [-0.2083743 ]
 [-4.62911195]]
[0.99993702]
Tiempo total:  0.5198686122894287
```

```
[3]: plt.plot(red_neuronal.pesos_signaticos,'-',color='red')
     #plt.plot(salida,'*',color='red')
```

```
[3]: [<matplotlib.lines.Line2D at 0xabc4250>]
```

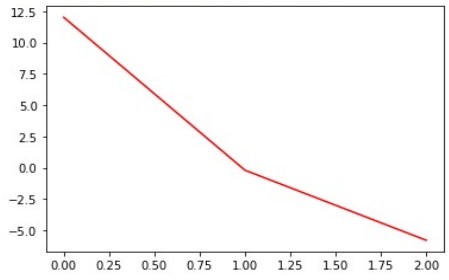


Numero de Iteraciones con 100000

```
[[12.00872317]
 [-0.20447099]
 [-5.80025807]]
[0.99999391]
Tiempo total:  3.730786085128784
```

```
[6]: plt.plot(red_neuronal.pesos_signaticos,'-',color='red')
     #plt.plot(salida,'*',color='red')
```

```
[6]: [<matplotlib.lines.Line2D at 0xac415f0>]
```



Red Neuronal Múltiple

Numero de Iteraciones con 1000

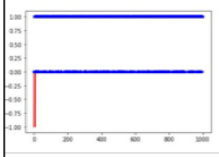
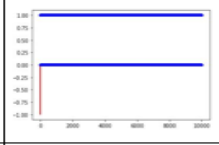
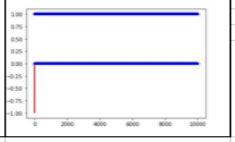
Numero de Iteraciones con 10000

Numero de Iteraciones con 100000

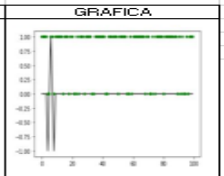
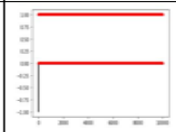
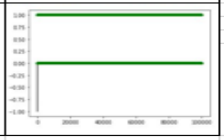
Evaluación de los resultados de los algoritmos implementados

Perceptron

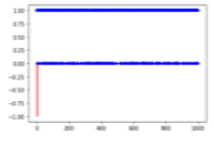
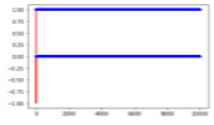
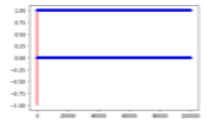
PERCEPTRON

ENTRADAS	SALIDAS	EPOCAS	F. APRENDIZAJE	RESULTADOS	TIEMPO	GRAFICA
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	1000	0,03	Tiene algunos errores con este numero de iteracciones eneste tiempo de 1,035 segundos Conclusión debería ser entrenado con un poco más de iteraciones para mejorar el ajuste de errores	1,03534	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	10000	0,03	A comparación de 1000 iteraciones el algoritmo de ajusta más a los resultados esperados Este perceptron tiene un mejor entrenamiento al cual está más asemejado a los resultados esperados.	0,86166	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100000	0,03	Este perceptron esta muy ajustado al resultado que se espera con este numero de iteracciones.	0,96896	

PERCEPTRON

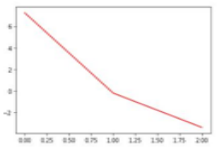
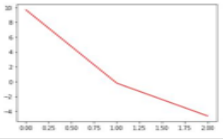
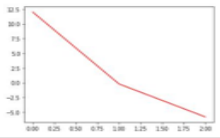
ENTRADAS	SALIDAS	EPOCAS	F. APRENDIZAJE	RESULTADOS	TIEMPO	GRAFICA
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100	0,03	El tiempo de ejecución es corto tan solo es de 569 milisegundos o 0.596 segundos para un entrenamiento realmente corto como lo es 100 iteraciones. Conclusión debería ser entrenado con más iteraciones para mejorar el ajuste de errores.	0,56985	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	10000	0,03	A comparación de 100 iteraciones el algoritmo de ajusta más a los resultados esperados que la de 100 iteraciones. Este perceptron tiene un mejore entrenamiento al cual está más asemejado a los resultados esperados.	1,3166	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100000	0,03	Conclusión ajuste muy adecuado respecto a los datos esperados (Salidas)	1,0405	

PERCEPTRON

ENTRADAS	SALIDAS	EPOCAS	F. APRENDIZAJE	RESULTADOS	TIEMPO	GRAFICA
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	1000	0,03	Tiene algunos errores con este numero de iteracciones eneste tiempo de 0,83 segundos Conclusión debería ser entrenado con un poco más de iteraciones para mejorar el ajuste de errores	0,83021	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	10000	0,03	A comparación de 1000 iteraciones el algoritmo de ajusta más a los resultados esperados Este perceptron tiene un mejor entrenamiento al cual está más asemejado a los resultados esperados.	0,9965	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100000	0,03	Este perceptron esta muy ajustado al resultado que se espera con este numero de iteracciones.	1,71863	

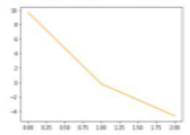
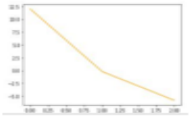
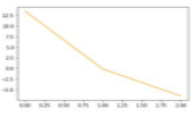
Red Neuronal Simple

RED NEURONAL SIMPLE

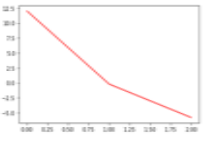
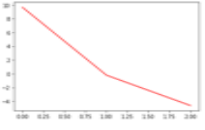
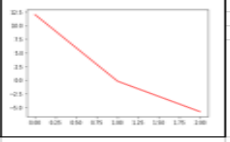
ENTRADAS	SALIDAS	EPOCAS	F. APRENDIZAJE	RESULTADOS	TIEMPO	GRAFICA
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	1000	0,03	Arrojando un tiempo de compilación de 0.03 milisegundos. Podemos ver que es un tiempo muy corto debido a su simplicidad en la red neuronal Los resultados que arroja no son los mejores ni los más preciso para nuestro trabajo	0,0309	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	10000	0,03	Arrojando un tiempo de compilación de 0.13 milisegundos. Podemos ver que mejora en los resultados pero no se nota mucho la diferencia a comparacion de 1000 iteraciones.	0,1396	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100000	0,03	Ha mejorado mucho en los resultados, es mucho mayor en tiempo pero no se llega a un resultado que se espera. Nota: con un poco mas de iteracciones se ve mas el cambio.	1,2548	

EQUIPO ELKIN

RED NEURONAL SIMPLE

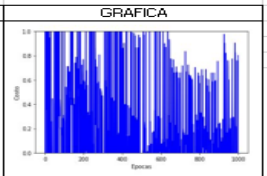
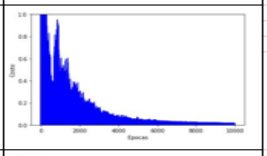
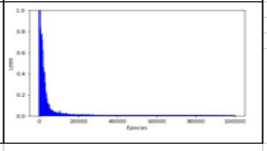
ENTRADA	SALIDAS	EPOCAS	F. APRENDIZA	RESULTADOS	TIEMPO	GRAFICAS
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	10000	0,03	Arrojando un tiempo de compilación de 0.15 milisegundos. Podemos ver que es un tiempo muy corto debido a su simplicidad en la red neuronal Los resultados que arroja no son los mejores ni los más preciso para nuestro trabajo	0,15854	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100000	0,03	podríamos decir que cambia un poco respecto a el entrenamiento anterior. Puesto que el tiempo ha aumentado notoriamente. Pero los resultados siguen estando poco ajustados a lo esperado.	1,58844	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	400000	0,03	ha mejorado se ajusta más al resultado. Y notoriamente el tiempo de manejo y aprendizaje es mucho mayor a las 2 anteriores.	6,62926	

RED NEURONAL SIMPLE

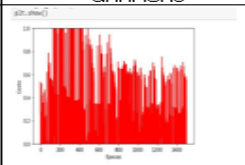
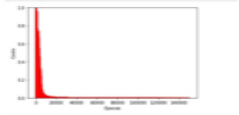
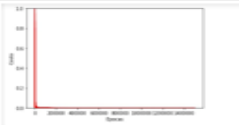
ENTRADA	SALIDAS	EPOCAS	F. APRENDIZA	RESULTADOS	TIEMPO	GRAFICA
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	1000	0,03	Arrojando un tiempo de compilación de 0.05 milisegundos. Podemos ver que es un tiempo muy corto debido a su simplicidad en la red neuronal Los resultados que arroja no son los mejores ni los más preciso para nuestro trabajo	0,0579	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	10000	0,03	Arrojando un tiempo de compilación de 0.413 milisegundos. Podemos ver que mejora en los resultados pero no se nota mucho la diferencia a comparacion de 1000 iteraciones.	0,41913	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100000	0,03	Ha mejorado mucho en los resultados, es mucho mayor en tiempo pero no se llega a un resultado que se espera. Nota: con un poco mas de iteracciones se ve mas el cambio.		

Red Neuronal Múltiple

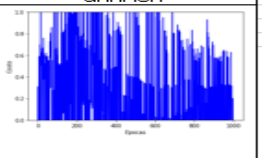
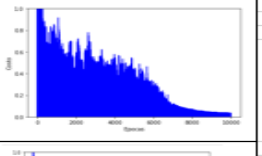
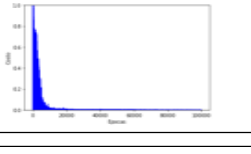
RED NEURONAL MULTICAPA

ENTRADAS	SALIDAS	EPOCAS	F. APRENDIZA	RESULTADOS	TIEMPO	GRAFICA
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	1000	0,03	Aun con 1000 épocas la neurona aún no se ajusta con claridad a un margen de error aceptable o claro para el ejercicio. Llegando a concluir que entre nas epocas mejor era el resultado esperado.	0,5907	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	10000	0,03	Ya con 10000 epocas se ve mucho mas ajustado al resultado en comparacion con las 1000 epocas anteriormente.	104,394	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100000	0,03	Con 100000 iteracciones la grafica se ve cada vez mas ajustado al resultado, quiere decir entre mas entrenamiento tenga la neurona el resultado sera cada vez mejor.	4,2625	

RED NEURONAL MULTICAPA

ENTRADA	SALIDAS	EPOCAS	F. APRENDIZ	RESULTADOS	TIEMPO	GRAFICAS
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	1500	0,03	Aun con 1500 épocas la neurona aún no se ajusta con claridad a un margen de error aceptable o claro para el ejercicio. Llegando a concluir que para una red neuronal tangente como la que estamos trabajando es mar recomendable muchas más épocas de ejercicio o entrenamiento para la misma.	1,545	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	150000	0,03	Cuando el entrenamiento es más extenso ya comenzamos a ver cambios muy significativos en el proceso de aprendizaje de la neurona, puesto que empieza a reconocer en mejor medida y de una forma óptima los resultados esperados y el margen de error.	9,88844	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	1500000	0,03	Ya cuando hablamos de un entrenamiento de más de un millón de épocas los resultados se afirman o afianzan más al resultado esperado. En la gráfica podríamos decir que cumple con un resultado esperado, gracias a su entrenamiento.	70,3409	

RED NEURONAL MULTICAPA

ENTRADA	SALIDAS	EPOCAS	F. APRENDIZ	RESULTADOS	TIEMPO	GRAFICA
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	1000	0,03	Aun con 1000 épocas la neurona aún no se ajusta con claridad a un margen de error aceptable o claro para el ejercicio. Llegando a concluir que entre nas epocas mejor era el resultado esperado.	0,7198	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	10000	0,03	Ya con 10000 epocas se ve mucho mas ajustado al resultado en comparacion con las 1000 epocas anteriormente, sin embargo todavia esta muy lejos del resultado que se espera.	1,7792	
0,0,1 1,1,1 1,0,1 0,1,1	0 1 1 0	100000	0,03	Con 100000 iteracciones la grafica se ve cada vez mas ajustado al resultado, quiere decir entre mas entrenamiento tenga la neurona el resultado sera cada vez mejor.	11,5874	

Computadores usados

Iteración del equipo 1 : Lorena Sanchez
Descripción Lenovo yoga 510, memoria ram: 8192MB, Intel Core i5 2.30Ghz, sistema operativo Win 10 64Bits

Iteración del equipo 2 :Elkin Ramirez
Descripción Acer aspire ES1-572, memoria ram: 8192MB, Intel Core i3 2.50Ghz, sistema operativo Win 10 64Bits

Iteración del equipo 3 : Juan Felipe Marin
Descripción Acer Aspire n214, memoria ram: 4GB, Intel Core i3 1.7Ghz, sistema operativo Win 10 64Bits

Link Github-Page
https://eknramirez.github.io/

Referencia bibliográficas

https://searchdatacenter.techtarget.com/es/definicion/Inteligencia-artificial-o-AI

https://desarrolloweb.com/articulos/1325.php